Colour Documentation

Release 0.3.11

Colour Developers

Feb 18, 2018

CONTENTS

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

It is open source and freely available under the New BSD License terms.

CHAPTER

ONE

FEATURES

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

CHAPTER

TWO

INSTALLATION

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

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

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

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

The detailed installation procedure is described in the Installation Guide.

CHAPTER

THREE

USAGE

The two main references for Colour usage are the Colour Manual and the Jupyter Notebooks with detailed historical and theoretical context and images.

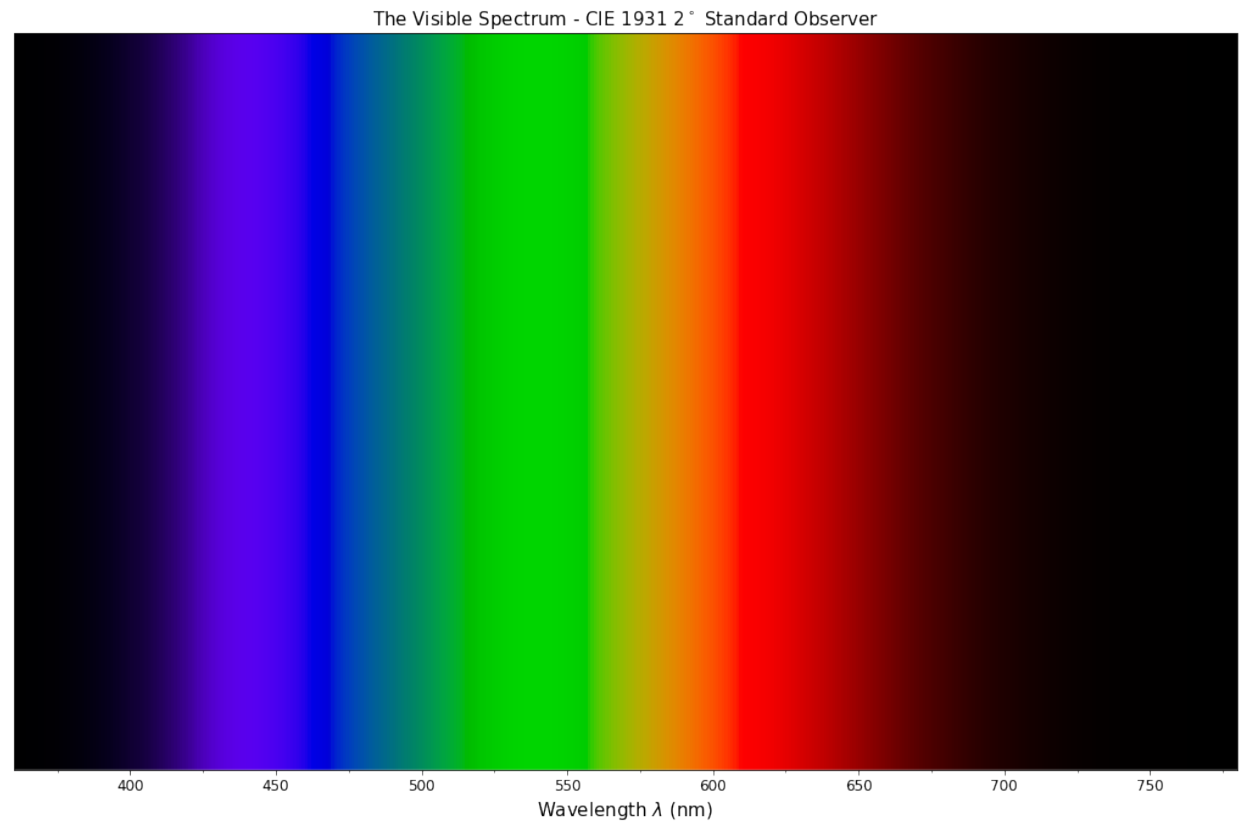
3.1 Colour Manual

3.1.1 Tutorial

Colour spreads over various domains of Colour Science from colour models to optical phenomena, this tutorial will not give you a complete overview of the API but will still be a good introduction.

Note: A directory full of examples is available at this path in your Colour installation: *colour/examples*. You can also explore it directly on Github: https://github.com/colour-science/colour/tree/master/colour/examples

```
from colour.plotting import *

colour_plotting_defaults()

visible_spectrum_plot()
```



3.1.1.1 Overview

Colour is organised around various sub-packages:

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

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

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

```
import colour

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

```
['handle_numpy_errors', 'ignore_numpy_errors', 'raise_numpy_errors', 'print_numpy_errors', 'warn_numpy_
↪errors', '...']
```

The various sub-packages also expose their public API:

```
from pprint import pprint

import colour.plotting

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

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


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


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

```
 '...']


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


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


Colorimetry
['SpectralShape',
 'SpectralPowerDistribution',
 'MultiSpectralPowerDistribution',
 'DEFAULT_SPECTRAL_SHAPE',
 'constant_spd',
 '...']


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


Constants
['CIE_E', 'CIE_K', 'K_M', 'KP_M', 'AVOGADRO_CONSTANT', '...']


Corresponding
['BRENEMAN_EXPERIMENTS',
 'BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES',
 'corresponding_chromaticities_prediction_CIE1994',
 'corresponding_chromaticities_prediction_CMCCAT2000',
 'corresponding_chromaticities_prediction_Fairchild1990',
 '...']


Difference
['DELTA_E_METHODS',
 'delta_E',
 'delta_E_CIE1976',
 'delta_E_CIE1994',
 'delta_E_CIE2000',
 '...']


Io
['IES_TM2714_Spd',
 'read_image',
 'write_image',
 'read_spectral_data_from_csv_file',
```

```
 'read_spds_from_csv_file',
 '...']


Models
['XYZ_to_xyY', 'xyY_to_XYZ', 'xy_to_xyY', 'xyY_to_xy', 'xy_to_XYZ', '...']


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


Phenomena
['scattering_cross_section',
 'rayleigh_optical_depth',
 'rayleigh_scattering',
 'rayleigh_scattering_spd',
 '...']


Plotting
['ASTM_G_173_ETR',
 'PLOTTING_RESOURCES_DIRECTORY',
 'DEFAULT_FIGURE_ASPECT_RATIO',
 'DEFAULT_FIGURE_WIDTH',
 'DEFAULT_FIGURE_HEIGHT',
 '...']


Quality
['TCS_SPDS',
 'VS_SPDS',
 'CRI_Specification',
 'colour_rendering_index',
 'CQS_Specification',
 '...']


Recovery
['SMITS_1999_SPDS',
 'XYZ_to_spectral_Meng2015',
 'RGB_to_spectral_Smits1999',
 'REFLECTANCE_RECOVERY_METHODS',
 'XYZ_to_spectral',
 '...']


Temperature
['CCT_TO_UV_METHODS',
 'UV_TO_CCT_METHODS',
 'CCT_to_uv',
 'CCT_to_uv_Ohno2013',
 'CCT_to_uv_Robertson1968',
```

```
 '...']


Utilities
['handle_numpy_errors',
 'ignore_numpy_errors',
 'raise_numpy_errors',
 'print_numpy_errors',
 'warn_numpy_errors',
 '...']


Volume
['ILLUMINANTS_OPTIMAL_COLOUR_STIMULI',
 'is_within_macadam_limits',
 'is_within_mesh_volume',
 'is_within_pointer_gamut',
 'is_within_visible_spectrum',
 '...']
```

The code is documented and almost every 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 : numeric
    Correlated colour temperature :math:`T_{cp}`.
D_uv : numeric, optional
    :math:`\Delta_{uv}`.
cmfs : XYZ_ColourMatchingFunctions, optional
    Standard observer colour matching functions.

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

References
----------
.. [4]  Ohno, Y. (2014). Practical Use and Calculation of CCT and Duv.
        LEUKOS, 10(1), 4755. doi:10.1080/15502724.2014.839020

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

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

```python
import colour.colorimetry as colorimetry

pprint(colorimetry.__all__)
```

```
['SpectralShape',
 'SpectralPowerDistribution',
 'MultiSpectralPowerDistribution',
 'DEFAULT_SPECTRAL_SHAPE',
 'constant_spd',
 'zeros_spd',
 'ones_spd',
 'blackbody_spd',
 'blackbody_spectral_radiance',
 'planck_law',
 'LMS_ConeFundamentals',
 'RGB_ColourMatchingFunctions',
 'XYZ_ColourMatchingFunctions',
 'CMFS',
 'LMS_CMFS',
 'RGB_CMFS',
 'STANDARD_OBSERVERS_CMFS',
 'ILLUMINANTS',
 'D_ILLUMINANTS_S_SPDS',
 'HUNTERLAB_ILLUMINANTS',
 'ILLUMINANTS_RELATIVE_SPDS',
 'LIGHT_SOURCES',
 'LIGHT_SOURCES_RELATIVE_SPDS',
 'LEFS',
 'PHOTOPIC_LEFS',
 'SCOTOPIC_LEFS',
 'BANDPASS_CORRECTION_METHODS',
 'bandpass_correction',
 'bandpass_correction_Stearns1988',
 'D_illuminant_relative_spd',
 'CIE_standard_illuminant_A_function',
 'mesopic_luminous_efficiency_function',
 'mesopic_weighting_function',
 'LIGHTNESS_METHODS',
 'lightness',
 'lightness_Glasser1958',
 'lightness_Wyszecki1963',
 'lightness_CIE1976',
 'lightness_Fairchild2010',
 'lightness_Fairchild2011',
 'LUMINANCE_METHODS',
 'luminance',
 'luminance_Newhall1943',
 'luminance_ASTMD153508',
 'luminance_CIE1976',
 'luminance_Fairchild2010',
 'luminance_Fairchild2011',
 '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',
 'LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs',
 'SPECTRAL_TO_XYZ_METHODS',
 'spectral_to_XYZ',
 'ASTME30815_PRACTISE_SHAPE',
 'lagrange_coefficients_ASTME202211',
 'tristimulus_weighting_factors_ASTME202211',
 'adjust_tristimulus_weighting_factors_ASTME30815',
 'spectral_to_XYZ_integration',
 'spectral_to_XYZ_tristimulus_weighting_factors_ASTME30815',
 'spectral_to_XYZ_ASTME30815',
 'wavelength_to_XYZ',
 'WHITENESS_METHODS',
 'whiteness',
 'whiteness_Berger1959',
 'whiteness_Taube1960',
 'whiteness_Stensby1968',
 'whiteness_ASTME313',
 'whiteness_Ganz1979',
 'whiteness_CIE2004',
 'YELLOWNESS_METHODS',
 'yellowness',
 'yellowness_ASTMD1925',
 'yellowness_ASTME313']
```

Colour computations leverage a comprehensive dataset available in pretty much each sub-packages, for example `colour.colorimetry.dataset` defines the following data:

```
import colour.colorimetry.dataset as dataset

pprint(dataset.__all__)
```

```
['CMFS',
 'LMS_CMFS',
 'RGB_CMFS',
 'STANDARD_OBSERVERS_CMFS',
 'ILLUMINANTS',
 'D_ILLUMINANTS_S_SPDS',
 'HUNTERLAB_ILLUMINANTS',
 'ILLUMINANTS_RELATIVE_SPDS',
 'LIGHT_SOURCES',
 'LIGHT_SOURCES_RELATIVE_SPDS',
 'LEFS',
 'PHOTOPIC_LEFS',
 'SCOTOPIC_LEFS']
```

3.1.1.2 From Spectral Power Distribution

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

```
# Defining a sample spectral power distribution data.
sample_spd_data = {
```

```
    380: 0.048,
    385: 0.051,
    390: 0.055,
    395: 0.060,
    400: 0.065,
    405: 0.068,
    410: 0.068,
    415: 0.067,
    420: 0.064,
    425: 0.062,
    430: 0.059,
    435: 0.057,
    440: 0.055,
    445: 0.054,
    450: 0.053,
    455: 0.053,
    460: 0.052,
    465: 0.052,
    470: 0.052,
    475: 0.053,
    480: 0.054,
    485: 0.055,
    490: 0.057,
    495: 0.059,
    500: 0.061,
    505: 0.062,
    510: 0.065,
    515: 0.067,
    520: 0.070,
    525: 0.072,
    530: 0.074,
    535: 0.075,
    540: 0.076,
    545: 0.078,
    550: 0.079,
    555: 0.082,
    560: 0.087,
    565: 0.092,
    570: 0.100,
    575: 0.107,
    580: 0.115,
    585: 0.122,
    590: 0.129,
    595: 0.134,
    600: 0.138,
    605: 0.142,
    610: 0.146,
    615: 0.150,
    620: 0.154,
    625: 0.158,
    630: 0.163,
    635: 0.167,
    640: 0.173,
    645: 0.180,
    650: 0.188,
    655: 0.196,
    660: 0.204,
    665: 0.213,
```

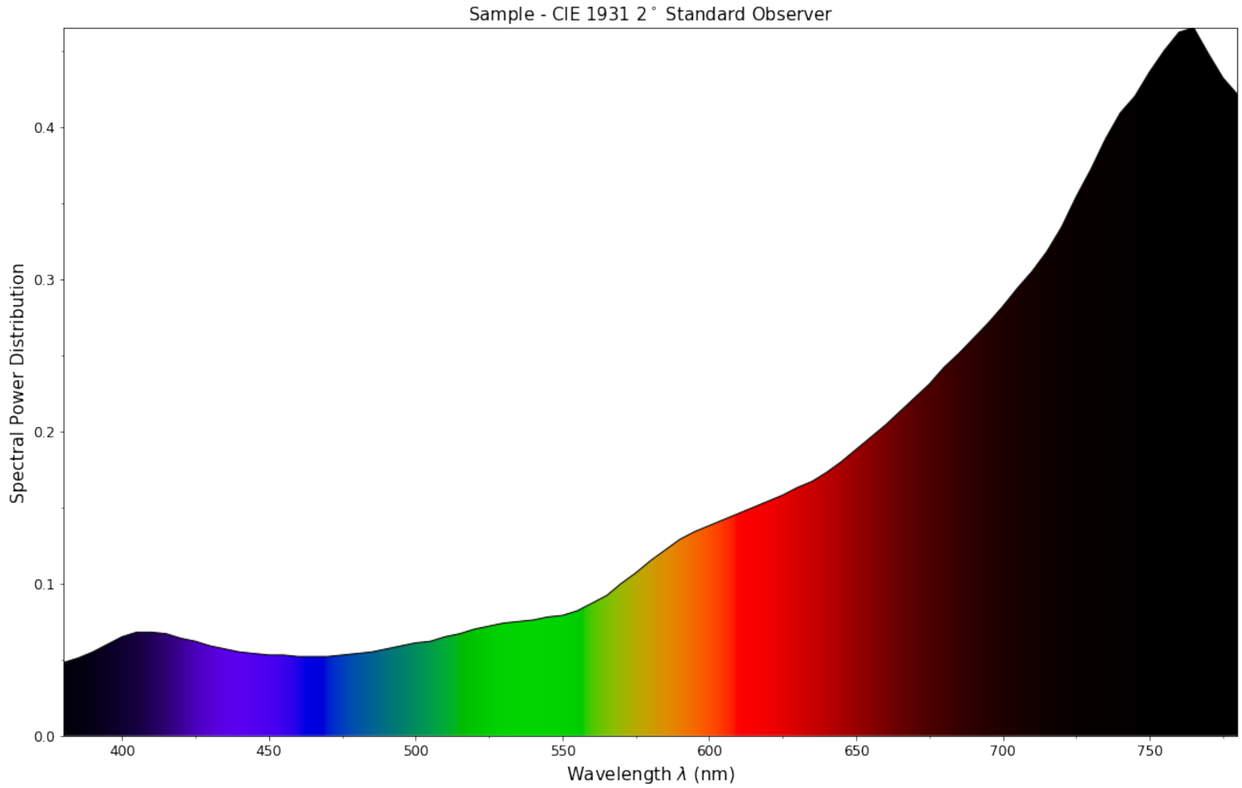
```
    670: 0.222,
    675: 0.231,
    680: 0.242,
    685: 0.251,
    690: 0.261,
    695: 0.271,
    700: 0.282,
    705: 0.294,
    710: 0.305,
    715: 0.318,
    720: 0.334,
    725: 0.354,
    730: 0.372,
    735: 0.392,
    740: 0.409,
    745: 0.420,
    750: 0.436,
    755: 0.450,
    760: 0.462,
    765: 0.465,
    770: 0.448,
    775: 0.432,
    780: 0.421}

spd = colour.SpectralPowerDistribution(sample_spd_data, name='Sample')
print(repr(spd))
```

```
SpectralPowerDistribution([[  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],
                                [  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 power distribution can be easily plotted against the visible spectrum:

```
# Plotting the sample spectral power distribution.
single_spd_plot(spd)
```



With the sample spectral power distribution defined, we can retrieve its shape:

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

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

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

```
repr(spd.shape)
```

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

`colour.SpectralShape` 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 power distributions:

- `colour.constant_spd`
- `colour.zeros_spd`
- `colour.ones_spd`

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

# Defining a zeros filled spectral power distribution.
print('\n"Zeros Filled Spectral Power Distribution"')
zeros_spd = colour.zeros_spd()
print(zeros_spd.shape)
print(zeros_spd[400])

# Defining a ones filled spectral power distribution.
print('\n"Ones Filled Spectral Power Distribution"')
ones_spd = colour.ones_spd()
print(ones_spd.shape)
print(ones_spd[400])
```

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

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

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

By default the shape used by `colour.constant_spd`, `colour.zeros_spd` and `colour.ones_spd` is the one defined by `colour.DEFAULT_SPECTRAL_SHAPE` attribute using the *CIE 1931 2° Standard Observer* shape.

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

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

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

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

```
1.0
```

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

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

```
spd1 = colour.ones_spd()
print('"Ones Filled Spectral Power Distribution"')
print(spd1[400])

print('\n"x2 Constant Multiplied"')
print((spd1 * 2)[400])

print('\n"+ Spectral Power Distribution"')
print((spd1 + colour.ones_spd())[400])
```

```
"Ones Filled Spectral Power Distribution"
1.0

"x2 Constant Multiplied"
2.0

"+ Spectral Power Distribution"
2.0
```

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

We can check the uniformity of the sample spectral power distribution:

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

```
True
```

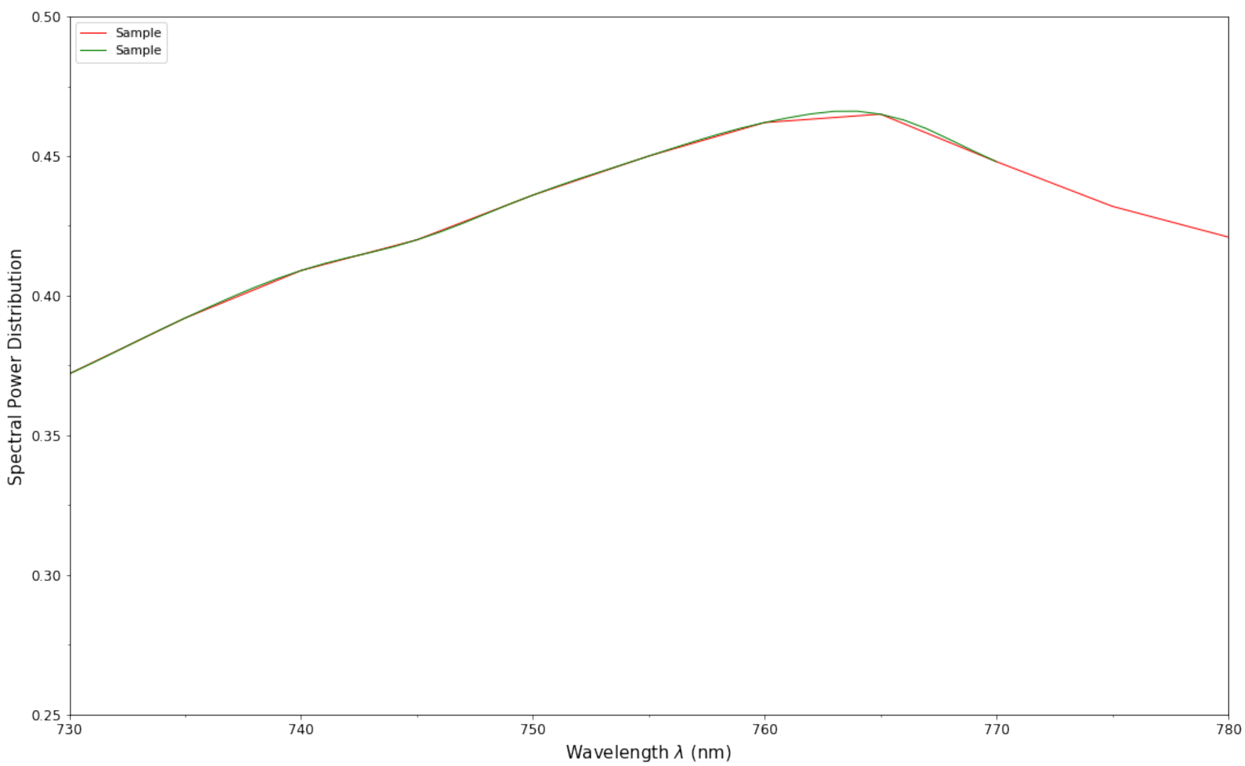
Since the sample spectral power distribution is uniform the interpolation will default to the `colour.SpragueInterpolator` interpolator.

Note: Interpolation happens in place and may alter your original data, use the colour. SpectralPowerDistribution.copy method to produce a copy of your spectral power distribution before interpolation.

```
# *Colour* can emit a substantial amount of warnings, we filter them.
colour.filter_warnings()

# Copying the sample spectral power distribution.
spd_copy = spd.copy()

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

```
0.065809599999999996
```

```
# Comparing the interpolated spectral power distribution with the original one.
multi_spd_plot([spd, spd_copy], bounding_box=[730,780, 0.25, 0.5])
```



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

```
# Extrapolating the copied sample spectral power distribution.
spd_copy.extrapolate(colour.SpectralShape(340, 830))
spd_copy[340], spd_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 power distribution.
spd_copy.interpolate(
    colour.SpectralShape(400, 700, 10), interpolator=colour.LinearInterpolator)
```

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

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

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

```
(0.046999999999999348, 0.0)
```

Aligning a spectral power 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 power distribution.
# We first trim the spectral power distribution as above.
spd_copy.interpolate(colour.SpectralShape(400, 700))
spd_copy.align(colour.SpectralShape(340, 830, 5))
spd_copy[340], spd_copy[830]
```

```
(0.065000000000000002, 0.28199999999999975)
```

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

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

print((spd.copy() + 1).values)
print((spd.copy() * 2).values)
print((spd * [0.35, 1.55, 0.75, 2.55, 0.95, 0.65, 0.15]).values)
print((spd * colour.constant_spd(2, spd.shape) * colour.constant_spd(3, spd.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 power distribution can be normalised with an arbitrary factor:

```
print(spd.normalise().values)
print(spd.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.SpectralPowerDistribution` 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 we can illustrate its core capability.

```
import numpy as np

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

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

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

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

3.1.1.3 Convert to Tristimulus Values

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

```
spd = colour.SpectralPowerDistribution(sample_spd_data)
cmfs = colour.STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
illuminant = colour.ILLUMINANTS_RELATIVE_SPDS['D65']

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

```
[ 10.97085572   9.70278591   6.05562778]
```

3.1.1.4 From *CIE XYZ* Colourspace

CIE XYZ is the central colourspace for Colour Science from which many computations are available, cascading 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_Hunt',
 'XYZ_to_ATD95',
 'XYZ_to_CIECAM02',
 'XYZ_to_LLAB',
 'XYZ_to_Nayatani95',
 'XYZ_to_RLAB',
 'XYZ_to_xyY',
 'XYZ_to_xy',
 'XYZ_to_Lab',
 'XYZ_to_Luv',
 'XYZ_to_UCS',
 'XYZ_to_UVW',
 'XYZ_to_hdr_CIELab',
 'XYZ_to_K_ab_HunterLab1966',
 'XYZ_to_Hunter_Lab',
 'XYZ_to_Hunter_Rdab',
 'XYZ_to_Hunter_Rdab',
 'XYZ_to_IPT',
 'XYZ_to_hdr_IPT',
 'XYZ_to_colourspace_model',
 'XYZ_to_RGB',
 'XYZ_to_sRGB',
 'XYZ_to_spectral_Meng2015',
 'XYZ_to_spectral']
```

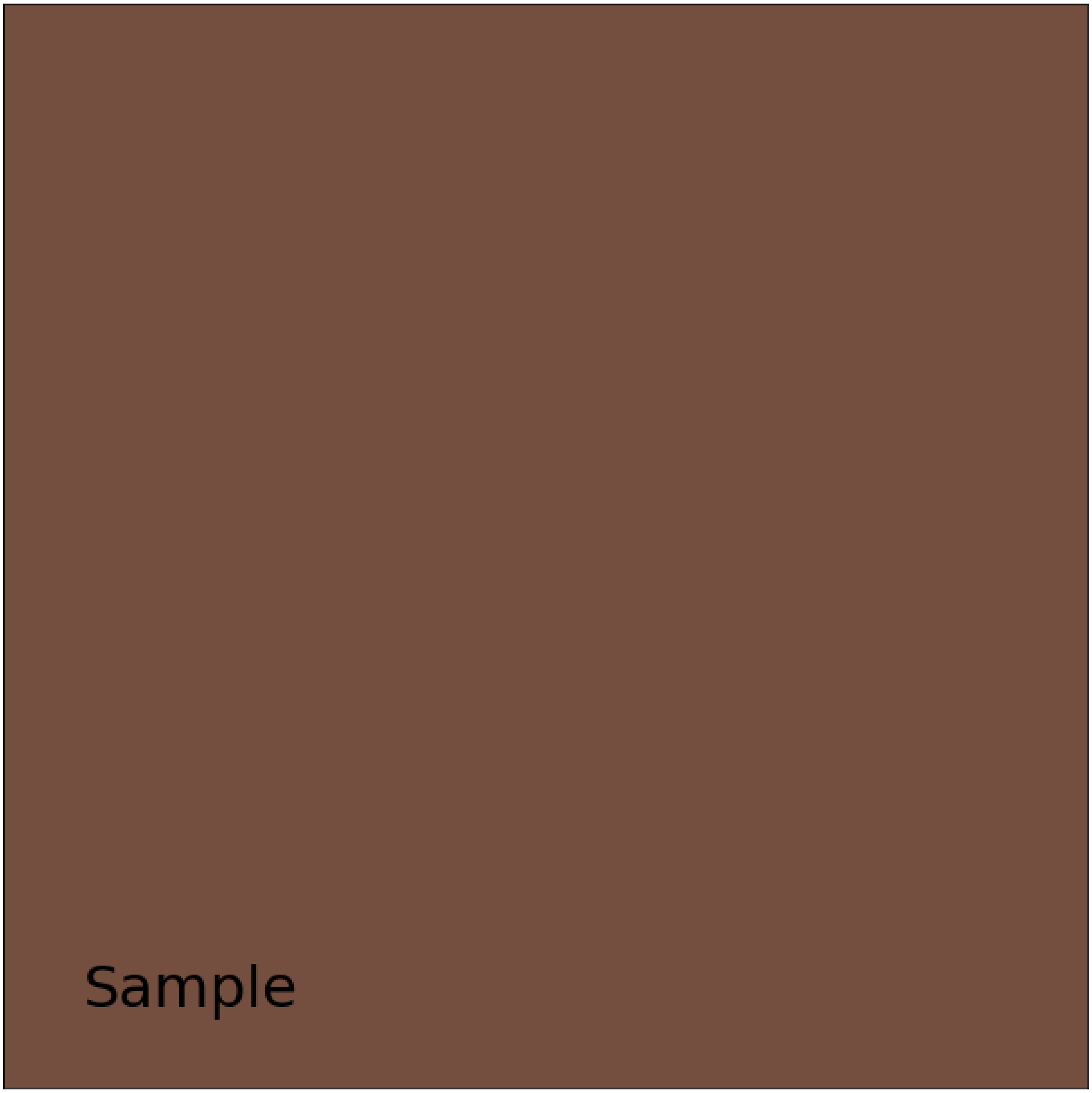
3.1.1.5 Convert to Screen Colours

We can for instance converts the *CIE XYZ* tristimulus values into *sRGB* colourspace *RGB* values in order to display them on screen:

```
# The output domain of *colour.spectral_to_XYZ* is [0, 100] and the input
# domain of *colour.XYZ_to_sRGB* is [0, 1]. We need to take it in account and
# rescale the input *CIE XYZ* colourspace matrix.
RGB = colour.XYZ_to_sRGB(XYZ / 100)
print(RGB)
```

```
[ 0.45675795  0.30986982  0.24861924]
```

```
# Plotting the *sRGB* colourspace colour of the *Sample* spectral power distribution.
single_colour_swatch_plot(ColourSwatch('Sample', RGB), text_size=32)
```



3.1.1.6 Generate Colour Rendition Charts

In the same way, we can compute values from a colour rendition chart sample.

Note: This is useful for render time checks in the VFX industry, where you can use a synthetic colour chart into your render and ensure the colour management is acting as expected.

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

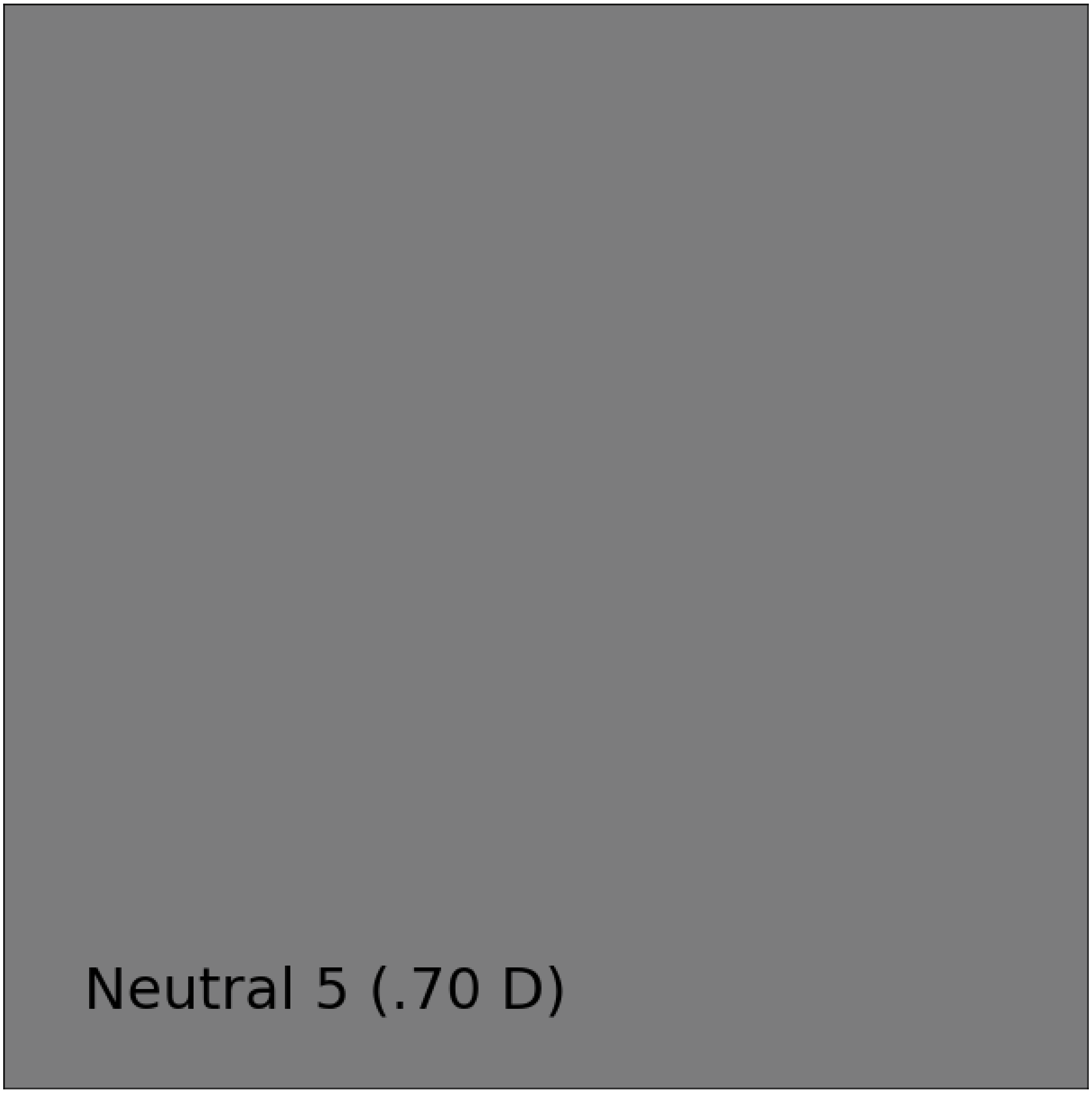
```
# Colour rendition charts chromaticity coordinates.
print(sorted(colour.characterisation.COLOURCHECKERS.keys()))
```

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

```
[u'BabelColor Average', u'ColorChecker 1976', u'ColorChecker 2005', u'babel_average', u'cc2005']
[u'BabelColor Average', u'ColorChecker N Ohta', u'babel_average', u'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_spd = colour.COLOURCHECKERS_SPDS['ColorChecker N Ohta'][patch_name]
XYZ = colour.spectral_to_XYZ(patch_spd, cmfs, illuminant)
RGB = colour.XYZ_to_sRGB(XYZ / 100)

single_colour_swatch_plot(ColourSwatch(patch_name.title(), RGB), text_size=32)
```



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

```
colour_checker_plot(colour_checker='ColorChecker 2005', text_display=False)
```

3.1.1.7 Convert to Chromaticity Coordinates

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

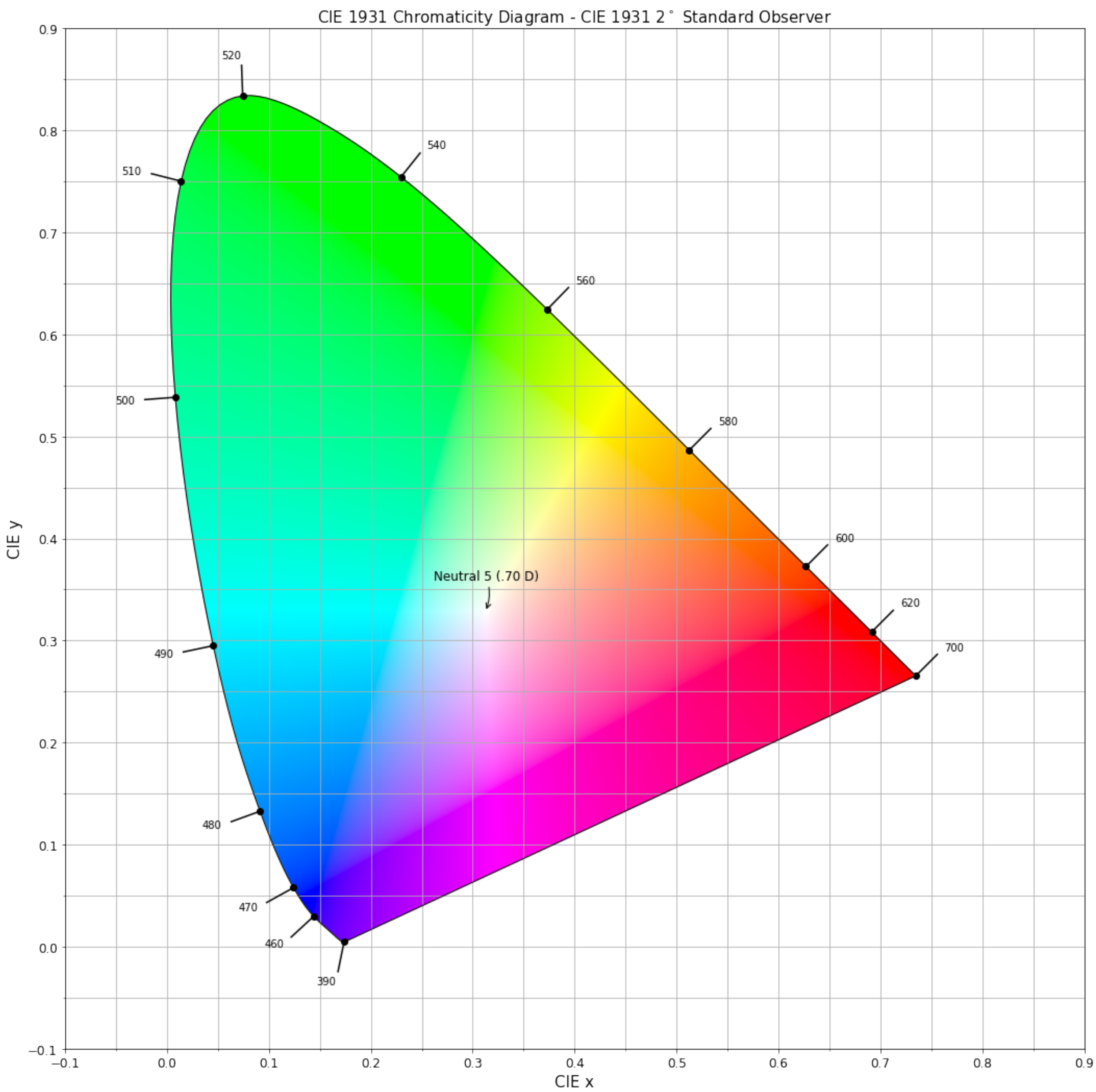
```
# Computing *xy* chromaticity coordinates for the *neutral 5 (.70 D)* patch.
xy =  colour.XYZ_to_xy(XYZ)
print(xy)
```

```
[ 0.31259787  0.32870029]
```

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

```
import pylab

# 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.
chromaticity_diagram_plot_CIE1931(standalone=False)

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

```
# Annotating the plot.
pylab.annotate(patch_spd.name.title(),
               xy=xy,
               xytext=(-50, 30),
               textcoords='offset points',
               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)
```



3.1.1.8 And More…

We hope that this small introduction has been useful and gave you the envy to see more, if you want to explore the API a good place to start is the Jupyter Notebooks page.

3.1.2 Reference

3.1.2.1 Colour

Chromatic Adaptation

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.
CMCCAT2000_VIEWING_CONDITIONS	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.

Other Parameters

- **E_o1** (*numeric*) – {colour.adaptation.chromatic_adaptation_CIE1994()}, Test illuminance E_{o1} in cd/m^2.

- **E_o2** (*numeric*) – {colour.adaptation.chromatic_adaptation_CIE1994()}, Reference illuminance E_{o2} in cd/m^2.
- **L_A1** (*numeric or array_like*) – {colour.adaptation.chromatic_adaptation_CMCCAT2000()}, Luminance of test adapting field L_{A1} in cd/m^2.
- **L_A2** (*numeric or array_like*) – {colour.adaptation.chromatic_adaptation_CMCCAT2000()}, Luminance of reference adapting field L_{A2} in cd/m^2.
- **Y_n** (*numeric or array_like*) – {colour.adaptation.chromatic_adaptation_Fairchild1990()}, Luminance Y_n of test adapting stimulus in cd/m^2.
- **Y_o** (*numeric*) – {colour.adaptation.chromatic_adaptation_CIE1994()}, Luminance factor Y_o of achromatic background as percentage in domain [18, 100].
- **direction** (*unicode, optional*) – {colour.adaptation.chromatic_adaptation_CMCCAT2000()}, **{'Forward', 'Reverse'}**, Chromatic adaptation direction.
- **discount_illuminant** (*bool, optional*) – {colour.adaptation.chromatic_adaptation_Fairchild1990()}, Truth value indicating if the illuminant should be discounted.
- **n** (*numeric, optional*) – {colour.adaptation.chromatic_adaptation_CIE1994()}, Noise component in fundamental primary system.
- **surround** (*CMCCAT2000_InductionFactors, optional*) – {colour.adaptation.chromatic_adaptation_CMCCAT2000()}, Surround viewing conditions induction factors.
- **transform** (*unicode, optional*) – {colour.adaptation.chromatic_adaptation_VonKries()}, **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, Chromatic adaptation transform.

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

Return type ndarray

References

- *[CIET13294]*
- *[Fai91]*
- *[Fai13c]*
- *[Fai13b]*
- *[LLRH02]*
- *[WRC12a]*

Examples

Von Kries chromatic adaptation:

```
>>> import numpy as np
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_w = np.array([1.09846607, 1.00000000, 0.35582280])
>>> XYZ_wr = np.array([0.95042855, 1.00000000, 1.08890037])
>>> chromatic_adaptation(XYZ, XYZ_w, XYZ_wr)
...
array([ 0.0839746...,  0.1141321...,  0.2862554...])
```

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 = 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({'Von Kries': ..., 'CMCCAT2000': ..., 'CIE 199**

Supported chromatic adaptation methods.

References

- *[CIET13294]*
- *[Fai91]*
- *[Fai13c]*
- *[Fai13b]*
- *[LLRH02]*

- *[WRC12a]*

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

colour.CMCCAT2000_VIEWING_CONDITIONS

`colour.CMCCAT2000_VIEWING_CONDITIONS = CaseInsensitiveMapping({u'Dark': ..., u'Dim': ..., u'Average': ...})`

Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

- *[LLRH02]*
- *[WRC12a]*

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

Dataset

`colour`

`CHROMATIC_ADAPTATION_TRANSFORMS`	Supported chromatic adaptation transforms.

colour.CHROMATIC_ADAPTATION_TRANSFORMS

`colour.CHROMATIC_ADAPTATION_TRANSFORMS = CaseInsensitiveMapping({u'Bradford': ..., u'Fairchild': ..., u'CAT`

Supported chromatic adaptation transforms.

References

- *[BS10]*
- *[BS08]*
- *[Fai]*
- *[LPLMartinezverdu07]*
- *[Lin09a]*
- *[WRC12b]*
- *[WRC12a]*
- *[Wikb]*

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

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 in domain [0, 100].
- **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

Warning: The input domain and output range of that definition are non standard!

Notes

- Input *CIE XYZ_1*, *CIE XYZ_n* and *CIE XYZ_r* tristimulus values are in domain [0, 100].
- Output *CIE XYZ_2* tristimulus values are in range [0, 100].

References

- *[Fai91]*
- *[Fai13c]*

Examples

```
>>> XYZ_1 = np.array([19.53, 23.07, 24.97])
>>> XYZ_n = np.array([111.15, 100.00, 35.20])
>>> XYZ_r = np.array([94.81, 100.00, 107.30])
>>> Y_n = 200
>>> chromatic_adaptation_Fairchild1990(XYZ_1, XYZ_n, XYZ_r, Y_n)
...
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 in domain [0, 100].
- **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 in domain [18, 100].
- **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

> **Warning:** The input domain and output range of that definition are non standard!

Notes

- Input *CIE XYZ_1* tristimulus values are in domain [0, 100].

- Output *CIE XYZ_2* tristimulus values are in range [0, 100].

References

- *[CIET13294]*

Examples

```
>>> XYZ_1 = np.array([28.00, 21.26, 5.27])
>>> xy_o1 = np.array([0.4476, 0.4074])
>>> xy_o2 = np.array([0.3127, 0.3290])
>>> Y_o = 20
>>> E_o1 = 1000
>>> E_o2 = 1000
>>> chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
...
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.
`CMCCAT2000_VIEWING_CONDITIONS`	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=CMCCAT2000_InductionFactors(F=1), direction=u'Forward'*)

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

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

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (array_like) – Source viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **XYZ_wr** (array_like) – Target viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **L_A1** (numeric or array_like) – Luminance of test adapting field L_{A1} in cd/m^2.
- **L_A2** (numeric or array_like) – Luminance of reference adapting field L_{A2} in cd/m^2.

- **surround** (CMCCAT2000_InductionFactors, optional) – Surround viewing conditions induction factors.
- **direction** (unicode, optional) – **{'Forward', 'Reverse'}**, Chromatic adaptation direction.

Returns Adapted stimulus *CIE XYZ* tristimulus values.

Return type ndarray

Warning: The input domain and output range of that definition are non standard!

Notes

- Input *CIE XYZ*, *CIE XYZ_w* and *CIE XYZ_wr* tristimulus values are in domain [0, 100].
- Output *CIE XYZ* tristimulus values are in range [0, 100].

References

- *[LLRH02]*
- *[WRC12a]*

Examples

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

Using the *CMCCAT2000* reverse model:

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

colour.adaptation.CMCCAT2000_VIEWING_CONDITIONS

colour.adaptation.**CMCCAT2000_VIEWING_CONDITIONS = CaseInsensitiveMapping({u'Dark': ..., u'Dim': ..., u'Aver**

Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

- [LLRH02]
- [WRC12a]

CMCCAT2000_VIEWING_CONDITIONS [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_reverse_CMCCAT2000`(...)	Adapts given stimulus corresponding colour *CIE XYZ* tristimulus values from reference viewing conditions to test viewing conditions using *CMCCAT2000* reverse chromatic adaptation model.
`CMCCAT2000_InductionFactors`	*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=CMCCAT2000_InductionFactors(F=1)*)

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

Parameters

- **XYZ** (`array_like`) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (`array_like`) – Test viewing condition *CIE XYZ* tristimulus values of the 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** (`CMCCAT2000_InductionFactors, optional`) – Surround viewing conditions induction factors.

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

Return type ndarray

> **Warning:** The input domain and output range of that definition are non standard!

Notes

- Input *CIE XYZ*, *CIE XYZ_w* and *CIE XYZ_wr* tristimulus values are in domain [0, 100].
- Output *CIE XYZ_c* tristimulus values are in range [0, 100].

References

- *[LLRH02]*
- *[WRC12a]*

Examples

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

colour.adaptation.chromatic_adaptation_reverse_CMCCAT2000

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

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

Parameters

- **XYZ_c** (array_like) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (array_like) – Test viewing condition *CIE XYZ* tristimulus values of the 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** (CMCCAT2000_InductionFactors, optional) – Surround viewing conditions induction factors.

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

Return type ndarray

Warning: The input domain and output range of that definition are non standard!

Notes

- Input *CIE XYZ_c*, *CIE XYZ_w* and *CIE XYZ_wr* tristimulus values are in domain [0, 100].
- Output *CIE XYZ* tristimulus values are in range [0, 100].

References

- *[LLRH02]*
- *[WRC12a]*

Examples

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

colour.adaptation.CMCCAT2000_InductionFactors

class colour.adaptation.**CMCCAT2000_InductionFactors**

CMCCAT2000 chromatic adaptation model induction factors.

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

References

- *[LLRH02]*
- *[WRC12a]*

Create new instance of CMCCAT2000_InductionFactors(F,)

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

Von Kries

colour.adaptation

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

colour.adaptation.chromatic_adaptation_VonKries

colour.adaptation.**chromatic_adaptation_VonKries**(*XYZ, XYZ_w, XYZ_wr, transform=u'CAT02'*)
Adapts given stimulus from test viewing conditions to reference viewing conditions.

Parameters

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

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

Return type ndarray

References

- *[Fai13b]*

Examples

```
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_w = np.array([1.09846607, 1.00000000, 0.35582280])
>>> XYZ_wr = np.array([0.95042855, 1.00000000, 1.08890037])
>>> chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr)
array([ 0.0839746...,  0.1141321...,  0.2862554...])
```

Using Bradford method:

```
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_w = np.array([1.09846607, 1.00000000, 0.35582280])
>>> XYZ_wr = np.array([0.95042855, 1.00000000, 1.08890037])
>>> transform = 'Bradford'
>>> chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr, transform)
...
array([ 0.0854032...,  0.1140122...,  0.2972149...])
```

colour.adaptation.CHROMATIC_ADAPTATION_TRANSFORMS

colour.adaptation.**CHROMATIC_ADAPTATION_TRANSFORMS = CaseInsensitiveMapping({u'Bradford': ..., u'Fairchild':**
Supported chromatic adaptation transforms.

References

- *[BS10]*
- *[BS08]*
- *[Fai]*
- *[LPLMartinezverdu07]*
- *[Lin09a]*
- *[WRC12b]*
- *[WRC12a]*
- *[Wikb]*

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

Dataset

colour.adaptation

BRADFORD_CAT
BS_CAT
BS_PC_CAT
CAT02_BRILL_CAT
CAT02_CAT
CMCCAT2000_CAT
CMCCAT97_CAT
FAIRCHILD_CAT
SHARP_CAT
VON_KRIES_CAT
XYZ_SCALING_CAT

colour.adaptation.BRADFORD_CAT

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

colour.adaptation.BS_CAT

colour.adaptation.**BS_CAT = array([[0.8752, 0.2787, -0.1539], [-0.8904, 1.8709, 0.0195], [-0.0061, 0.0162, 0.**

colour.adaptation.BS_PC_CAT

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

colour.adaptation.CAT02_BRILL_CAT

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

colour.adaptation.CAT02_CAT

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

colour.adaptation.CMCCAT2000_CAT

colour.adaptation.**CMCCAT2000_CAT = array([[7.98200000e-01, 3.38900000e-01, -1.37100000e-01], [-5.91800000e-**

colour.adaptation.CMCCAT97_CAT

colour.adaptation.**CMCCAT97_CAT = array([[0.8951, -0.7502, 0.0389], [0.2664, 1.7135, 0.0685], [-0.1614, 0.03**

colour.adaptation.FAIRCHILD_CAT

colour.adaptation.**FAIRCHILD_CAT = array([[0.8562, 0.3372, -0.1934], [-0.836 , 1.8327, 0.0033], [0.0357, -0.**

colour.adaptation.SHARP_CAT

colour.adaptation.**SHARP_CAT = array([[1.2694, -0.0988, -0.1706], [-0.8364, 1.8006, 0.0357], [0.0297, -0.03**

colour.adaptation.VON_KRIES_CAT

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

colour.adaptation.XYZ_SCALING_CAT

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

Ancillary Objects

colour.adaptation

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

colour.adaptation.chromatic_adaptation_matrix_VonKries

colour.adaptation.**chromatic_adaptation_matrix_VonKries**(*XYZ_w, XYZ_wr, transform=u'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_CAT', 'Bianco', 'Bianco PC'}**, Chromatic adaptation transform.

Returns Chromatic adaptation matrix.

Return type ndarray

Raises KeyError – If chromatic adaptation method is not defined.

References

- *[Fai13b]*

Examples

```
>>> XYZ_w = np.array([1.09846607, 1.00000000, 0.35582280])
>>> XYZ_wr = np.array([0.95042855, 1.00000000, 1.08890037])
>>> chromatic_adaptation_matrix_VonKries(XYZ_w, XYZ_wr)
...
array([[ 0.8687653...,  -0.1416539...,   0.3871961...],
       [-0.1030072...,   1.0584014...,   0.1538646...],
       [ 0.0078167...,   0.0267875...,   2.9608177...]])
```

Using Bradford method:

```
>>> XYZ_w = np.array([1.09846607, 1.00000000, 0.35582280])
>>> XYZ_wr = np.array([0.95042855, 1.00000000, 1.08890037])
>>> method = 'Bradford'
>>> chromatic_adaptation_matrix_VonKries(XYZ_w, XYZ_wr, method)
...
array([[ 0.8446794...,  -0.1179355...,   0.3948940...],
       [-0.1366408...,   1.1041236...,   0.1291981...],
       [ 0.0798671...,  -0.1349315...,   3.1928829...]])
```

Algebra

Extrapolation

colour

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

colour.Extrapolator

class colour.**Extrapolator**(*interpolator=None*, *method=u'Linear'*, *left=None*, *right=None*, *dtype=<type 'numpy.float64'>*)

Extrapolates the 1-D function of given interpolator.

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

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

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

Parameters

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

__class__()

Notes

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

References

- *[Sas]*
- *[WRC12d]*

Examples

Extrapolating a single numeric variable:

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

Extrapolating an *array_like* variable:

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

Using the *Constant* extrapolation method:

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

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

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

__init__(*interpolator=None, method=u'Linear', left=None, right=None, dtype=<type 'numpy.float64'>*)

Methods

`__init__`([interpolator, method, left, …])

Interpolation

`colour`

`KernelInterpolator`(x, y[, window, kernel, …])	Kernel based interpolation of a 1-D function.
`LinearInterpolator`(x, y[, dtype])	Linearly interpolates a 1-D function.
`NullInterpolator`(x, y[, absolute_tolerance, …])	Performs 1-D function null interpolation, i.e.
`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.
`lagrange_coefficients`(r[, n])	Computes the *Lagrange Coefficients* at given point r for degree n.

colour.KernelInterpolator

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

Kernel based interpolation of a 1-D function.

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

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

Parameters

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

x

y

window

kernel

kernel_args

padding_args

__call__()

References

- *[BB09]*
- *[Wikm]*

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_args={'a': 16})
>>> f([0.25, 0.75])
array([ 5.396179...,  5.652109...])
```

__init__(*x*, *y*, *window=3*, *kernel=<function kernel_lanczos>*, *kernel_args=None*, *padding_args=None*, *dtype=<type 'numpy.float64'>*)

Methods

`__init__`(x, y[, window, kernel, . . .])	

colour.LinearInterpolator

class colour.**LinearInterpolator**(*x*, *y*, *dtype=<type 'numpy.float64'>*)

Linearly interpolates a 1-D function.

Parameters

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

x

y

__call__()

Notes

This class is a wrapper around *numpy.interp* definition.

Examples

Interpolating a single numeric variable:

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

```
>>> f(0.5)
7.64...
```

Interpolating an *array_like* variable:

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

__init__(*x, y, dtype=<type 'numpy.float64'>*)

Methods

`__init__`(x, y[, dtype])	

colour.NullInterpolator

class colour.**NullInterpolator**(*x, y, absolute_tolerance=1e-06, relative_tolerance=1e-06, default=nan, dtype=<type 'numpy.float64'>*)

Performs 1-D function null interpolation, i.e. a call within given tolerances will return existing y variable values and `default` if outside tolerances.

Parameters

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

x

y

relative_tolerance

absolute_tolerance

default

__call__()

Examples

```
>>> y = np.array([5.9200, 9.3700, 10.8135, 4.5100,
...               69.5900, 27.8007, 86.0500])
>>> x = np.arange(len(y))
>>> f = NullInterpolator(x, y)
>>> f(0.5)
nan
>>> f(1.0)
9.3699999...
```

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

__init__(*x, y, absolute_tolerance=1e-06, relative_tolerance=1e-06, default=nan, dtype=<type 'numpy.float64'>*)

Methods

`__init__`(x, y[, absolute_tolerance, ...])	

colour.PchipInterpolator

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

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

y

Notes

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

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

Methods

`__init__`(x, y, *args, **kwargs)	
`antiderivative`([nu])	Construct a new piecewise polynomial representing the antiderivative.
`construct_fast`(c, x[, extrapolate, axis])	Construct the piecewise polynomial without making checks.
`derivative`([nu])	Construct a new piecewise polynomial representing the derivative.
`extend`(c, x[, right])	Add additional breakpoints and coefficients to the polynomial.
`from_derivatives`(xi, yi[, orders, extrapolate])	Construct a piecewise polynomial in the Bernstein basis, compatible with the specified values and derivatives at breakpoints.
`from_power_basis`(pp[, extrapolate])	Construct a piecewise polynomial in Bernstein basis from a power basis polynomial.
`integrate`(a, b[, extrapolate])	Compute a definite integral over a piecewise polynomial.
`roots`()	Return the roots of the interpolated function.

colour.SpragueInterpolator

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

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

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

Parameters

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

x

y

__call__()

Notes

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

References

- *[CIET13805b]*
- *[WRC12e]*

Examples

Interpolating a single numeric variable:

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

Interpolating an *array_like* variable:

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

__init__(*x*, *y*, *dtype=<type 'numpy.float64'>*)

Methods

__init__(x, y[, dtype])	

colour.lagrange_coefficients

colour.**lagrange_coefficients**(*r*, *n=4*)

Computes the *Lagrange Coefficients* at given point r for degree n.

Parameters

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

Returns

Return type ndarray

References

- *[Fai85]*
- *[WikI]*

Examples

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

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

- *[BB09]*

Examples

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

colour.kernel_linear

colour.**kernel_linear**(*x*)

Returns the *linear* kernel evaluated at given samples.

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

Returns The *linear* kernel evaluated at given samples.

Return type ndarray

References

- *[BB09]*

Examples

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

colour.kernel_sinc

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

Returns the *sinc* kernel evaluated at given samples.

Parameters

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

Returns The *sinc* kernel evaluated at given samples.

Return type ndarray

References

- *[BB09]*

Examples

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

- *[Wikm]*

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

- *[BB09]*

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.        ])
```

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

colour.algebra.cartesian_to_spherical

colour.algebra.**cartesian_to_spherical**(*a*)

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

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

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

Return type ndarray

References

- *[Wiko]*
- *[Wikk]*

Examples

```
>>> a = np.array([3, 1, 6])
>>> cartesian_to_spherical(a)
array([ 6.7823299...,  1.0857465...,  0.3217505...])
```

colour.algebra.spherical_to_cartesian

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

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

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

Returns Cartesian coordinates array xyz.

Return type ndarray

References

- *[Wiko]*
- *[Wikk]*

Examples

```
>>> a = np.array([6.78232998, 1.08574654, 0.32175055])
>>> spherical_to_cartesian(a)
array([ 3.        ,  0.9999999...,  6.        ])
```

colour.algebra.cartesian_to_polar

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

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

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

Returns Polar coordinates array $\rho\phi$.

Return type ndarray

References

- *[Wiko]*
- *[Wikk]*

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.

Returns Cartesian coordinates array xy.

Return type ndarray

References

- *[Wiko]*
- *[Wikk]*

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 $\rho\phi z$ (azimuth, radial distance and height).

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

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

Return type ndarray

References

- *[Wiko]*
- *[Wikk]*

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

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

Returns Cartesian coordinates array xyz.

Return type ndarray

References

- *[Wiko]*
- *[Wikk]*

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.

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([0.07049534, 0.10080000, 0.09558313])
>>> normalise_vector(a)
array([ 0.4525410...,  0.6470802...,  0.6135908...])
```

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

- *[Sae]*

Notes

- Input line segment points coordinates are 2d coordinates.

Examples

```
>>> a = np.array([0.95694934, 0.13720932])
>>> b = np.array([0.28382835, 0.60608318])
>>> extend_line_segment(a, b)
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

- *[Bou]*
- *[Erda]*

Notes

- Input line segments points coordinates are 2d coordinates.

Examples

```
>>> l_1 = np.array(
...     [[[0.15416284, 0.7400497],
...       [0.26331502, 0.53373939]],
...      [[0.01457496, 0.91874701],
...       [0.90071485, 0.03342143]]]
... )
>>> l_2 = np.array(
...     [[[0.95694934, 0.13720932],
...        [0.28382835, 0.60608318]],
```

```
...       [[0.94422514, 0.85273554],
...        [0.00225923, 0.52122603]],
...       [[0.55203763, 0.48537741],
...        [0.76813415, 0.16071675]]]
... )
>>> s = intersect_line_segments(l_1, l_2)
>>> s.xy
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)
```

Ancillary Objects

`colour.algebra`

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

colour.algebra.LineSegmentsIntersections_Specification

class `colour.algebra.`**`LineSegmentsIntersections_Specification`**

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

Parameters

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

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

`__init__`()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

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=<mtrand.RandomState object>*)

Returns a generator yielding random triplets.

Parameters

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

Colour Appearance Models

ATD (1995)

`colour`

XYZ_to_ATD95(XYZ, XYZ_0, Y_0, k_1, k_2[, sigma])	Computes the *ATD (1995)* colour vision model correlates.
ATD95_Specification	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 in domain [0, 100].
- **XYZ_0** (array_like) – *CIE XYZ* tristimulus values of reference white in domain [0, 100].
- **Y_0** (numeric or array_like) – Absolute adapting field luminance in cd/m^2.
- **k_1** (numeric or array_like) – Application specific weight k_1.
- **k_2** (numeric or array_like) – Application specific weight k_2.
- **sigma** (numeric or array_like, optional) – Constant σ varied to predict different types of data.

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

Return type *ATD95_Specification*

Warning: The input domain of that definition is non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].
- Input *CIE XYZ_0* tristimulus values are in domain [0, 100].
- For unrelated colors, there is only self-adaptation and k_1 is set to 1.0 while k_2 is set to 0.0. For related colors such as typical colorimetric applications, k_1 is set to 0.0 and k_2 is set to a value between 15 and 50 *(Guth, 1995)*.

References

- *[Fai13a]*
- *[Gut95]*

Examples

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

colour.ATD95_Specification

class colour.**ATD95_Specification**

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

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

Parameters

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

Notes

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

References

- *[Fai13a]*
- *[Gut95]*

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

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

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(CIECAM02_specification, ...)	Converts *CIECAM02* specification to *CIE XYZ* tristimulus values.
CIECAM02_Specification	Defines the *CIECAM02* colour appearance model specification.
CIECAM02_VIEWING_CONDITIONS	Reference *CIECAM02* colour appearance model viewing conditions.

colour.XYZ_to_CIECAM02

colour.**XYZ_to_CIECAM02**(*XYZ, XYZ_w, L_A, Y_b, surround=CIECAM02_InductionFactors(F=1, c=0.69, N_c=1), discount_illuminant=False*)

Computes the *CIECAM02* colour appearance model correlates from given *CIE XYZ* tristimulus values.

This is the *forward* implementation.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus in domain [0, 100].
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white in domain [0, 100].
- **L_A** (numeric or array_like) – Adapting field *luminance* L_A in cd/m^2, (often taken to be 20% of the luminance of a white object in the scene).
- **Y_b** (numeric or array_like) – Relative luminance of background Y_b in cd/m^2.
- **surround** (CIECAM02_InductionFactors, optional) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool, optional) – Truth value indicating if the illuminant should be discounted.

Returns *CIECAM02* colour appearance model specification.

Return type *CIECAM02_Specification*

Warning: The input domain of that definition is non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].
- Input *CIE XYZ_w* tristimulus values are in domain [0, 100].

References

- *[Fai04]*
- *[LL13]*
- *[MFH+02]*
- *[Wikf]*

Examples

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

colour.CIECAM02_to_XYZ

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

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

This is the *reverse* implementation.

Parameters

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

Returns **XYZ** – *CIE XYZ* tristimulus values.

Return type ndarray

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

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

Notes

- CIECAM02_specification can also be passed as a compatible argument colour.utilities.as_namedtuple() definition.
- Input *CIE XYZ_w* tristimulus values are in domain [0, 100].
- Output *CIE XYZ* tristimulus values are in range [0, 100].

References

- *[Fai04]*
- *[LL13]*
- *[MFH+02]*
- *[Wikf]*

Examples

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

colour.CIECAM02_Specification

class colour.**CIECAM02_Specification**

Defines the *CIECAM02* colour appearance model specification.

Parameters

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

- **HC** (numeric or array_like) – *Hue* h composition H^C.

References

- *[Fai04]*
- *[LL13]*
- *[MFH+02]*
- *[Wikf]*

Returns a new instance of the colour.CIECAM02_Specification class.

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

colour.CIECAM02_VIEWING_CONDITIONS

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

References

- *[Fai04]*
- *[LL13]*
- *[MFH+02]*
- *[Wikf]*

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

Ancillary Objects

colour.appearance

CIECAM02_InductionFactors	*CIECAM02* colour appearance model induction factors.

colour.appearance.CIECAM02_InductionFactors

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

Parameters

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

References

- [Fai04]
- [LL13]
- [MFH+02]
- [Wikf]

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

__init__()
x.__init__(. . .) initializes x; see help(type(x)) for signature

Methods

count(. . .)	
index((value, [start, . . .)	Raises ValueError if the value is not present.

CAM16

colour

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

colour.XYZ_to_CAM16

colour.**XYZ_to_CAM16**(*XYZ, XYZ_w, L_A, Y_b, surround=CAM16_InductionFactors(F=1.0, c=0.69, N_c=1.0), 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 in domain [0, 100].
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white in domain [0, 100].

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

Returns *CAM16* colour appearance model specification.

Return type *CAM16_Specification*

Warning: The input domain of that definition is non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].
- Input *CIE XYZ_w* tristimulus values are in domain [0, 100].

References

- *[LLW+17]*

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CAM16_VIEWING_CONDITIONS['Average']
>>> XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b, surround)
CAM16_Specification(J=41.7180250..., C=11.9413446..., h=210.3838955..., s=25.3564036..., Q=193.
↪0617673..., M=12.4128523..., H=267.0983345..., HC=None)
```

colour.CAM16_to_XYZ

colour.**CAM16_to_XYZ**(*CAM16_specification, XYZ_w, L_A, Y_b, surround=CAM16_InductionFactors(F=1.0, c=0.69, N_c=1.0), discount_illuminant=False*)

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

This is the *reverse* implementation.

Parameters

- **CAM16_specification** (CAM16_Specification) – *CAM16* colour appearance model specification. Correlate of *Lightness* J, correlate of *chroma* C or correlate of *colourfulness* M and *hue* angle h in degrees must be specified, e.g. JCh or JMh.

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

Returns **XYZ** – *CIE XYZ* tristimulus values.

Return type ndarray

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

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

Notes

- CAM16_specification can also be passed as a compatible argument colour.utilities.as_namedtuple() definition.
- Input *CIE XYZ_w* tristimulus values are in domain [0, 100].
- Output *CIE XYZ* tristimulus values are in range [0, 100].

References

- *[LLW+17]*

Examples

```
>>> specification = CAM16_Specification(J=41.718025051415616,
...                                     C=11.941344635245843,
...                                     h=210.38389558131118)
>>> 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.CAM16_Specification

class colour.**CAM16_Specification**
Defines the *CAM16* colour appearance model specification.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* J.
- **C** (numeric or array_like) – Correlate of *chroma* C.

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

References

- *[LLW+17]*

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

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

colour.CAM16_VIEWING_CONDITIONS

`colour.CAM16_VIEWING_CONDITIONS = CaseInsensitiveMapping({u'Dark': ..., u'Dim': ..., u'Average': ...})`
Reference *CAM16* colour appearance model viewing conditions.

References

- *[LLW+17]*

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

Ancillary Objects

colour.appearance

CAM16_InductionFactors	*CAM16* colour appearance model induction factors.

colour.appearance.CAM16_InductionFactors

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

Parameters

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

- **N_c** (numeric or array_like) – Chromatic induction factor N_c.

References

- [LLW+17]

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

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

Hunt

colour

XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A[, ...])	Computes the *Hunt* colour appearance model correlates.
Hunt_Specification	Defines the *Hunt* colour appearance model specification.
HUNT_VIEWING_CONDITIONS	Reference *Hunt* colour appearance model viewing conditions.

colour.XYZ_to_Hunt

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

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus in domain [0, 100].
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white in domain [0, 100].
- **XYZ_b** (array_like) – *CIE XYZ* tristimulus values of background in domain [0, 100].
- **L_A** (numeric or array_like) – Adapting field *luminance* L_A in cd/m^2.
- **surround** (Hunt_InductionFactors, optional) – Surround viewing conditions induction factors.
- **L_AS** (numeric or array_like, optional) – Scotopic luminance L_{AS} of the illuminant, approximated if not specified.

- **CCT_w** (numeric or array_like, optional) – Correlated color temperature T_{cp}: of the illuminant, needed to approximate L_{AS}.
- **XYZ_p** (array_like, optional) – *CIE XYZ* tristimulus values of proximal field in domain [0, 100], assumed to be equal to background if not specified.
- **p** (numeric or array_like, optional) – Simultaneous contrast / assimilation factor p with value in domain [-1, 0] when simultaneous contrast occurs and domain [0, 1] when assimilation occurs.
- **S** (numeric or array_like, optional) – Scotopic response S to the stimulus, approximated using tristimulus values Y of the stimulus if not specified.
- **S_w** (numeric or array_like, optional) – Scotopic response S_w for the reference white, approximated using the tristimulus values Y_w of the reference white if not specified.
- **helson_judd_effect** (bool, optional) – Truth value indicating whether the *Helson-Judd* effect should be accounted for.
- **discount_illuminant** (bool, optional) – Truth value indicating if the illuminant should be discounted.

Returns *Hunt* colour appearance model specification.

Return type *Hunt_Specification*

Raises ValueError – If an illegal arguments combination is specified.

> **Warning:** The input domain of that definition is non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].
- Input *CIE XYZ_b* tristimulus values are in domain [0, 100].
- Input *CIE XYZ_w* tristimulus values are in domain [0, 100].
- Input *CIE XYZ_p* tristimulus values are in domain [0, 100].

References

- *[Fai13f]*
- *[Hun04]*

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> XYZ_b = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> surround = HUNT_VIEWING_CONDITIONS['Normal Scenes']
>>> CCT_w = 6504.0
>>> XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A, surround, CCT_w=CCT_w)
```

```
...
Hunt_Specification(J=30.0462678..., C=0.1210508..., h=269.2737594..., s=0.0199093..., Q=22.
↪2097654..., M=0.1238964..., H=None, HC=None)
```

colour.Hunt_Specification

class colour.**Hunt_Specification**

Defines the *Hunt* colour appearance model specification.

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

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* J.
- **C** (numeric or array_like) – Correlate of *chroma* C_94.
- **h** (numeric or array_like) – *Hue* angle h_S in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* s.
- **Q** (numeric or array_like) – Correlate of *brightness* Q.
- **M** (numeric or array_like) – Correlate of *colourfulness* M_94.
- **H** (numeric or array_like) – *Hue* h quadrature H.
- **HC** (numeric or array_like) – *Hue* h composition H_C.

Notes

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

References

- *[Fai13f]*
- *[Hun04]*

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

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

colour.HUNT_VIEWING_CONDITIONS

colour.**HUNT_VIEWING_CONDITIONS = CaseInsensitiveMapping({u'Large Transparencies On Light Boxes': ..., u'Tele**

Reference *Hunt* colour appearance model viewing conditions.

References

- *[Fai13f]*
- *[Hun04]*

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

Aliases:

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

LLAB(l:c)

`colour`

`XYZ_to_LLAB`(XYZ, XYZ_0, Y_b, L[, surround, ...])	Computes the *LLAB(l:c)* colour appearance model correlates.
`LLAB_Specification`	Defines the *LLAB(l:c)* colour appearance model specification.
`LLAB_VIEWING_CONDITIONS`	*Reference *LLAB(l − c)** colour appearance model viewing conditions.

colour.XYZ_to_LLAB

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

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

Parameters

- **`XYZ`** (`array_like`) – *CIE XYZ* tristimulus values of test sample / stimulus in domain [0, 100].
- **`XYZ_0`** (`array_like`) – *CIE XYZ* tristimulus values of reference white in domain [0, 100].
- **`Y_b`** (`numeric or array_like`) – Luminance factor of the background in cd/m^2.
- **`L`** (`numeric or array_like`) – Absolute luminance L of reference white in cd/m^2.
- **`surround`** (`LLAB_InductionFactors, optional`) – Surround viewing conditions induction factors.

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

Return type *LLAB_Specification*

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

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].
- Input *CIE XYZ_0* tristimulus values are in domain [0, 100].

References

- *[Fai13e]*
- *[LLK96]*
- *[LM96]*

Examples

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

colour.LLAB_Specification

class colour.**LLAB_Specification**

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

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

Parameters

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

Notes

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

References

- *[Fai13e]*
- *[LLK96]*
- *[LM96]*

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

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

colour.LLAB_VIEWING_CONDITIONS

colour.**LLAB_VIEWING_CONDITIONS = CaseInsensitiveMapping({u'35mm Projection Transparency, Dark Surround': ...**
*Reference *LLAB(l* – c)* colour appearance model viewing conditions.

References

- *[Fai13e]*
- *[LLK96]*
- *[LM96]*

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

Aliases:

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

Ancillary Objects

`colour.appearance`

LLAB_InductionFactors	*LLAB(l:c)* colour appearance model induction factors.

colour.appearance.LLAB_InductionFactors

class colour.appearance.**LLAB_InductionFactors**

LLAB(l:c) colour appearance model induction factors.

Parameters

- **D** (numeric or array_like) – *Discounting-the-Illuminant* factor D in domain [0, 1].
- **F_S** (numeric or array_like) – Surround induction factor F_S.
- **F_L** (numeric or array_like) – *Lightness* induction factor F_L.
- **F_C** (numeric or array_like) – *Chroma* induction factor F_C.

References

- *[Fai13e]*
- *[LLK96]*
- *[LM96]*

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

__init__()

x.__init__(…) initializes x; see help(type(x)) for signature

Methods

count(…)	
index((value, [start, …)	Raises ValueError if the value is not present.

Nayatani (1995)

colour

XYZ_to_Nayatani95(XYZ, XYZ_n, Y_o, E_o, E_or)	Computes the *Nayatani (1995)* colour appearance model correlates.
Nayatani95_Specification	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 in domain

[0, 100].

- **XYZ_n** (array_like) – *CIE XYZ* tristimulus values of reference white in domain [0, 100].
- **Y_o** (numeric or array_like) – Luminance factor Y_o of achromatic background as percentage in domain [0.18, 1.0]
- **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 in domain [1000, 3000]
- **n** (numeric or array_like, optional) – Noise term used in the non linear chromatic adaptation model.

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

Return type *Nayatani95_Specification*

Warning: The input domain of that definition is non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].
- Input *CIE XYZ_n* tristimulus values are in domain [0, 100].

References

- *[Fai13g]*
- *[NSY95]*

Examples

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

colour.Nayatani95_Specification

class colour.**Nayatani95_Specification**

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

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

Parameters

- **Lstar_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.
- **Lstar_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

- *[Fai13g]*
- *[NSY95]*

Create new instance of Nayatani95_Specification(Lstar_P, C, h, s, Q, M, H, HC, Lstar_N)

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

RLAB

colour

XYZ_to_RLAB(XYZ, XYZ_n, Y_n[, sigma, D])	Computes the *RLAB* model color appearance correlates.
RLAB_D_FACTOR	*RLAB* colour appearance model *Discounting-the-Illuminant* factor values.
RLAB_Specification	Defines the *RLAB* colour appearance model specification.
RLAB_VIEWING_CONDITIONS	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 in domain [0, 100].
- **XYZ_n** (array_like) – *CIE XYZ* tristimulus values of reference white in domain [0, 100].
- **Y_n** (numeric or array_like) – Absolute adapting luminance in cd/m^2.
- **sigma** (numeric or array_like, optional) – Relative luminance of the surround, see colour.RLAB_VIEWING_CONDITIONS for reference.
- **D** (numeric or array_like, optional) – *Discounting-the-Illuminant* factor in domain [0, 1].

Returns *RLAB* colour appearance model specification.

Return type *RLAB_Specification*

Warning: The input domain of that definition is non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].
- Input *CIE XYZ_n* tristimulus values are in domain [0, 100].

References

- *[Fai96]*
- *[Fai13h]*

Examples

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

colour.RLAB_D_FACTOR

`colour.RLAB_D_FACTOR = CaseInsensitiveMapping({u'Soft Copy Images': ..., u'Projected Transparencies, Dark Ro`

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

References

- *[Fai96]*
- *[Fai13h]*

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

Aliases:

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

colour.RLAB_Specification

class `colour.RLAB_Specification`

Defines the *RLAB* colour appearance model specification.

This specification has field names consistent with the remaining colour appearance models in `colour.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

- *[Fai96]*
- *[Fai13h]*

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

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

`count(...)`	
`index((value, [start, ...)`	Raises ValueError if the value is not present.

colour.RLAB_VIEWING_CONDITIONS

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

References

- *[Fai96]*
- *[Fai13h]*

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

Biochemistry

- *Michaelis–Menten Kinetics*

Michaelis–Menten Kinetics

`colour.biochemistry`

`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

- *[Wikt]*

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

- *[Wikt]*

Examples

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

Colour Characterisation

- *Colour Fitting*
- *Colour Rendition Charts*

- *Cameras*
- *Displays*

Colour Fitting

`colour`

`first_order_colour_fit`(m_1, m_2)	Performs a first order colour fit from given m_1 colour array to m_2 colour array.

colour.first_order_colour_fit

`colour.`**`first_order_colour_fit`**`(`*m_1*, *m_2*`)`

Performs a first order colour fit from given m_1 colour array to m_2 colour array. The resulting colour fitting matrix is computed using multiple linear regression.

The purpose of that object is for example the matching of two *ColorChecker* colour rendition charts together.

Parameters

- **m_1** (`array_like, (3, n)`) – Test array m_1 to fit onto array m_2.
- **m_2** (`array_like, (3, n)`) – Reference array the array m_1 will be colour fitted against.

Returns Colour fitting matrix.

Return type ndarray, (3, 3)

Examples

```
>>> m_1 = 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_2 = 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]]
... )
>>> first_order_colour_fit(m_1, m_2)
array([[ 0.6982266...,  0.0307162...,  0.1621042...],
       [ 0.0689349...,  0.6757961...,  0.1643038...],
       [-0.0631495...,  0.0921247...,  0.9713415...]])
```

Colour Rendition Charts

Dataset

colour

COLOURCHECKERS	Aggregated *ColourCheckers* chromaticity coordinates.
COLOURCHECKERS_SPDS	Aggregated *ColourCheckers* spectral power distributions.

colour.COLOURCHECKERS

colour.**COLOURCHECKERS = CaseInsensitiveMapping({u'ColorChecker 2005': ..., u'babel_average': ..., u'ColorCh**

Aggregated *ColourCheckers* chromaticity coordinates.

References

- *[Bab12b]*
- *[Bab12a]*

COLOURCHECKERS [CaseInsensitiveMapping] **{'BabelColor Average', 'ColorChecker 2005', 'ColorChecker 1976'}**

Aliases:

- 'babel_average': 'BabelColor Average'
- 'cc2005': 'ColorChecker 2005'

colour.COLOURCHECKERS_SPDS

`colour.COLOURCHECKERS_SPDS = CaseInsensitiveMapping({u'babel_average': ..., u'ColorChecker N Ohta': ..., u'`

Aggregated *ColourCheckers* spectral power distributions.

References

- *[Oht97]*
- *[Bab12b]*
- *[Bab12a]*
- *[MunsellCSciencea]*

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

Aliases:

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

Cameras

`colour.characterisation`

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

colour.characterisation.RGB_SpectralSensitivities

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

Implements support for a camera *RGB* spectral sensitivities.

Parameters

- **data** (Series or Dataframe or `Signal` or `MultiSignal` or

`MultiSpectralPowerDistribution` or array_like or dict_like, optional) – Data to be stored in the multi-spectral power distribution.

- **domain** (array_like, optional) – Values to initialise the multiple `colour.SpectralPowerDistribution` 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.SpectralPowerDistribution` class instances.

Other Parameters

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

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

Methods

`__init__`([data, domain, labels])	
`align`(shape[, interpolator, . . .])	Aligns the multi-spectral power distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
`arithmetical_operation`(a, operation[, in_place])	Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.
`clone`()	
`copy`()	Returns a copy of the sub-class instance, must be reimplemented by sub-classes.
`domain_distance`(a)	Returns the euclidean distance between given array and independent domain x closest element.
`extrapolate`(shape[, extrapolator, . . .])	Extrapolates the multi-spectral power distribution in-place accordingly to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.
`fill_nan`([method, default])	Fill NaNs in independent domain x variable and corresponding range y variable using given method.
`get`()	

Continued on next page

Table 3.50 – continued from previous page

`interpolate(shape[, interpolator, ...])`	Interpolates the multi-spectral power distribution in-place accordingly to *CIE 167:2005* recommendation or given interpolation arguments.
`is_uniform()`	Returns if independent domain x variable is uniform.
`multi_signal_unpack_data([data, domain, ...])`	Unpack given data for multi-continuous signal instantiation.
`normalise([factor])`	Normalises the multi-spectral power distribution with given normalization factor.
`to_dataframe()`	Converts the continuous signal to a *Pandas* `DataFrame` class instance.
`trim(shape)`	Trims the multi-spectral power distribution wavelengths to given shape.
`trim_wavelengths(shape)`	
`zeros()`	

Dataset

`colour`

`CAMERAS_RGB_SPECTRAL_SENSITIVITIES`	Cameras *RGB* spectral sensitivities.

colour.CAMERAS_RGB_SPECTRAL_SENSITIVITIES

`colour.`**`CAMERAS_RGB_SPECTRAL_SENSITIVITIES = CaseInsensitiveMapping({u'Nikon 5100 (NPL)': ..., u'Sigma SDMeril`**

Cameras *RGB* spectral sensitivities.

References

- *[DFGM15]*

CAMERAS_RGB_SPECTRAL_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 power distributions.

colour.characterisation.RGB_DisplayPrimaries

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

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

Parameters

- **data** (Series or Dataframe or Signal or MultiSignal or MultiSpectralPowerDistribution or array_like or dict_like, optional) – Data to be stored in the multi-spectral power distribution.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralPowerDistribution 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.SpectralPowerDistribution class instances.

Other Parameters

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

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

Methods

__init__([data, domain, labels])	
align(shape[, interpolator, ...])	Aligns the multi-spectral power distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
arithmetical_operation(a, operation[, in_place])	Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.
clone()	
copy()	Returns a copy of the sub-class instance, must be reimplemented by sub-classes.
domain_distance(a)	Returns the euclidean distance between given array and independent domain x closest element.
extrapolate(shape[, extrapolator, ...])	Extrapolates the multi-spectral power distribution in-place accordingly to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.
fill_nan([method, default])	Fill NaNs in independent domain x variable and corresponding range y variable using given method.

Continued on next page

Table 3.53 – continued from previous page

get()	
interpolate(shape[, interpolator, ...])	Interpolates the multi-spectral power distribution in-place accordingly to *CIE 167:2005* recommendation or given interpolation arguments.
is_uniform()	Returns if independent domain x variable is uniform.
multi_signal_unpack_data([data, domain, ...])	Unpack given data for multi-continuous signal instantiation.
normalise([factor])	Normalises the multi-spectral power distribution with given normalization factor.
to_dataframe()	Converts the continuous signal to a *Pandas* DataFrame class instance.
trim(shape)	Trims the multi-spectral power distribution wavelengths to given shape.
trim_wavelengths(shape)	
zeros()	

Dataset

colour

DISPLAYS_RGB_PRIMARIES	Displays *RGB* primaries multi-spectral power distributions.

colour.DISPLAYS_RGB_PRIMARIES

colour.**DISPLAYS_RGB_PRIMARIES = CaseInsensitiveMapping({u'Typical CRT Brainard 1997': ..., u'Apple Studio Di**

Displays *RGB* primaries multi-spectral power distributions.

References

- *[FW98]*
- *[Mac10]*

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

Colorimetry

Spectral Data Structure

`colour`

`SpectralPowerDistribution`([data, domain])	Defines the spectral power distribution: the base object for spectral computations.
`MultiSpectralPowerDistribution`([data, . . .])	Defines multi-spectral power distribution: the base object for multi spectral computations.

Continued on next page

Table 3.55 – continued from previous page

SpectralShape([start, end, interval])	Defines the base object for spectral power distribution shape.
DEFAULT_SPECTRAL_SHAPE	Default spectral shape according to *ASTM E308-15* practise shape.
ASTME30815_PRACTISE_SHAPE	*Shape for *ASTM E308-15* practise* – (360, 780, 1).

colour.SpectralPowerDistribution

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

Defines the spectral power distribution: the base object for spectral computations.

The spectral power distribution will be initialised according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable. Extrapolation is performed according to *CIE 167:2005* recommendation.

Parameters

- **data** (Series or Signal, SpectralPowerDistribution or array_like or dict_like, optional) – Data to be stored in the spectral power distribution.
- **domain** (array_like, optional) – Values to initialise the colour.SpectralPowerDistribution.wavelength attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.SpectralPowerDistribution.wavelength attribute.

Other Parameters

- **name** (*unicode, optional*) – Spectral power distribution name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function.
- **strict_name** (*unicode, optional*) – Spectral power distribution name for figures, default to colour.SpectralPowerDistribution.name attribute value.

strict_name

wavelengths

values

shape

__init__()

extrapolate()

interpolate()

align()

trim()

normalise()

References

- *[CIET13805a]*
- *[CIET13805c]*
- *[CIET14804g]*

Examples

Instantiating a spectral power distribution with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> with numpy_print_options(suppress=True):
...     SpectralPowerDistribution(data)
SpectralPowerDistribution([[ 500.    ,    0.0651],
                           [ 520.    ,    0.0705],
                           [ 540.    ,    0.0772],
                           [ 560.    ,    0.087 ],
                           [ 580.    ,    0.1128],
                           [ 600.    ,    0.136 ]],
                          interpolator=SpragueInterpolator,
                          interpolator_args={},
                          extrapolator=Extrapolator,
                          extrapolator_args={...})
```

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

```
>>> data[510] = 0.31416
>>> with numpy_print_options(suppress=True):
...     SpectralPowerDistribution(data)
SpectralPowerDistribution([[ 500.     ,    0.0651 ],
                           [ 510.     ,    0.31416],
                           [ 520.     ,    0.0705 ],
                           [ 540.     ,    0.0772 ],
                           [ 560.     ,    0.087  ],
                           [ 580.     ,    0.1128 ],
                           [ 600.     ,    0.136  ]],
                          interpolator=CubicSplineInterpolator,
                          interpolator_args={},
                          extrapolator=Extrapolator,
                          extrapolator_args={...})
```

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

Methods

`__init__`([data, domain])	
`align`(shape[, interpolator, …])	Aligns the spectral power distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
`arithmetical_operation`(a, operation[, in_place])	Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.
`clone`()	
`copy`()	Returns a copy of the sub-class instance, must be reimplemented by sub-classes.
`domain_distance`(a)	Returns the euclidean distance between given array and independent domain x closest element.
`extrapolate`(shape[, extrapolator, …])	Extrapolates the spectral power distribution in-place according to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.
`fill_nan`([method, default])	Fill NaNs in independent domain x variable and corresponding range y variable using given method.
`get`()	
`interpolate`(shape[, interpolator, …])	Interpolates the spectral power distribution in-place according to *CIE 167:2005* recommendation or given interpolation arguments.
`is_uniform`()	Returns if independent domain x variable is uniform.
`normalise`([factor])	Normalises the spectral power distribution using given normalization factor.
`signal_unpack_data`([data, domain, dtype])	Unpack given data for continuous signal instantiation.
`to_series`()	Converts the continuous signal to a *Pandas* `Series` class instance.
`trim`(shape)	Trims the spectral power distribution wavelengths to given spectral shape.
`trim_wavelengths`(shape)	
`zeros`()	

colour.MultiSpectralPowerDistribution

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

Defines multi-spectral power distribution: the base object for multi spectral computations. It is used to model colour matching functions, display primaries, camera sensitivities, etc…

The multi-spectral power distribution will be initialised according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable. Extrapolation is performed according to *CIE 167:2005* recommendation.

Parameters

- **data** (`Series` or `Dataframe` or `Signal` or `MultiSignal` or

`MultiSpectralPowerDistribution` or `array_like` or `dict_like`, `optional`) – Data to be stored in the multi-spectral power distribution.

- **domain** (`array_like`, `optional`) – Values to initialise the multiple `colour.SpectralPowerDistribution` 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.SpectralPowerDistribution` class instances.

Other Parameters

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

`strict_name`

`strict_labels`

`wavelengths`

`values`

`shape`

`extrapolate`()

`interpolate`()

`align`()

`trim`()

`normalise`()

References

- *[CIET13805a]*
- *[CIET13805c]*
- *[CIET14804g]*

Examples

Instantiating a multi-spectral power distribution with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> labels = ('x_bar', 'y_bar', 'z_bar')
>>> with numpy_print_options(suppress=True):
...     MultiSpectralPowerDistribution(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_args={},
             ... extrapolator=Extrapolator,
             ... extrapolator_args={...})
```

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

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> with numpy_print_options(suppress=True):
...     MultiSpectralPowerDistribution(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_args={},
             ... extrapolator=Extrapolator,
             ... extrapolator_args={...})
```

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

Methods

__init__([data, domain, labels])	

Continued on next page

Table 3.57 – continued from previous page

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

colour.SpectralShape

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

Defines the base object for spectral power distribution shape.

Parameters

- **start** (numeric, optional) – Wavelength λ_i range start in nm.
- **end** (numeric, optional) – Wavelength λ_i range end in nm.
- **interval** (numeric, optional) – Wavelength λ_i range interval.

start

end

interval

boundaries

__str__()

__repr__()

__iter__()

__contains__()

__len__()

__eq__()

__ne__()

range()

Examples

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

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

Methods

__init__([start, end, interval])	
range([dtype])	Returns an iterable range for the spectral shape.

colour.DEFAULT_SPECTRAL_SHAPE

colour.**DEFAULT_SPECTRAL_SHAPE = SpectralShape(360, 780, 1)**

Default spectral shape according to *ASTM E308-15* practise shape.

DEFAULT_SPECTRAL_SHAPE : SpectralShape

colour.ASTME30815_PRACTISE_SHAPE

colour.**ASTME30815_PRACTISE_SHAPE = SpectralShape(360, 780, 1)**

*Shape for *ASTM E308-15* practise* – (360, 780, 1).

References

- *[ASTMInternational15]*

ASTME30815_PRACTISE_SHAPE : SpectralShape

Spectral Data Generation

colour

blackbody_spd(temperature[, shape, c1, c2, n])	Returns the spectral power distribution of the planckian radiator for given temperature $T[K]$.
CIE_standard_illuminant_A_function(wl)	*CIE Standard Illuminant A* is intended to represent typical, domestic,
D_illuminant_relative_spd(xy)	Returns the relative spectral power distribution of given *CIE Standard Illuminant D Series* using given *xy* chromaticity coordinates.
constant_spd(k[, shape, dtype])	Returns a spectral power distribution of given spectral shape filled with constant k values.
ones_spd([shape])	Returns a spectral power distribution of given spectral shape filled with ones.
zeros_spd([shape])	Returns a spectral power distribution of given spectral shape filled with zeros.

colour.blackbody_spd

colour.**blackbody_spd**(*temperature*, *shape=SpectralShape(360, 780, 1)*, *c1=3.741771e-16*, *c2=0.014388*, *n=1*)

Returns the spectral power distribution of the planckian radiator for given temperature $T[K]$.

Parameters

- **temperature** (numeric) – Temperature $T[K]$ in kelvin degrees.
- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral power 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 current International Temperature Scale (ITS-90) *(Preston-Thomas, 1990; Mielenz et aI., 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 power distribution.

Return type *SpectralPowerDistribution*

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> from colour.utilities import numpy_print_options
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> with numpy_print_options(suppress=True):
...     blackbody_spd(5000, cmfs.shape)
SpectralPowerDistribution([[  3.60000000e+02,   6.65427827e+12],
                           [  3.61000000e+02,   6.70960528e+12],
                           [  3.62000000e+02,   6.76482512e+12],
                           [  3.63000000e+02,   6.81993308e+12],
```

```
       [  3.64000000e+02,   6.87492449e+12],
       [  3.65000000e+02,   6.92979475e+12],
       [  3.66000000e+02,   6.98453932e+12],
       [  3.67000000e+02,   7.03915372e+12],
       [  3.68000000e+02,   7.09363351e+12],
       [  3.69000000e+02,   7.14797433e+12],
       [  3.70000000e+02,   7.20217187e+12],
       [  3.71000000e+02,   7.25622190e+12],
       [  3.72000000e+02,   7.31012021e+12],
       [  3.73000000e+02,   7.36386268e+12],
       [  3.74000000e+02,   7.41744525e+12],
       [  3.75000000e+02,   7.47086391e+12],
       [  3.76000000e+02,   7.52411471e+12],
       [  3.77000000e+02,   7.57719377e+12],
       [  3.78000000e+02,   7.63009726e+12],
       [  3.79000000e+02,   7.68282141e+12],
       [  3.80000000e+02,   7.73536252e+12],
       [  3.81000000e+02,   7.78771695e+12],
       [  3.82000000e+02,   7.83988111e+12],
       [  3.83000000e+02,   7.89185148e+12],
       [  3.84000000e+02,   7.94362458e+12],
       [  3.85000000e+02,   7.99519703e+12],
       [  3.86000000e+02,   8.04656547e+12],
       [  3.87000000e+02,   8.09772662e+12],
       [  3.88000000e+02,   8.14867726e+12],
       [  3.89000000e+02,   8.19941421e+12],
       [  3.90000000e+02,   8.24993438e+12],
       [  3.91000000e+02,   8.30023471e+12],
       [  3.92000000e+02,   8.35031222e+12],
       [  3.93000000e+02,   8.40016398e+12],
       [  3.94000000e+02,   8.44978711e+12],
       [  3.95000000e+02,   8.49917881e+12],
       [  3.96000000e+02,   8.54833632e+12],
       [  3.97000000e+02,   8.59725693e+12],
       [  3.98000000e+02,   8.64593802e+12],
       [  3.99000000e+02,   8.69437700e+12],
       [  4.00000000e+02,   8.74257133e+12],
       [  4.01000000e+02,   8.79051856e+12],
       [  4.02000000e+02,   8.83821626e+12],
       [  4.03000000e+02,   8.88566209e+12],
       [  4.04000000e+02,   8.93285373e+12],
       [  4.05000000e+02,   8.97978893e+12],
       [  4.06000000e+02,   9.02646551e+12],
       [  4.07000000e+02,   9.07288133e+12],
       [  4.08000000e+02,   9.11903431e+12],
       [  4.09000000e+02,   9.16492240e+12],
       [  4.10000000e+02,   9.21054364e+12],
       [  4.11000000e+02,   9.25589609e+12],
       [  4.12000000e+02,   9.30097789e+12],
       [  4.13000000e+02,   9.34578722e+12],
       [  4.14000000e+02,   9.39032230e+12],
       [  4.15000000e+02,   9.43458143e+12],
       [  4.16000000e+02,   9.47856292e+12],
       [  4.17000000e+02,   9.52226517e+12],
       [  4.18000000e+02,   9.56568661e+12],
       [  4.19000000e+02,   9.60882571e+12],
       [  4.20000000e+02,   9.65168102e+12],
       [  4.21000000e+02,   9.69425111e+12],
```

```
[  4.22000000e+02,   9.73653461e+12],
[  4.23000000e+02,   9.77853020e+12],
[  4.24000000e+02,   9.82023659e+12],
[  4.25000000e+02,   9.86165257e+12],
[  4.26000000e+02,   9.90277693e+12],
[  4.27000000e+02,   9.94360856e+12],
[  4.28000000e+02,   9.98414634e+12],
[  4.29000000e+02,   1.00243892e+13],
[  4.30000000e+02,   1.00643363e+13],
[  4.31000000e+02,   1.01039864e+13],
[  4.32000000e+02,   1.01433388e+13],
[  4.33000000e+02,   1.01823926e+13],
[  4.34000000e+02,   1.02211468e+13],
[  4.35000000e+02,   1.02596009e+13],
[  4.36000000e+02,   1.02977539e+13],
[  4.37000000e+02,   1.03356052e+13],
[  4.38000000e+02,   1.03731541e+13],
[  4.39000000e+02,   1.04104000e+13],
[  4.40000000e+02,   1.04473423e+13],
[  4.41000000e+02,   1.04839805e+13],
[  4.42000000e+02,   1.05203140e+13],
[  4.43000000e+02,   1.05563424e+13],
[  4.44000000e+02,   1.05920652e+13],
[  4.45000000e+02,   1.06274821e+13],
[  4.46000000e+02,   1.06625927e+13],
[  4.47000000e+02,   1.06973967e+13],
[  4.48000000e+02,   1.07318937e+13],
[  4.49000000e+02,   1.07660835e+13],
[  4.50000000e+02,   1.07999660e+13],
[  4.51000000e+02,   1.08335408e+13],
[  4.52000000e+02,   1.08668080e+13],
[  4.53000000e+02,   1.08997673e+13],
[  4.54000000e+02,   1.09324187e+13],
[  4.55000000e+02,   1.09647621e+13],
[  4.56000000e+02,   1.09967975e+13],
[  4.57000000e+02,   1.10285249e+13],
[  4.58000000e+02,   1.10599443e+13],
[  4.59000000e+02,   1.10910559e+13],
[  4.60000000e+02,   1.11218598e+13],
[  4.61000000e+02,   1.11523560e+13],
[  4.62000000e+02,   1.11825447e+13],
[  4.63000000e+02,   1.12124262e+13],
[  4.64000000e+02,   1.12420006e+13],
[  4.65000000e+02,   1.12712681e+13],
[  4.66000000e+02,   1.13002292e+13],
[  4.67000000e+02,   1.13288840e+13],
[  4.68000000e+02,   1.13572329e+13],
[  4.69000000e+02,   1.13852762e+13],
[  4.70000000e+02,   1.14130144e+13],
[  4.71000000e+02,   1.14404478e+13],
[  4.72000000e+02,   1.14675768e+13],
[  4.73000000e+02,   1.14944020e+13],
[  4.74000000e+02,   1.15209237e+13],
[  4.75000000e+02,   1.15471425e+13],
[  4.76000000e+02,   1.15730590e+13],
[  4.77000000e+02,   1.15986736e+13],
[  4.78000000e+02,   1.16239869e+13],
[  4.79000000e+02,   1.16489996e+13],
```

```
                    [   4.80000000e+02,    1.16737122e+13],
                    [   4.81000000e+02,    1.16981253e+13],
                    [   4.82000000e+02,    1.17222397e+13],
                    [   4.83000000e+02,    1.17460559e+13],
                    [   4.84000000e+02,    1.17695747e+13],
                    [   4.85000000e+02,    1.17927969e+13],
                    [   4.86000000e+02,    1.18157230e+13],
                    [   4.87000000e+02,    1.18383540e+13],
                    [   4.88000000e+02,    1.18606904e+13],
                    [   4.89000000e+02,    1.18827333e+13],
                    [   4.90000000e+02,    1.19044832e+13],
                    [   4.91000000e+02,    1.19259412e+13],
                    [   4.92000000e+02,    1.19471079e+13],
                    [   4.93000000e+02,    1.19679843e+13],
                    [   4.94000000e+02,    1.19885712e+13],
                    [   4.95000000e+02,    1.20088695e+13],
                    [   4.96000000e+02,    1.20288802e+13],
                    [   4.97000000e+02,    1.20486041e+13],
                    [   4.98000000e+02,    1.20680421e+13],
                    [   4.99000000e+02,    1.20871953e+13],
                    [   5.00000000e+02,    1.21060645e+13],
                    [   5.01000000e+02,    1.21246508e+13],
                    [   5.02000000e+02,    1.21429552e+13],
                    [   5.03000000e+02,    1.21609785e+13],
                    [   5.04000000e+02,    1.21787220e+13],
                    [   5.05000000e+02,    1.21961865e+13],
                    [   5.06000000e+02,    1.22133731e+13],
                    [   5.07000000e+02,    1.22302829e+13],
                    [   5.08000000e+02,    1.22469170e+13],
                    [   5.09000000e+02,    1.22632763e+13],
                    [   5.10000000e+02,    1.22793620e+13],
                    [   5.11000000e+02,    1.22951752e+13],
                    [   5.12000000e+02,    1.23107171e+13],
                    [   5.13000000e+02,    1.23259886e+13],
                    [   5.14000000e+02,    1.23409909e+13],
                    [   5.15000000e+02,    1.23557252e+13],
                    [   5.16000000e+02,    1.23701926e+13],
                    [   5.17000000e+02,    1.23843943e+13],
                    [   5.18000000e+02,    1.23983314e+13],
                    [   5.19000000e+02,    1.24120051e+13],
                    [   5.20000000e+02,    1.24254166e+13],
                    [   5.21000000e+02,    1.24385670e+13],
                    [   5.22000000e+02,    1.24514576e+13],
                    [   5.23000000e+02,    1.24640896e+13],
                    [   5.24000000e+02,    1.24764641e+13],
                    [   5.25000000e+02,    1.24885824e+13],
                    [   5.26000000e+02,    1.25004457e+13],
                    [   5.27000000e+02,    1.25120552e+13],
                    [   5.28000000e+02,    1.25234122e+13],
                    [   5.29000000e+02,    1.25345178e+13],
                    [   5.30000000e+02,    1.25453735e+13],
                    [   5.31000000e+02,    1.25559803e+13],
                    [   5.32000000e+02,    1.25663396e+13],
                    [   5.33000000e+02,    1.25764527e+13],
                    [   5.34000000e+02,    1.25863207e+13],
                    [   5.35000000e+02,    1.25959449e+13],
                    [   5.36000000e+02,    1.26053268e+13],
                    [   5.37000000e+02,    1.26144674e+13],
```

```
                                 [  5.38000000e+02,   1.26233681e+13],
                                 [  5.39000000e+02,   1.26320302e+13],
                                 [  5.40000000e+02,   1.26404551e+13],
                                 [  5.41000000e+02,   1.26486438e+13],
                                 [  5.42000000e+02,   1.26565979e+13],
                                 [  5.43000000e+02,   1.26643185e+13],
                                 [  5.44000000e+02,   1.26718071e+13],
                                 [  5.45000000e+02,   1.26790648e+13],
                                 [  5.46000000e+02,   1.26860930e+13],
                                 [  5.47000000e+02,   1.26928930e+13],
                                 [  5.48000000e+02,   1.26994662e+13],
                                 [  5.49000000e+02,   1.27058138e+13],
                                 [  5.50000000e+02,   1.27119372e+13],
                                 [  5.51000000e+02,   1.27178376e+13],
                                 [  5.52000000e+02,   1.27235164e+13],
                                 [  5.53000000e+02,   1.27289750e+13],
                                 [  5.54000000e+02,   1.27342146e+13],
                                 [  5.55000000e+02,   1.27392366e+13],
                                 [  5.56000000e+02,   1.27440423e+13],
                                 [  5.57000000e+02,   1.27486330e+13],
                                 [  5.58000000e+02,   1.27530100e+13],
                                 [  5.59000000e+02,   1.27571748e+13],
                                 [  5.60000000e+02,   1.27611285e+13],
                                 [  5.61000000e+02,   1.27648725e+13],
                                 [  5.62000000e+02,   1.27684083e+13],
                                 [  5.63000000e+02,   1.27717370e+13],
                                 [  5.64000000e+02,   1.27748600e+13],
                                 [  5.65000000e+02,   1.27777787e+13],
                                 [  5.66000000e+02,   1.27804943e+13],
                                 [  5.67000000e+02,   1.27830082e+13],
                                 [  5.68000000e+02,   1.27853217e+13],
                                 [  5.69000000e+02,   1.27874362e+13],
                                 [  5.70000000e+02,   1.27893529e+13],
                                 [  5.71000000e+02,   1.27910732e+13],
                                 [  5.72000000e+02,   1.27925984e+13],
                                 [  5.73000000e+02,   1.27939299e+13],
                                 [  5.74000000e+02,   1.27950689e+13],
                                 [  5.75000000e+02,   1.27960167e+13],
                                 [  5.76000000e+02,   1.27967747e+13],
                                 [  5.77000000e+02,   1.27973442e+13],
                                 [  5.78000000e+02,   1.27977264e+13],
                                 [  5.79000000e+02,   1.27979228e+13],
                                 [  5.80000000e+02,   1.27979346e+13],
                                 [  5.81000000e+02,   1.27977630e+13],
                                 [  5.82000000e+02,   1.27974095e+13],
                                 [  5.83000000e+02,   1.27968753e+13],
                                 [  5.84000000e+02,   1.27961617e+13],
                                 [  5.85000000e+02,   1.27952700e+13],
                                 [  5.86000000e+02,   1.27942015e+13],
                                 [  5.87000000e+02,   1.27929575e+13],
                                 [  5.88000000e+02,   1.27915392e+13],
                                 [  5.89000000e+02,   1.27899480e+13],
                                 [  5.90000000e+02,   1.27881852e+13],
                                 [  5.91000000e+02,   1.27862519e+13],
                                 [  5.92000000e+02,   1.27841495e+13],
                                 [  5.93000000e+02,   1.27818793e+13],
                                 [  5.94000000e+02,   1.27794424e+13],
                                 [  5.95000000e+02,   1.27768403e+13],
```

```
[  5.96000000e+02,   1.27740741e+13],
[  5.97000000e+02,   1.27711451e+13],
[  5.98000000e+02,   1.27680546e+13],
[  5.99000000e+02,   1.27648037e+13],
[  6.00000000e+02,   1.27613938e+13],
[  6.01000000e+02,   1.27578261e+13],
[  6.02000000e+02,   1.27541018e+13],
[  6.03000000e+02,   1.27502222e+13],
[  6.04000000e+02,   1.27461885e+13],
[  6.05000000e+02,   1.27420020e+13],
[  6.06000000e+02,   1.27376637e+13],
[  6.07000000e+02,   1.27331750e+13],
[  6.08000000e+02,   1.27285371e+13],
[  6.09000000e+02,   1.27237512e+13],
[  6.10000000e+02,   1.27188185e+13],
[  6.11000000e+02,   1.27137402e+13],
[  6.12000000e+02,   1.27085175e+13],
[  6.13000000e+02,   1.27031516e+13],
[  6.14000000e+02,   1.26976436e+13],
[  6.15000000e+02,   1.26919949e+13],
[  6.16000000e+02,   1.26862064e+13],
[  6.17000000e+02,   1.26802795e+13],
[  6.18000000e+02,   1.26742153e+13],
[  6.19000000e+02,   1.26680149e+13],
[  6.20000000e+02,   1.26616795e+13],
[  6.21000000e+02,   1.26552103e+13],
[  6.22000000e+02,   1.26486085e+13],
[  6.23000000e+02,   1.26418751e+13],
[  6.24000000e+02,   1.26350113e+13],
[  6.25000000e+02,   1.26280183e+13],
[  6.26000000e+02,   1.26208972e+13],
[  6.27000000e+02,   1.26136491e+13],
[  6.28000000e+02,   1.26062751e+13],
[  6.29000000e+02,   1.25987764e+13],
[  6.30000000e+02,   1.25911540e+13],
[  6.31000000e+02,   1.25834092e+13],
[  6.32000000e+02,   1.25755429e+13],
[  6.33000000e+02,   1.25675563e+13],
[  6.34000000e+02,   1.25594505e+13],
[  6.35000000e+02,   1.25512265e+13],
[  6.36000000e+02,   1.25428855e+13],
[  6.37000000e+02,   1.25344285e+13],
[  6.38000000e+02,   1.25258566e+13],
[  6.39000000e+02,   1.25171709e+13],
[  6.40000000e+02,   1.25083724e+13],
[  6.41000000e+02,   1.24994622e+13],
[  6.42000000e+02,   1.24904413e+13],
[  6.43000000e+02,   1.24813108e+13],
[  6.44000000e+02,   1.24720718e+13],
[  6.45000000e+02,   1.24627252e+13],
[  6.46000000e+02,   1.24532721e+13],
[  6.47000000e+02,   1.24437136e+13],
[  6.48000000e+02,   1.24340506e+13],
[  6.49000000e+02,   1.24242842e+13],
[  6.50000000e+02,   1.24144153e+13],
[  6.51000000e+02,   1.24044450e+13],
[  6.52000000e+02,   1.23943743e+13],
[  6.53000000e+02,   1.23842042e+13],
```

```
                [  6.54000000e+02,   1.23739356e+13],
                [  6.55000000e+02,   1.23635696e+13],
                [  6.56000000e+02,   1.23531072e+13],
                [  6.57000000e+02,   1.23425492e+13],
                [  6.58000000e+02,   1.23318967e+13],
                [  6.59000000e+02,   1.23211506e+13],
                [  6.60000000e+02,   1.23103120e+13],
                [  6.61000000e+02,   1.22993816e+13],
                [  6.62000000e+02,   1.22883606e+13],
                [  6.63000000e+02,   1.22772498e+13],
                [  6.64000000e+02,   1.22660502e+13],
                [  6.65000000e+02,   1.22547627e+13],
                [  6.66000000e+02,   1.22433883e+13],
                [  6.67000000e+02,   1.22319278e+13],
                [  6.68000000e+02,   1.22203821e+13],
                [  6.69000000e+02,   1.22087523e+13],
                [  6.70000000e+02,   1.21970391e+13],
                [  6.71000000e+02,   1.21852435e+13],
                [  6.72000000e+02,   1.21733664e+13],
                [  6.73000000e+02,   1.21614087e+13],
                [  6.74000000e+02,   1.21493712e+13],
                [  6.75000000e+02,   1.21372548e+13],
                [  6.76000000e+02,   1.21250605e+13],
                [  6.77000000e+02,   1.21127890e+13],
                [  6.78000000e+02,   1.21004413e+13],
                [  6.79000000e+02,   1.20880182e+13],
                [  6.80000000e+02,   1.20755205e+13],
                [  6.81000000e+02,   1.20629491e+13],
                [  6.82000000e+02,   1.20503049e+13],
                [  6.83000000e+02,   1.20375887e+13],
                [  6.84000000e+02,   1.20248012e+13],
                [  6.85000000e+02,   1.20119434e+13],
                [  6.86000000e+02,   1.19990161e+13],
                [  6.87000000e+02,   1.19860200e+13],
                [  6.88000000e+02,   1.19729560e+13],
                [  6.89000000e+02,   1.19598249e+13],
                [  6.90000000e+02,   1.19466275e+13],
                [  6.91000000e+02,   1.19333646e+13],
                [  6.92000000e+02,   1.19200370e+13],
                [  6.93000000e+02,   1.19066454e+13],
                [  6.94000000e+02,   1.18931907e+13],
                [  6.95000000e+02,   1.18796736e+13],
                [  6.96000000e+02,   1.18660949e+13],
                [  6.97000000e+02,   1.18524554e+13],
                [  6.98000000e+02,   1.18387558e+13],
                [  6.99000000e+02,   1.18249969e+13],
                [  7.00000000e+02,   1.18111794e+13],
                [  7.01000000e+02,   1.17973040e+13],
                [  7.02000000e+02,   1.17833716e+13],
                [  7.03000000e+02,   1.17693829e+13],
                [  7.04000000e+02,   1.17553385e+13],
                [  7.05000000e+02,   1.17412392e+13],
                [  7.06000000e+02,   1.17270858e+13],
                [  7.07000000e+02,   1.17128789e+13],
                [  7.08000000e+02,   1.16986192e+13],
                [  7.09000000e+02,   1.16843075e+13],
                [  7.10000000e+02,   1.16699445e+13],
                [  7.11000000e+02,   1.16555309e+13],
```

```
       [  7.12000000e+02,   1.16410673e+13],
       [  7.13000000e+02,   1.16265544e+13],
       [  7.14000000e+02,   1.16119930e+13],
       [  7.15000000e+02,   1.15973836e+13],
       [  7.16000000e+02,   1.15827271e+13],
       [  7.17000000e+02,   1.15680240e+13],
       [  7.18000000e+02,   1.15532749e+13],
       [  7.19000000e+02,   1.15384807e+13],
       [  7.20000000e+02,   1.15236419e+13],
       [  7.21000000e+02,   1.15087591e+13],
       [  7.22000000e+02,   1.14938331e+13],
       [  7.23000000e+02,   1.14788644e+13],
       [  7.24000000e+02,   1.14638537e+13],
       [  7.25000000e+02,   1.14488017e+13],
       [  7.26000000e+02,   1.14337088e+13],
       [  7.27000000e+02,   1.14185759e+13],
       [  7.28000000e+02,   1.14034034e+13],
       [  7.29000000e+02,   1.13881921e+13],
       [  7.30000000e+02,   1.13729424e+13],
       [  7.31000000e+02,   1.13576551e+13],
       [  7.32000000e+02,   1.13423307e+13],
       [  7.33000000e+02,   1.13269698e+13],
       [  7.34000000e+02,   1.13115730e+13],
       [  7.35000000e+02,   1.12961409e+13],
       [  7.36000000e+02,   1.12806741e+13],
       [  7.37000000e+02,   1.12651731e+13],
       [  7.38000000e+02,   1.12496385e+13],
       [  7.39000000e+02,   1.12340710e+13],
       [  7.40000000e+02,   1.12184710e+13],
       [  7.41000000e+02,   1.12028391e+13],
       [  7.42000000e+02,   1.11871759e+13],
       [  7.43000000e+02,   1.11714819e+13],
       [  7.44000000e+02,   1.11557577e+13],
       [  7.45000000e+02,   1.11400038e+13],
       [  7.46000000e+02,   1.11242208e+13],
       [  7.47000000e+02,   1.11084092e+13],
       [  7.48000000e+02,   1.10925695e+13],
       [  7.49000000e+02,   1.10767023e+13],
       [  7.50000000e+02,   1.10608080e+13],
       [  7.51000000e+02,   1.10448872e+13],
       [  7.52000000e+02,   1.10289405e+13],
       [  7.53000000e+02,   1.10129683e+13],
       [  7.54000000e+02,   1.09969711e+13],
       [  7.55000000e+02,   1.09809495e+13],
       [  7.56000000e+02,   1.09649039e+13],
       [  7.57000000e+02,   1.09488348e+13],
       [  7.58000000e+02,   1.09327427e+13],
       [  7.59000000e+02,   1.09166282e+13],
       [  7.60000000e+02,   1.09004917e+13],
       [  7.61000000e+02,   1.08843336e+13],
       [  7.62000000e+02,   1.08681545e+13],
       [  7.63000000e+02,   1.08519548e+13],
       [  7.64000000e+02,   1.08357350e+13],
       [  7.65000000e+02,   1.08194956e+13],
       [  7.66000000e+02,   1.08032370e+13],
       [  7.67000000e+02,   1.07869596e+13],
       [  7.68000000e+02,   1.07706640e+13],
       [  7.69000000e+02,   1.07543506e+13],
```

```
[  7.70000000e+02,   1.07380198e+13],
[  7.71000000e+02,   1.07216721e+13],
[  7.72000000e+02,   1.07053078e+13],
[  7.73000000e+02,   1.06889276e+13],
[  7.74000000e+02,   1.06725317e+13],
[  7.75000000e+02,   1.06561206e+13],
[  7.76000000e+02,   1.06396947e+13],
[  7.77000000e+02,   1.06232545e+13],
[  7.78000000e+02,   1.06068004e+13],
[  7.79000000e+02,   1.05903327e+13],
[  7.80000000e+02,   1.05738520e+13],
[  7.81000000e+02,   1.05573585e+13],
[  7.82000000e+02,   1.05408527e+13],
[  7.83000000e+02,   1.05243351e+13],
[  7.84000000e+02,   1.05078060e+13],
[  7.85000000e+02,   1.04912657e+13],
[  7.86000000e+02,   1.04747147e+13],
[  7.87000000e+02,   1.04581535e+13],
[  7.88000000e+02,   1.04415822e+13],
[  7.89000000e+02,   1.04250014e+13],
[  7.90000000e+02,   1.04084115e+13],
[  7.91000000e+02,   1.03918127e+13],
[  7.92000000e+02,   1.03752055e+13],
[  7.93000000e+02,   1.03585902e+13],
[  7.94000000e+02,   1.03419672e+13],
[  7.95000000e+02,   1.03253369e+13],
[  7.96000000e+02,   1.03086995e+13],
[  7.97000000e+02,   1.02920556e+13],
[  7.98000000e+02,   1.02754053e+13],
[  7.99000000e+02,   1.02587492e+13],
[  8.00000000e+02,   1.02420874e+13],
[  8.01000000e+02,   1.02254204e+13],
[  8.02000000e+02,   1.02087485e+13],
[  8.03000000e+02,   1.01920720e+13],
[  8.04000000e+02,   1.01753913e+13],
[  8.05000000e+02,   1.01587067e+13],
[  8.06000000e+02,   1.01420186e+13],
[  8.07000000e+02,   1.01253271e+13],
[  8.08000000e+02,   1.01086328e+13],
[  8.09000000e+02,   1.00919358e+13],
[  8.10000000e+02,   1.00752366e+13],
[  8.11000000e+02,   1.00585353e+13],
[  8.12000000e+02,   1.00418324e+13],
[  8.13000000e+02,   1.00251282e+13],
[  8.14000000e+02,   1.00084228e+13],
[  8.15000000e+02,   9.99171677e+12],
[  8.16000000e+02,   9.97501023e+12],
[  8.17000000e+02,   9.95830354e+12],
[  8.18000000e+02,   9.94159699e+12],
[  8.19000000e+02,   9.92489086e+12],
[  8.20000000e+02,   9.90818544e+12],
[  8.21000000e+02,   9.89148102e+12],
[  8.22000000e+02,   9.87477789e+12],
[  8.23000000e+02,   9.85807631e+12],
[  8.24000000e+02,   9.84137658e+12],
[  8.25000000e+02,   9.82467896e+12],
[  8.26000000e+02,   9.80798374e+12],
[  8.27000000e+02,   9.79129116e+12],
```

```
                                 [  8.28000000e+02,   9.77460152e+12],
                                 [  8.29000000e+02,   9.75791506e+12],
                                 [  8.30000000e+02,   9.74123205e+12]],
                                interpolator=SpragueInterpolator,
                                interpolator_args={},
                                extrapolator=Extrapolator,
                                extrapolator_args={...})
```

colour.CIE_standard_illuminant_A_function

colour.**CIE_standard_illuminant_A_function**(*wl*)

CIE Standard Illuminant A is intended to represent typical, domestic, tungsten-filament lighting.

Its relative spectral power 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 **wl** (array_like) – Wavelength to evaluate the function at.

Returns *CIE Standard Illuminant A* value at given wavelength.

Return type ndarray

References

- *[CIET14804a]*

Examples

```
>>> wl = np.array([560, 580, 581.5])
>>> CIE_standard_illuminant_A_function(wl)
array([ 100.        ,  114.4363383...,  115.5285063...])
```

colour.D_illuminant_relative_spd

colour.**D_illuminant_relative_spd**(*xy*)

Returns the relative spectral power distribution of given *CIE Standard Illuminant D Series* using given *xy* chromaticity coordinates.

References

- *[Lin07]*
- *[WS00b]*

Parameters **xy** (array_like) – *xy* chromaticity coordinates.

Returns *CIE Standard Illuminant D Series* relative spectral power distribution.

Return type *SpectralPowerDistribution*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> xy = np.array([0.34570, 0.35850])
>>> with numpy_print_options(suppress=True):
...     D_illuminant_relative_spd(xy)
SpectralPowerDistribution([[ 300.        ,    0.0193039...],
                           [ 310.        ,    2.1265303...],
                           [ 320.        ,    7.9867359...],
                           [ 330.        ,   15.1666959...],
                           [ 340.        ,   18.3413202...],
                           [ 350.        ,   21.3757973...],
                           [ 360.        ,   24.2528862...],
                           [ 370.        ,   26.2782171...],
                           [ 380.        ,   24.7348842...],
                           [ 390.        ,   30.0518667...],
                           [ 400.        ,   49.458942 ...],
                           [ 410.        ,   56.6929605...],
                           [ 420.        ,   60.1981682...],
                           [ 430.        ,   57.9390276...],
                           [ 440.        ,   74.9047554...],
                           [ 450.        ,   87.3151258...],
                           [ 460.        ,   90.6691236...],
                           [ 470.        ,   91.4109985...],
                           [ 480.        ,   95.1362798...],
                           [ 490.        ,   91.9956940...],
                           [ 500.        ,   95.7488852...],
                           [ 510.        ,   96.6315995...],
                           [ 520.        ,   97.1308377...],
                           [ 530.        ,  102.0961518...],
                           [ 540.        ,  100.7580555...],
                           [ 550.        ,  102.3164095...],
                           [ 560.        ,  100.        ...],
                           [ 570.        ,   97.7339937...],
                           [ 580.        ,   98.9175842...],
                           [ 590.        ,   93.5440898...],
                           [ 600.        ,   97.7548532...],
                           [ 610.        ,   99.3559831...],
                           [ 620.        ,   99.1396431...],
                           [ 630.        ,   95.8275899...],
                           [ 640.        ,   99.0028159...],
                           [ 650.        ,   95.8307955...],
                           [ 660.        ,   98.3850717...],
                           [ 670.        ,  103.2245516...],
                           [ 680.        ,   99.3672578...],
                           [ 690.        ,   87.5676019...],
                           [ 700.        ,   91.8218781...],
                           [ 710.        ,   93.0772354...],
                           [ 720.        ,   77.0098456...],
                           [ 730.        ,   86.6795856...],
                           [ 740.        ,   92.7570922...],
                           [ 750.        ,   78.3784557...],
                           [ 760.        ,   57.8075859...],
                           [ 770.        ,   83.0873522...],
                           [ 780.        ,   78.4245724...],
                           [ 790.        ,   79.7098456...],
                           [ 800.        ,   73.5435857...],
                           [ 810.        ,   64.0424558...],
```

```
                              [ 820.         ,    70.9121958...],
                              [ 830.         ,    74.5862223...]],
                             interpolator=SpragueInterpolator,
                             interpolator_args={},
                             extrapolator=Extrapolator,
                             extrapolator_args={...})
```

colour.constant_spd

colour.**constant_spd**(*k*, *shape=SpectralShape(360, 780, 1)*, *dtype=<type 'numpy.float64'>*)

Returns a spectral power distribution of given spectral shape filled with constant k values.

Parameters

- **k** (numeric) – Constant k to fill the spectral power distribution with.
- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral power distribution.
- **dtype** (type) – Data type used for the spectral power distribution.

Returns Constant k to filled spectral power distribution.

Return type *SpectralPowerDistribution*

Notes

- By default, the spectral power distribution will use the shape given by colour.DEFAULT_SPECTRAL_SHAPE attribute.

Examples

```
>>> spd = constant_spd(100)
>>> spd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> spd[400]
100.0
```

colour.ones_spd

colour.**ones_spd**(*shape=SpectralShape(360, 780, 1)*)

Returns a spectral power distribution of given spectral shape filled with ones.

Parameters **shape** (SpectralShape, optional) – Spectral shape used to create the spectral power distribution.

Returns Ones filled spectral power distribution.

Return type *SpectralPowerDistribution*

Notes

- By default, the spectral power distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

Examples

```
>>> spd = ones_spd()
>>> spd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> spd[400]
1.0
```

colour.zeros_spd

`colour.zeros_spd`(*shape=SpectralShape(360, 780, 1)*)

Returns a spectral power distribution of given spectral shape filled with zeros.

Parameters shape (`SpectralShape`, `optional`) – Spectral shape used to create the spectral power distribution.

Returns Zeros filled spectral power distribution.

Return type *SpectralPowerDistribution*

Notes

- By default, the spectral power distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

Examples

```
>>> spd = zeros_spd()
>>> spd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> spd[400]
0.0
```

`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.
`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.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*.

Return type numeric or ndarray

Notes

- The following form implementation is expressed in term of wavelength.
- The SI unit of radiance is *watts per steradian per square metre*.

References

- *[CIET14804c]*

Examples

```
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> planck_law(500 * 1e-9, 5500)
20472701909806.5...
```

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*.

Return type numeric or ndarray

Notes

- The following form implementation is expressed in term of wavelength.
- The SI unit of radiance is *watts per steradian per square metre*.

References

- *[CIET14804c]*

Examples

```
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> planck_law(500 * 1e-9, 5500)
20472701909806.5...
```

Conversion to Tristimulus Values

colour

spectral_to_XYZ(spd[, cmfs, illuminant, method])	Converts given spectral power distribution to *CIE XYZ* tristimulus values using given colour matching functions, illuminant and method.
SPECTRAL_TO_XYZ_METHODS	Supported spectral power distribution to *CIE XYZ* tristimulus values
wavelength_to_XYZ(wavelength[, cmfs])	Converts given wavelength λ to *CIE XYZ* tristimulus values using given colour matching functions.

colour.spectral_to_XYZ

colour.**spectral_to_XYZ**(*spd, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), illuminant=SpectralPowerDistribution(name='1 Constant', ...), method=u'ASTM E308-15', **kwargs*)

Converts given spectral power distribution to *CIE XYZ* tristimulus values using given colour matching functions, illuminant and method.

Parameters

- **spd** (SpectralPowerDistribution) – Spectral power distribution.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralPowerDistribution, optional) – Illuminant spectral power distribution.
- **method** (unicode, optional) – **{'ASTM E308-15', 'Integration'}**, Computation method.

Other Parameters

- **use_practice_range** (*bool, optional*) – {colour.colorimetry.spectral_to_XYZ_ASTME30815()}, 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.spectral_to_XYZ_ASTME30815()}, 5 nm measurement intervals spectral power 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.spectral_to_XYZ_ASTME30815()}, 20 nm measurement intervals spectral power distribution conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.

Returns *CIE XYZ* tristimulus values.

Return type ndarray, (3,)

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

Notes

- Output *CIE XYZ* tristimulus values are in range [0, 100].

References

- *[ASTMInternational11]*
- *[ASTMInternational15]*
- *[WS00d]*

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_RELATIVE_SPDS, SpectralPowerDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
>>> spd = SpectralPowerDistribution(data)
>>> illuminant = ILLUMINANTS_RELATIVE_SPDS['D50']
>>> spectral_to_XYZ(spd, cmfs, illuminant)
...
array([ 11.5290265...,   9.9502091...,   4.7098882...])
>>> spectral_to_XYZ(spd, cmfs, illuminant, use_practice_range=False)
...
array([ 11.5291275...,   9.9502369...,   4.7098811...])
>>> spectral_to_XYZ(spd, cmfs, illuminant, method='Integration')
...
array([ 11.5296285...,   9.9499467...,   4.7066079...])
```

colour.SPECTRAL_TO_XYZ_METHODS

colour.**SPECTRAL_TO_XYZ_METHODS = CaseInsensitiveMapping({u'ASTM E308-15': ..., u'Integration': ..., u'astm2**

Supported spectral power distribution to *CIE XYZ* tristimulus values conversion methods

References

- *[ASTMInternational11]*
- *[ASTMInternational15]*
- *[WS00d]*

SPECTRAL_TO_XYZ_METHODS [CaseInsensitiveMapping] **{'ASTM E308-15', 'Integration'}**

Aliases:

- 'astm2015': 'ASTM E308-15'

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

- Output *CIE XYZ* tristimulus values are in range [0, 1].

Examples

```
>>> from colour import CMFS
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> wavelength_to_XYZ(480, cmfs)
array([ 0.09564   ,  0.13902   ,  0.812950...])
>>> wavelength_to_XYZ(480.5, cmfs)
array([ 0.0914287...,  0.1418350...,  0.7915726...])
```

ASTM E308-15

colour.colorimetry

spectral_to_XYZ_ASTME30815(spd[, cmfs, . . .])	Converts given spectral power distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

colour.colorimetry.spectral_to_XYZ_ASTME30815

`colour.colorimetry.`**`spectral_to_XYZ_ASTME30815`**(*spd*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralPowerDistribution(name='1 Constant', ...)*, *use_practice_range=True*, *mi_5nm_omission_method=True*, *mi_20nm_interpolation_method=True*)

Converts given spectral power distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

Parameters

- **spd** (`SpectralPowerDistribution`) – Spectral power distribution.
- **cmfs** (`XYZ_ColourMatchingFunctions`) – Standard observer colour matching functions.
- **illuminant** (`SpectralPowerDistribution, optional`) – Illuminant spectral power 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 power 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 power distribution conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.

Returns *CIE XYZ* tristimulus values.

Return type ndarray, (3,)

Warning:

- The tables of tristimulus weighting factors are cached in `colour.colorimetry.tristimulus._TRISTIMULUS_WEIGHTING_FACTORS_CACHE` attribute. Their identifier key is defined by the colour matching functions and illuminant names along the current shape such as: *CIE 1964 10 Degree Standard Observer, A, (360.0, 830.0, 10.0)* Considering the above, one should be mindful that using similar colour matching functions and illuminant names but with different spectral data will lead to unexpected behaviour.
- The output range of that definition is non standard!

Notes

- Output *CIE XYZ* tristimulus values are in range [0, 100].

References

- *[ASTMInternational15]*

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_RELATIVE_SPDS, SpectralPowerDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
>>> spd = SpectralPowerDistribution(data)
>>> illuminant = ILLUMINANTS_RELATIVE_SPDS['D50']
>>> spectral_to_XYZ_ASTME30815(spd, cmfs, illuminant)
...
array([ 11.5290265...,   9.9502091...,   4.7098882...])
```

Ancillary Objects

`colour.colorimetry`

`spectral_to_XYZ_tristimulus_weighting_factors_ASTME30815`(spd)	Converts given spectral power 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_ASTME30815`(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_ASTME202211`([...])	Computes the *Lagrange Coefficients* for given interval size using practise *ASTM E2022-11* method.
`tristimulus_weighting_factors_ASTME202211`(...)	Returns a table of tristimulus weighting factors for given colour matching functions and illuminant using practise *ASTM E2022-11* method.

colour.colorimetry.spectral_to_XYZ_tristimulus_weighting_factors_ASTME30815

colour.colorimetry.**spectral_to_XYZ_tristimulus_weighting_factors_ASTME30815**(*spd, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), illuminant=SpectralPowerDistribution(name='1.0 Constant', ...)*)

Converts given spectral power 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

- **spd** (SpectralPowerDistribution) – Spectral power distribution.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralPowerDistribution, optional) – Illuminant spectral power distribution.

Returns *CIE XYZ* tristimulus values.

Return type ndarray, (3,)

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

Notes

- Output *CIE XYZ* tristimulus values are in range [0, 100].

References

- *[ASTMInternational15]*

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_RELATIVE_SPDS, SpectralPowerDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
```

```
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
>>> spd = SpectralPowerDistribution(data)
>>> illuminant = ILLUMINANTS_RELATIVE_SPDS['D50']
>>> spectral_to_XYZ_tristimulus_weighting_factors_ASTME30815(
...     spd, cmfs, illuminant)
array([ 11.5296311...,   9.9505845...,   4.7098037...])
```

colour.colorimetry.adjust_tristimulus_weighting_factors_ASTME30815

colour.colorimetry.**adjust_tristimulus_weighting_factors_ASTME30815**(*W*, *shape_r*, *shape_t*)

Adjusts given table of tristimulus weighting factors to account for a shorter wavelengths range of the test spectral shape compared to the reference spectral shape using practise *ASTM E308-15* method: Weights at the wavelengths for which data are not available are added to the weights at the shortest and longest wavelength for which spectral data are available.

Parameters

- **W** (array_like) – Tristimulus weighting factors table.
- **shape_r** (SpectralShape) – Reference spectral shape.
- **shape_t** (SpectralShape) – Test spectral shape.

Returns Adjusted tristimulus weighting factors.

Return type ndarray

References

- *[ASTMInternational15]*

Examples

```
>>> from colour import (CMFS, CIE_standard_illuminant_A_function,
...     SpectralPowerDistribution, SpectralShape)
>>> from colour.utilities import numpy_print_options
>>> cmfs = CMFS['CIE 1964 10 Degree Standard Observer']
>>> wl = cmfs.shape.range()
>>> A = SpectralPowerDistribution(
...     dict(zip(wl, CIE_standard_illuminant_A_function(wl))),
...     name='A (360, 830, 1)')
>>> W = tristimulus_weighting_factors_ASTME202211(
...     cmfs, A, SpectralShape(360, 830, 20))
>>> with numpy_print_options(suppress=True):
...     adjust_tristimulus_weighting_factors_ASTME30815(
...         W,  SpectralShape(360, 830, 20), SpectralShape(400, 700, 20))
...
array([[  0.0509543...,   0.0040971...,   0.2144280...],
```

```
       [  0.7734225...,   0.0779839...,   3.6965732...],
       [  1.9000905...,   0.3037005...,   9.7554195...],
       [  1.9707727...,   0.8552809...,  11.4867325...],
       [  0.7183623...,   2.1457000...,   6.7845806...],
       [  0.0426667...,   4.8985328...,   2.3208000...],
       [  1.5223302...,   9.6471138...,   0.7430671...],
       [  5.6770329...,  14.4609708...,   0.1958194...],
       [ 12.4451744...,  17.4742541...,   0.0051827...],
       [ 20.5535772...,  17.5838219...,  -0.0026512...],
       [ 25.3315384...,  14.8957035...,   0.        ...],
       [ 21.5711570...,  10.0796619...,   0.        ...],
       [ 12.1785817...,   5.0680655...,   0.        ...],
       [  4.6675746...,   1.8303239...,   0.        ...],
       [  1.3236117...,   0.5129694...,   0.        ...],
       [  0.4171109...,   0.1618194...,   0.        ...]])
```

colour.colorimetry.lagrange_coefficients_ASTME202211

colour.colorimetry.**lagrange_coefficients_ASTME202211**(*interval=10*, *interval_type=u'inner'*)

Computes the *Lagrange Coefficients* for given interval size using practise *ASTM E2022-11* method.

Parameters

- **interval** (int) – Interval size in nm.
- **interval_type** (unicode, optional) – **{'inner', 'boundary'}**, If the interval is an *inner* interval *Lagrange Coefficients* are computed for degree 4. Degree 3 is used for a *boundary* interval.

Returns *Lagrange Coefficients*.

Return type ndarray

References

- *[ASTMInternational11]*

Examples

```
>>> lagrange_coefficients_ASTME202211(10, 'inner')
...
array([[-0.028...,  0.940...,  0.104..., -0.016...],
       [-0.048...,  0.864...,  0.216..., -0.032...],
       [-0.059...,  0.773...,  0.331..., -0.045...],
       [-0.064...,  0.672...,  0.448..., -0.056...],
       [-0.062...,  0.562...,  0.562..., -0.062...],
       [-0.056...,  0.448...,  0.672..., -0.064...],
       [-0.045...,  0.331...,  0.773..., -0.059...],
       [-0.032...,  0.216...,  0.864..., -0.048...],
       [-0.016...,  0.104...,  0.940..., -0.028...]])
>>> lagrange_coefficients_ASTME202211(10, 'boundary')
...
array([[ 0.85...,  0.19..., -0.04...],
       [ 0.72...,  0.36..., -0.08...],
```

```
       [ 0.59...,  0.51..., -0.10...],
       [ 0.48...,  0.64..., -0.12...],
       [ 0.37...,  0.75..., -0.12...],
       [ 0.28...,  0.84..., -0.12...],
       [ 0.19...,  0.91..., -0.10...],
       [ 0.12...,  0.96..., -0.08...],
       [ 0.05...,  0.99..., -0.04...]])
```

colour.colorimetry.tristimulus_weighting_factors_ASTME202211

colour.colorimetry.**tristimulus_weighting_factors_ASTME202211**(*cmfs*, *illuminant*, *shape*)

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** (SpectralPowerDistribution) – Illuminant spectral power distribution.
- **shape** (SpectralShape) – Shape used to build the table, only the interval is needed.

Returns Tristimulus weighting factors table.

Return type ndarray

Raises ValueError – If the colour matching functions or illuminant intervals are not equal to 1 nm.

Warning:

- The tables of tristimulus weighting factors are cached in colour.colorimetry.tristimulus._TRISTIMULUS_WEIGHTING_FACTORS_CACHE attribute. Their identifier key is defined by the colour matching functions and illuminant names along the current shape such as: *CIE 1964 10 Degree Standard Observer, A, (360.0, 830.0, 10.0)* Considering the above, one should be mindful that using similar colour matching functions and illuminant names but with different spectral data will lead to unexpected behaviour.

Notes

- Input colour matching functions and illuminant intervals are expected to be equal to 1 nm. If the illuminant data is not available at 1 nm interval, it needs to be interpolated using *CIE* recommendations: The method developed by *Sprague (1880)* should be used for interpolating functions having a uniformly spaced independent variable and a *Cubic Spline* method for non-uniformly spaced independent variable.

References

- *[ASTMInternational11]*

Examples

```
>>> from colour import (CMFS, CIE_standard_illuminant_A_function,
...     SpectralPowerDistribution, SpectralShape, numpy_print_options)
>>> cmfs = CMFS['CIE 1964 10 Degree Standard Observer']
>>> wl = cmfs.shape.range()
>>> A = SpectralPowerDistribution(
...     dict(zip(wl, CIE_standard_illuminant_A_function(wl))),
...     name='A (360, 830, 1)')
>>> with numpy_print_options(suppress=True):
...     tristimulus_weighting_factors_ASTME202211(
...         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`

`spectral_to_XYZ_integration`(spd[, cmfs, …])	Converts given spectral power distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to classical integration method.

colour.colorimetry.spectral_to_XYZ_integration

colour.colorimetry.**spectral_to_XYZ_integration**(*spd, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), illuminant=SpectralPowerDistribution(name='1 Constant', ...)*)

Converts given spectral power distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to classical integration method.

Parameters

- **spd** (SpectralPowerDistribution) – Spectral power distribution.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralPowerDistribution, optional) – Illuminant spectral power distribution.

Returns *CIE XYZ* tristimulus values.

Return type ndarray, (3,)

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

Notes

- Output *CIE XYZ* tristimulus values are in range [0, 100].

References

- *[WS00d]*

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_RELATIVE_SPDS, SpectralPowerDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
```

```
...     680: 0.2397,
...     700: 0.2852
... }
>>> spd = SpectralPowerDistribution(data)
>>> illuminant = ILLUMINANTS_RELATIVE_SPDS['D50']
>>> spectral_to_XYZ_integration(spd, cmfs, illuminant)
...
array([ 11.5296285...,   9.9499467...,   4.7066079...])
```

Spectral Bandpass Dependence Correction

`colour`

`bandpass_correction`(spd[, method])	Implements spectral bandpass dependence correction on given spectral power distribution using given method.
`BANDPASS_CORRECTION_METHODS`	Supported spectral bandpass dependence correction methods.

colour.bandpass_correction

`colour.`**`bandpass_correction`**(*spd, method=u'Stearns 1988'*)

Implements spectral bandpass dependence correction on given spectral power distribution using given method.

Parameters

- **spd** (`SpectralPowerDistribution`) – Spectral power distribution.
- **method** (`unicode, optional`) – ('Stearns 1988',) Correction method.

Returns Spectral bandpass dependence corrected spectral power distribution.

Return type *SpectralPowerDistribution*

colour.BANDPASS_CORRECTION_METHODS

`colour.`**`BANDPASS_CORRECTION_METHODS = CaseInsensitiveMapping({u'Stearns 1988': ...})`**

Supported spectral bandpass dependence correction methods.

BANDPASS_CORRECTION_METHODS [CaseInsensitiveMapping] **{'Stearns 1988', }**

Stearns and Stearns (1988)

`colour.colorimetry`

`bandpass_correction_Stearns1988`(spd)	Implements spectral bandpass dependence correction on given spectral power distribution using *Stearns and Stearns (1988)* method.

colour.colorimetry.bandpass_correction_Stearns1988

colour.colorimetry.**bandpass_correction_Stearns1988**(*spd*)

Implements spectral bandpass dependence correction on given spectral power distribution using *Stearns and Stearns (1988)* method.

Parameters **spd** (SpectralPowerDistribution) – Spectral power distribution.

Returns Spectral bandpass dependence corrected spectral power distribution.

Return type *SpectralPowerDistribution*

References

- *[SS88]*
- *[WRC12c]*

Examples

```
>>> from colour import SpectralPowerDistribution
>>> 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(
...         SpectralPowerDistribution(data))
...
SpectralPowerDistribution([[ 500.        ,    0.0646518...],
                           [ 520.        ,    0.0704293...],
                           [ 540.        ,    0.0769485...],
                           [ 560.        ,    0.0856928...],
                           [ 580.        ,    0.1129644...],
                           [ 600.        ,    0.1379256...]],
                          interpolator=SpragueInterpolator,
                          interpolator_args={},
                          extrapolator=Extrapolator,
                          extrapolator_args={...})
```

Colour Matching Functions

colour.colorimetry

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.

Continued on next page

Table 3.67 – continued from previous page

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*)

Implements support for the Stockman and Sharpe *LMS* cone fundamentals colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignal or MultiSpectralPowerDistribution or array_like or dict_like, optional) – Data to be stored in the multi-spectral power distribution.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralPowerDistribution 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.SpectralPowerDistribution class instances.

Other Parameters

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

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

Methods

__init__([data, domain, labels])	
align(shape[, interpolator, ...])	Aligns the multi-spectral power distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
arithmetical_operation(a, operation[, in_place])	Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.

Continued on next page

Table 3.68 – continued from previous page

clone()	
copy()	Returns a copy of the sub-class instance, must be reimplemented by sub-classes.
domain_distance(a)	Returns the euclidean distance between given array and independent domain x closest element.
extrapolate(shape[, extrapolator, ...])	Extrapolates the multi-spectral power distribution in-place accordingly to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.
fill_nan([method, default])	Fill NaNs in independent domain x variable and corresponding range y variable using given method.
get()	
interpolate(shape[, interpolator, ...])	Interpolates the multi-spectral power distribution in-place accordingly to *CIE 167:2005* recommendation or given interpolation arguments.
is_uniform()	Returns if independent domain x variable is uniform.
multi_signal_unpack_data([data, domain, ...])	Unpack given data for multi-continuous signal instantiation.
normalise([factor])	Normalises the multi-spectral power distribution with given normalization factor.
to_dataframe()	Converts the continuous signal to a *Pandas* DataFrame class instance.
trim(shape)	Trims the multi-spectral power distribution wavelengths to given shape.
trim_wavelengths(shape)	
zeros()	

colour.colorimetry.RGB_ColourMatchingFunctions

class colour.colorimetry.**RGB_ColourMatchingFunctions**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Implements support for the *CIE RGB* colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignal or MultiSpectralPowerDistribution or array_like or dict_like, optional) – Data to be stored in the multi-spectral power distribution.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralPowerDistribution 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.SpectralPowerDistribution class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral power distribution name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.SpectralPowerDistribution class instances.

- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the `colour.SpectralPowerDistribution` class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the `colour.SpectralPowerDistribution` class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the `colour.SpectralPowerDistribution` class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral power distribution labels for figures, default to `colour.colorimetry.RGB_ColourMatchingFunctions.labels` attribute value.

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

Methods

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

colour.colorimetry.XYZ_ColourMatchingFunctions

class colour.colorimetry.**XYZ_ColourMatchingFunctions**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Implements support for the *CIE* Standard Observers *XYZ* colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignal or MultiSpectralPowerDistribution or array_like or dict_like, optional) – Data to be stored in the multi-spectral power distribution.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralPowerDistribution 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.SpectralPowerDistribution class instances.

Other Parameters

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

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

Methods

__init__([data, domain, labels])	
align(shape[, interpolator, ...])	Aligns the multi-spectral power distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
arithmetical_operation(a, operation[, in_place])	Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.
clone()	
copy()	Returns a copy of the sub-class instance, must be reimplemented by sub-classes.
domain_distance(a)	Returns the euclidean distance between given array and independent domain x closest element.

Continued on next page

Table 3.70 – continued from previous page

extrapolate(shape[, extrapolator, …])	Extrapolates the multi-spectral power distribution in-place accordingly to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.
fill_nan([method, default])	Fill NaNs in independent domain x variable and corresponding range y variable using given method.
get()	
interpolate(shape[, interpolator, …])	Interpolates the multi-spectral power distribution in-place accordingly to *CIE 167:2005* recommendation or given interpolation arguments.
is_uniform()	Returns if independent domain x variable is uniform.
multi_signal_unpack_data([data, domain, …])	Unpack given data for multi-continuous signal instantiation.
normalise([factor])	Normalises the multi-spectral power distribution with given normalization factor.
to_dataframe()	Converts the continuous signal to a *Pandas* DataFrame class instance.
trim(shape)	Trims the multi-spectral power distribution wavelengths to given shape.
trim_wavelengths(shape)	
zeros()	

Dataset

colour

CMFS	Aggregated colour matching functions.
LMS_CMFS	*LMS* colour matching functions.
RGB_CMFS	*CIE RGB* colour matching functions.
STANDARD_OBSERVERS_CMFS	*CIE* Standard Observers *XYZ* colour matching functions.

colour.CMFS

colour.**CMFS = CaseInsensitiveMapping({u'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., u'Stiles & Burch**

Aggregated colour matching functions.

References

- *[Bro09]*
- *[CVRe]*
- *[CVRf]*
- *[CVRg]*
- *[CVRc]*
- *[CVRh]*
- *[Mac10]*

CMFS [CaseInsensitiveMapping] **{'Stockman & Sharpe 10 Degree Cone Fundamentals', 'Stockman & Sharpe 2 Degree Cone Fundamentals', 'Wright & Guild 1931 2 Degree RGB CMFs', 'Stiles & Burch 1955 2 Degree RGB CMFs', 'Stiles & Burch 1959 10 Degree RGB CMFs', 'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer', 'CIE 2012 2 Degree Standard Observer', 'CIE 2012 10 Degree Standard Observer'}**

colour.LMS_CMFS

```
colour.LMS_CMFS = CaseInsensitiveMapping({u'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., u'Smith & Po
```

LMS colour matching functions.

References

- *[CVRc]*
- *[Mac10]*

LMS_CMFS [CaseInsensitiveMapping] {'Stockman & Sharpe 2 Degree Cone Fundamentals', 'Stockman & Sharpe 10 Degree Cone Fundamentals', 'Smith & Pokorny 1975 Normal Trichromats'}

colour.RGB_CMFS

```
colour.RGB_CMFS = CaseInsensitiveMapping({u'Stiles & Burch 1959 10 Degree RGB CMFs': ..., u'Wright & Guild 1
```

CIE RGB colour matching functions.

References

- *[Bro09]*
- *[CVRg]*
- *[CVRh]*

RGB_CMFS [CaseInsensitiveMapping] **{'Wright & Guild 1931 2 Degree RGB CMFs', 'Stiles & Burch 1955 2 Degree RGB CMFs', 'Stiles & Burch 1959 10 Degree RGB CMFs'}**

colour.STANDARD_OBSERVERS_CMFS

```
colour.STANDARD_OBSERVERS_CMFS = CaseInsensitiveMapping({u'CIE 1931 2 Degree Standard Observer': ..., u'cie_
```

CIE Standard Observers *XYZ* colour matching functions.

References

- *[CVRe]*
- *[CVRf]*

STANDARD_OBSERVERS_CMFS [CaseInsensitiveMapping] **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer', 'CIE 2012 2 Degree Standard Observer', 'CIE 2012 10 Degree Standard Observer'}**

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

Colour Matching Functions Transformations

Ancillary Objects

`colour.colorimetry`

`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

- *[WS00g]*

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

- *[WS00k]*

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

- *[CIET13606]*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_10_degree_cmfs_to_LMS_10_degree_cmfs(700)
array([ 0.0052860...,  0.0003252...,  0.        ])
```

colour.colorimetry.LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs

colour.colorimetry.**LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs**(*wavelength*)
Converts *Stockman & Sharpe 2 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 2 Degree Standard Observer* colour matching functions.

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

Returns *CIE 2012 2 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 2012 2 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

- *[CVRb]*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs(700)
array([ 0.0109677...,  0.0041959...,  0.        ])
```

colour.colorimetry.LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs

colour.colorimetry.**LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs**(*wavelength*)

Converts *Stockman & Sharpe 10 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 10 Degree Standard Observer* colour matching functions.

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

Returns *CIE 2012 10 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 2012 10 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

- *[CVRa]*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs(700)
array([ 0.0098162...,  0.0037761...,  0.        ])
```

Illuminants and Light Sources

Dataset

colour

ILLUMINANTS	Aggregated *CIE* illuminants chromaticity coordinates.
ILLUMINANTS_RELATIVE_SPDS	*CIE* illuminants relative spectral power distributions.
HUNTERLAB_ILLUMINANTS	Aggregated *Hunter L,a,b* illuminant dataset.
LIGHT_SOURCES	Aggregated light sources chromaticity coordinates.
LIGHT_SOURCES_RELATIVE_SPDS	Aggregated light sources spectral power distributions.

colour.ILLUMINANTS

colour.**ILLUMINANTS = CaseInsensitiveMapping({u'CIE 1931 2 Degree Standard Observer': ..., u'CIE 1964 10 Degr**

Aggregated *CIE* illuminants chromaticity coordinates.

References

- *[Wiky]*

- *[DigitalCInitiatives07]*

ILLUMINANTS [CaseInsensitiveMapping] **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}**

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.ILLUMINANTS_RELATIVE_SPDS

colour.**ILLUMINANTS_RELATIVE_SPDS = CaseInsensitiveMapping({u'FL3.8': ..., u'FL3.9': ..., u'D75': ..., u'FL3**

CIE illuminants relative spectral power distributions.

References

- *[CIE04]*
- *[CIE]*

ILLUMINANTS_RELATIVE_SPDS : CaseInsensitiveMapping

colour.HUNTERLAB_ILLUMINANTS

colour.**HUNTERLAB_ILLUMINANTS = CaseInsensitiveMapping({u'CIE 1931 2 Degree Standard Observer': ..., u'CIE 19**

Aggregated *Hunter L,a,b* illuminant dataset.

References

- *[Hun08a]*
- *[Hun08b]*

HUNTERLAB_ILLUMINANTS [CaseInsensitiveMapping] **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}**

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.LIGHT_SOURCES

colour.**LIGHT_SOURCES = CaseInsensitiveMapping({u'CIE 1931 2 Degree Standard Observer': ..., u'CIE 1964 10 De**

Aggregated light sources chromaticity coordinates.

LIGHT_SOURCES [CaseInsensitiveMapping] **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}**

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.LIGHT_SOURCES_RELATIVE_SPDS

colour.**LIGHT_SOURCES_RELATIVE_SPDS = CaseInsensitiveMapping({u'Mercury': ..., u'F34T12WW/RS /EW (Warm White**

Aggregated light sources spectral power distributions.

LIGHT_SOURCES_RELATIVE_SPDS : CaseInsensitiveMapping

Dominant Wavelength and Purity

colour

dominant_wavelength(xy, xy_n[, cmfs, reverse])	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', ...), reverse=False*)

Returns the *dominant wavelength* λ_d for given colour stimulus xy and the related xy_wl first and xy_{cw} second intersection coordinates with the spectral locus.

In the eventuality where the xy_wl first intersection coordinates are on the line of purples, the *complementary wavelength* will be computed in lieu.

The *complementary wavelength* is indicated by a negative sign and the xy_{cw} second intersection coordinates which are set by default to the same value than xy_wl first intersection coordinates will be set to the *complementary dominant wavelength* intersection coordinates with the spectral locus.

Parameters

- **xy** (array_like) – Colour stimulus *xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **reverse** (bool, optional) – Reverse the computation direction to retrieve the *complementary wavelength*.

Returns *Dominant wavelength*, first intersection point *xy* chromaticity coordinates, second intersection point *xy* chromaticity coordinates.

Return type tuple

References

- *[CIET14804b]*
- *[Erdb]*

Examples

Dominant wavelength computation:

```
>>> from pprint import pprint
>>> xy = np.array([0.26415, 0.37770])
>>> xy_n = np.array([0.31270, 0.32900])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> pprint(dominant_wavelength(xy, xy_n, cmfs))
(array(504...),
 array([ 0.0036969...,  0.6389577...]),
 array([ 0.0036969...,  0.6389577...]))
```

Complementary dominant wavelength is returned if the first intersection is located on the line of purples:

```
>>> xy = np.array([0.35000, 0.25000])
>>> pprint(dominant_wavelength(xy, xy_n, cmfs))
(array(-520...),
 array([ 0.4133314...,  0.1158663...]),
 array([ 0.0743553...,  0.8338050...]))
```

colour.complementary_wavelength

`colour.`**`complementary_wavelength`**(*xy, xy_n, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*)

Returns the *complementary wavelength* λ_c for given colour stimulus xy and the related xy_wl first and xy_{cw} second intersection coordinates with the spectral locus.

In the eventuality where the xy_wl first intersection coordinates are on the line of purples, the *dominant wavelength* will be computed in lieu.

The *dominant wavelength* is indicated by a negative sign and the xy_{cw} second intersection coordinates which are set by default to the same value than xy_wl first intersection coordinates will be set to the *dominant wavelength* intersection coordinates with the spectral locus.

Parameters

- **xy** (`array_like`) – Colour stimulus *xy* chromaticity coordinates.
- **xy_n** (`array_like`) – Achromatic stimulus *xy* chromaticity coordinates.
- **cmfs** (`XYZ_ColourMatchingFunctions, optional`) – Standard observer colour matching functions.

Returns *Complementary wavelength*, first intersection point *xy* chromaticity coordinates, second intersection point *xy* chromaticity coordinates.

Return type tuple

References

- *[CIET14804b]*
- *[Erdb]*

Examples

Complementary wavelength computation:

```
>>> from pprint import pprint
>>> xy = np.array([0.35000, 0.25000])
>>> xy_n = np.array([0.31270, 0.32900])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> pprint(complementary_wavelength(xy, xy_n, cmfs))
(array(520...),
 array([ 0.0743553...,  0.8338050...]),
 array([ 0.0743553...,  0.8338050...]))
```

Dominant wavelength is returned if the first intersection is located on the line of purples:

```
>>> xy = np.array([0.26415, 0.37770])
>>> pprint(complementary_wavelength(xy, xy_n, cmfs))
(array(-504...),
 array([ 0.4897494...,  0.1514035...]),
 array([ 0.0036969...,  0.6389577...]))
```

colour.excitation_purity

`colour.`**`excitation_purity`**(*xy, xy_n, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*)

Returns the *excitation purity* P_e for given colour stimulus xy.

Parameters

- **xy** (array_like) – Colour stimulus *xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *Excitation purity* P_e.

Return type numeric or array_like

References

- *[CIET14804b]*
- *[Erdb]*

Examples

```
>>> xy = np.array([0.28350, 0.68700])
>>> xy_n = np.array([0.31270, 0.32900])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> excitation_purity(xy, xy_n, cmfs)
0.9386035...
```

colour.colorimetric_purity

colour.**colorimetric_purity**(*xy*, *xy_n*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*)

Returns the *colorimetric purity* P_c for given colour stimulus xy.

Parameters

- **xy** (array_like) – Colour stimulus *xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *Colorimetric purity* P_c.

Return type numeric or array_like

References

- *[CIET14804b]*
- *[Erdb]*

Examples

```
>>> xy = np.array([0.28350, 0.68700])
>>> xy_n = np.array([0.31270, 0.32900])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> colorimetric_purity(xy, xy_n, cmfs)
0.9705976...
```

Luminous Efficiency Functions

colour

luminous_efficacy(spd[, lef])	Returns the *luminous efficacy* in $lm \cdot W^{-1}$ of given spectral power distribution using given luminous efficiency function.
luminous_efficiency(spd[, lef])	Returns the *luminous efficiency* of given spectral power distribution using given luminous efficiency function.

Continued on next page

Table 3.75 – continued from previous page

luminous_flux(spd[, lef, K_m])	Returns the *luminous flux* for given spectral power distribution using given luminous efficiency function.
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**(*spd, lef=SpectralPowerDistribution(name='CIE 1924 Photopic Standard Observer', ...)*)

Returns the *luminous efficacy* in $lm \cdot W^{-1}$ of given spectral power distribution using given luminous efficiency function.

Parameters

- **spd** (SpectralPowerDistribution) – test spectral power distribution
- **lef** (SpectralPowerDistribution, optional) – $V(\lambda)$ luminous efficiency function.

Returns Luminous efficacy in $lm \cdot W^{-1}$.

Return type numeric

References

- *[Wikr]*

Examples

```
>>> from colour import LIGHT_SOURCES_RELATIVE_SPDS
>>> spd = LIGHT_SOURCES_RELATIVE_SPDS['Neodimium Incandescent']
>>> luminous_efficacy(spd)
136.2170803...
```

colour.luminous_efficiency

colour.**luminous_efficiency**(*spd, lef=SpectralPowerDistribution(name='CIE 1924 Photopic Standard Observer', ...)*)

Returns the *luminous efficiency* of given spectral power distribution using given luminous efficiency function.

Parameters

- **spd** (SpectralPowerDistribution) – test spectral power distribution
- **lef** (SpectralPowerDistribution, optional) – $V(\lambda)$ luminous efficiency function.

Returns Luminous efficiency.

Return type numeric

References

- *[Wikq]*

Examples

```
>>> from colour import LIGHT_SOURCES_RELATIVE_SPDS
>>> spd = LIGHT_SOURCES_RELATIVE_SPDS['Neodimium Incandescent']
>>> luminous_efficiency(spd)
0.1994393...
```

colour.luminous_flux

colour.**luminous_flux**(*spd*, *lef=SpectralPowerDistribution(name='CIE 1924 Photopic Standard Observer', ...)*, *K_m=683.0*)

Returns the *luminous flux* for given spectral power distribution using given luminous efficiency function.

Parameters

- **spd** (SpectralPowerDistribution) – test spectral power distribution
- **lef** (SpectralPowerDistribution, 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

- *[Wikq]*

Examples

```
>>> from colour import LIGHT_SOURCES_RELATIVE_SPDS
>>> spd = LIGHT_SOURCES_RELATIVE_SPDS['Neodimium Incandescent']
>>> luminous_flux(spd)
23807.6555273...
```

colour.mesopic_luminous_efficiency_function

colour.**mesopic_luminous_efficiency_function**(*Lp*, *source=u'Blue Heavy'*, *method=u'MOVE'*, *photopic_lef=SpectralPowerDistribution(name='CIE 1924 Photopic Standard Observer', ...)*, *scotopic_lef=SpectralPowerDistribution(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** (SpectralPowerDistribution, optional) – $V(\lambda)$ photopic luminous efficiency function.
- **scotopic_lef** (SpectralPowerDistribution, optional) – $V'(\lambda)$ scotopic luminous efficiency function.

Returns Mesopic luminous efficiency function $V_m(\lambda)$.

Return type *SpectralPowerDistribution*

References

- *[Wiks]*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     mesopic_luminous_efficiency_function(0.2)
SpectralPowerDistribution([[ 380.        ,    0.000424 ...],
                           [ 381.        ,    0.0004781...],
                           [ 382.        ,    0.0005399...],
                           [ 383.        ,    0.0006122...],
                           [ 384.        ,    0.0006961...],
                           [ 385.        ,    0.0007929...],
                           [ 386.        ,    0.000907 ...],
                           [ 387.        ,    0.0010389...],
                           [ 388.        ,    0.0011923...],
                           [ 389.        ,    0.0013703...],
                           [ 390.        ,    0.0015771...],
                           [ 391.        ,    0.0018167...],
                           [ 392.        ,    0.0020942...],
                           [ 393.        ,    0.0024160...],
                           [ 394.        ,    0.0027888...],
                           [ 395.        ,    0.0032196...],
                           [ 396.        ,    0.0037222...],
                           [ 397.        ,    0.0042957...],
                           [ 398.        ,    0.0049531...],
                           [ 399.        ,    0.0057143...],
                           [ 400.        ,    0.0065784...],
                           [ 401.        ,    0.0075658...],
                           [ 402.        ,    0.0086912...],
                           [ 403.        ,    0.0099638...],
                           [ 404.        ,    0.0114058...],
                           [ 405.        ,    0.0130401...],
                           [ 406.        ,    0.0148750...],
                           [ 407.        ,    0.0169310...],
                           [ 408.        ,    0.0192211...],
                           [ 409.        ,    0.0217511...],
                           [ 410.        ,    0.0245342...],
                           [ 411.        ,    0.0275773...],
                           [ 412.        ,    0.0309172...],
                           [ 413.        ,    0.0345149...],
                           [ 414.        ,    0.0383998...],
```

```
[ 415.        ,    0.0425744...],
[ 416.        ,    0.0471074...],
[ 417.        ,    0.0519322...],
[ 418.        ,    0.0570541...],
[ 419.        ,    0.0625466...],
[ 420.        ,    0.0683463...],
[ 421.        ,    0.0745255...],
[ 422.        ,    0.0809440...],
[ 423.        ,    0.0877344...],
[ 424.        ,    0.0948915...],
[ 425.        ,    0.1022731...],
[ 426.        ,    0.109877 ...],
[ 427.        ,    0.1178421...],
[ 428.        ,    0.1260316...],
[ 429.        ,    0.1343772...],
[ 430.        ,    0.143017 ...],
[ 431.        ,    0.1518128...],
[ 432.        ,    0.1608328...],
[ 433.        ,    0.1700088...],
[ 434.        ,    0.1792726...],
[ 435.        ,    0.1886934...],
[ 436.        ,    0.1982041...],
[ 437.        ,    0.2078032...],
[ 438.        ,    0.2174184...],
[ 439.        ,    0.2271147...],
[ 440.        ,    0.2368196...],
[ 441.        ,    0.2464623...],
[ 442.        ,    0.2561153...],
[ 443.        ,    0.2657160...],
[ 444.        ,    0.2753387...],
[ 445.        ,    0.2848520...],
[ 446.        ,    0.2944648...],
[ 447.        ,    0.3034902...],
[ 448.        ,    0.3132347...],
[ 449.        ,    0.3223257...],
[ 450.        ,    0.3314513...],
[ 451.        ,    0.3406129...],
[ 452.        ,    0.3498117...],
[ 453.        ,    0.3583617...],
[ 454.        ,    0.3676377...],
[ 455.        ,    0.3762670...],
[ 456.        ,    0.3849392...],
[ 457.        ,    0.3936540...],
[ 458.        ,    0.4024077...],
[ 459.        ,    0.4111965...],
[ 460.        ,    0.4193298...],
[ 461.        ,    0.4281803...],
[ 462.        ,    0.4363804...],
[ 463.        ,    0.4453117...],
[ 464.        ,    0.4542949...],
[ 465.        ,    0.4626509...],
[ 466.        ,    0.4717570...],
[ 467.        ,    0.4809300...],
[ 468.        ,    0.4901776...],
[ 469.        ,    0.4995075...],
[ 470.        ,    0.5096145...],
[ 471.        ,    0.5191293...],
[ 472.        ,    0.5294259...],
```

```
                  [ 473.        ,   0.5391316...],
                  [ 474.        ,   0.5496217...],
                  [ 475.        ,   0.5602103...],
                  [ 476.        ,   0.5702197...],
                  [ 477.        ,   0.5810207...],
                  [ 478.        ,   0.5919093...],
                  [ 479.        ,   0.6028683...],
                  [ 480.        ,   0.6138806...],
                  [ 481.        ,   0.6249373...],
                  [ 482.        ,   0.6360619...],
                  [ 483.        ,   0.6465989...],
                  [ 484.        ,   0.6579538...],
                  [ 485.        ,   0.6687841...],
                  [ 486.        ,   0.6797939...],
                  [ 487.        ,   0.6909887...],
                  [ 488.        ,   0.7023827...],
                  [ 489.        ,   0.7133032...],
                  [ 490.        ,   0.7244513...],
                  [ 491.        ,   0.7358470...],
                  [ 492.        ,   0.7468118...],
                  [ 493.        ,   0.7580294...],
                  [ 494.        ,   0.7694964...],
                  [ 495.        ,   0.7805225...],
                  [ 496.        ,   0.7917805...],
                  [ 497.        ,   0.8026123...],
                  [ 498.        ,   0.8130793...],
                  [ 499.        ,   0.8239297...],
                  [ 500.        ,   0.8352251...],
                  [ 501.        ,   0.8456342...],
                  [ 502.        ,   0.8564818...],
                  [ 503.        ,   0.8676921...],
                  [ 504.        ,   0.8785021...],
                  [ 505.        ,   0.8881489...],
                  [ 506.        ,   0.8986405...],
                  [ 507.        ,   0.9079322...],
                  [ 508.        ,   0.9174255...],
                  [ 509.        ,   0.9257739...],
                  [ 510.        ,   0.9350656...],
                  [ 511.        ,   0.9432365...],
                  [ 512.        ,   0.9509063...],
                  [ 513.        ,   0.9586931...],
                  [ 514.        ,   0.9658413...],
                  [ 515.        ,   0.9722825...],
                  [ 516.        ,   0.9779924...],
                  [ 517.        ,   0.9836106...],
                  [ 518.        ,   0.9883465...],
                  [ 519.        ,   0.9920964...],
                  [ 520.        ,   0.9954436...],
                  [ 521.        ,   0.9976202...],
                  [ 522.        ,   0.9993457...],
                  [ 523.        ,   1.       ...],
                  [ 524.        ,   0.9996498...],
                  [ 525.        ,   0.9990487...],
                  [ 526.        ,   0.9975356...],
                  [ 527.        ,   0.9957615...],
                  [ 528.        ,   0.9930143...],
                  [ 529.        ,   0.9899559...],
                  [ 530.        ,   0.9858741...],
```

```
[ 531.        ,    0.9814453...],
[ 532.        ,    0.9766885...],
[ 533.        ,    0.9709363...],
[ 534.        ,    0.9648947...],
[ 535.        ,    0.9585832...],
[ 536.        ,    0.952012 ...],
[ 537.        ,    0.9444916...],
[ 538.        ,    0.9367089...],
[ 539.        ,    0.9293506...],
[ 540.        ,    0.9210429...],
[ 541.        ,    0.9124772...],
[ 542.        ,    0.9036604...],
[ 543.        ,    0.8945958...],
[ 544.        ,    0.8845999...],
[ 545.        ,    0.8750500...],
[ 546.        ,    0.8659457...],
[ 547.        ,    0.8559224...],
[ 548.        ,    0.8456846...],
[ 549.        ,    0.8352499...],
[ 550.        ,    0.8253229...],
[ 551.        ,    0.8152079...],
[ 552.        ,    0.8042205...],
[ 553.        ,    0.7944209...],
[ 554.        ,    0.7837466...],
[ 555.        ,    0.7735680...],
[ 556.        ,    0.7627808...],
[ 557.        ,    0.7522710...],
[ 558.        ,    0.7417549...],
[ 559.        ,    0.7312909...],
[ 560.        ,    0.7207983...],
[ 561.        ,    0.7101939...],
[ 562.        ,    0.6996362...],
[ 563.        ,    0.6890656...],
[ 564.        ,    0.6785599...],
[ 565.        ,    0.6680593...],
[ 566.        ,    0.6575697...],
[ 567.        ,    0.6471578...],
[ 568.        ,    0.6368208...],
[ 569.        ,    0.6264871...],
[ 570.        ,    0.6161541...],
[ 571.        ,    0.6058896...],
[ 572.        ,    0.5957000...],
[ 573.        ,    0.5855937...],
[ 574.        ,    0.5754412...],
[ 575.        ,    0.5653883...],
[ 576.        ,    0.5553742...],
[ 577.        ,    0.5454680...],
[ 578.        ,    0.5355972...],
[ 579.        ,    0.5258267...],
[ 580.        ,    0.5160152...],
[ 581.        ,    0.5062322...],
[ 582.        ,    0.4965595...],
[ 583.        ,    0.4868746...],
[ 584.        ,    0.4773299...],
[ 585.        ,    0.4678028...],
[ 586.        ,    0.4583704...],
[ 587.        ,    0.4489722...],
[ 588.        ,    0.4397606...],
```

```
                  [ 589.        ,    0.4306131...],
                  [ 590.        ,    0.4215446...],
                  [ 591.        ,    0.4125681...],
                  [ 592.        ,    0.4037550...],
                  [ 593.        ,    0.3950359...],
                  [ 594.        ,    0.3864104...],
                  [ 595.        ,    0.3778777...],
                  [ 596.        ,    0.3694405...],
                  [ 597.        ,    0.3611074...],
                  [ 598.        ,    0.3528596...],
                  [ 599.        ,    0.3447056...],
                  [ 600.        ,    0.3366470...],
                  [ 601.        ,    0.3286917...],
                  [ 602.        ,    0.3208410...],
                  [ 603.        ,    0.3130808...],
                  [ 604.        ,    0.3054105...],
                  [ 605.        ,    0.2978225...],
                  [ 606.        ,    0.2903027...],
                  [ 607.        ,    0.2828727...],
                  [ 608.        ,    0.2755311...],
                  [ 609.        ,    0.2682900...],
                  [ 610.        ,    0.2611478...],
                  [ 611.        ,    0.2541176...],
                  [ 612.        ,    0.2471885...],
                  [ 613.        ,    0.2403570...],
                  [ 614.        ,    0.2336057...],
                  [ 615.        ,    0.2269379...],
                  [ 616.        ,    0.2203527...],
                  [ 617.        ,    0.2138465...],
                  [ 618.        ,    0.2073946...],
                  [ 619.        ,    0.2009789...],
                  [ 620.        ,    0.1945818...],
                  [ 621.        ,    0.1881943...],
                  [ 622.        ,    0.1818226...],
                  [ 623.        ,    0.1754987...],
                  [ 624.        ,    0.1692476...],
                  [ 625.        ,    0.1630876...],
                  [ 626.        ,    0.1570257...],
                  [ 627.        ,    0.151071 ...],
                  [ 628.        ,    0.1452469...],
                  [ 629.        ,    0.1395845...],
                  [ 630.        ,    0.1341087...],
                  [ 631.        ,    0.1288408...],
                  [ 632.        ,    0.1237666...],
                  [ 633.        ,    0.1188631...],
                  [ 634.        ,    0.1141075...],
                  [ 635.        ,    0.1094766...],
                  [ 636.        ,    0.1049613...],
                  [ 637.        ,    0.1005679...],
                  [ 638.        ,    0.0962924...],
                  [ 639.        ,    0.0921296...],
                  [ 640.        ,    0.0880778...],
                  [ 641.        ,    0.0841306...],
                  [ 642.        ,    0.0802887...],
                  [ 643.        ,    0.0765559...],
                  [ 644.        ,    0.0729367...],
                  [ 645.        ,    0.0694345...],
                  [ 646.        ,    0.0660491...],
```

```
[ 647.        ,    0.0627792...],
[ 648.        ,    0.0596278...],
[ 649.        ,    0.0565970...],
[ 650.        ,    0.0536896...],
[ 651.        ,    0.0509068...],
[ 652.        ,    0.0482444...],
[ 653.        ,    0.0456951...],
[ 654.        ,    0.0432510...],
[ 655.        ,    0.0409052...],
[ 656.        ,    0.0386537...],
[ 657.        ,    0.0364955...],
[ 658.        ,    0.0344285...],
[ 659.        ,    0.0324501...],
[ 660.        ,    0.0305579...],
[ 661.        ,    0.0287496...],
[ 662.        ,    0.0270233...],
[ 663.        ,    0.0253776...],
[ 664.        ,    0.0238113...],
[ 665.        ,    0.0223226...],
[ 666.        ,    0.0209086...],
[ 667.        ,    0.0195688...],
[ 668.        ,    0.0183056...],
[ 669.        ,    0.0171216...],
[ 670.        ,    0.0160192...],
[ 671.        ,    0.0149986...],
[ 672.        ,    0.0140537...],
[ 673.        ,    0.0131784...],
[ 674.        ,    0.0123662...],
[ 675.        ,    0.0116107...],
[ 676.        ,    0.0109098...],
[ 677.        ,    0.0102587...],
[ 678.        ,    0.0096476...],
[ 679.        ,    0.0090665...],
[ 680.        ,    0.0085053...],
[ 681.        ,    0.0079567...],
[ 682.        ,    0.0074229...],
[ 683.        ,    0.0069094...],
[ 684.        ,    0.0064213...],
[ 685.        ,    0.0059637...],
[ 686.        ,    0.0055377...],
[ 687.        ,    0.0051402...],
[ 688.        ,    0.00477  ...],
[ 689.        ,    0.0044263...],
[ 690.        ,    0.0041081...],
[ 691.        ,    0.0038149...],
[ 692.        ,    0.0035456...],
[ 693.        ,    0.0032984...],
[ 694.        ,    0.0030718...],
[ 695.        ,    0.0028639...],
[ 696.        ,    0.0026738...],
[ 697.        ,    0.0025000...],
[ 698.        ,    0.0023401...],
[ 699.        ,    0.0021918...],
[ 700.        ,    0.0020526...],
[ 701.        ,    0.0019207...],
[ 702.        ,    0.001796 ...],
[ 703.        ,    0.0016784...],
[ 704.        ,    0.0015683...],
```

```
[ 705.        ,    0.0014657...],
[ 706.        ,    0.0013702...],
[ 707.        ,    0.001281 ...],
[ 708.        ,    0.0011976...],
[ 709.        ,    0.0011195...],
[ 710.        ,    0.0010464...],
[ 711.        ,    0.0009776...],
[ 712.        ,    0.0009131...],
[ 713.        ,    0.0008525...],
[ 714.        ,    0.0007958...],
[ 715.        ,    0.0007427...],
[ 716.        ,    0.0006929...],
[ 717.        ,    0.0006462...],
[ 718.        ,    0.0006026...],
[ 719.        ,    0.0005619...],
[ 720.        ,    0.0005240...],
[ 721.        ,    0.0004888...],
[ 722.        ,    0.0004561...],
[ 723.        ,    0.0004255...],
[ 724.        ,    0.0003971...],
[ 725.        ,    0.0003704...],
[ 726.        ,    0.0003455...],
[ 727.        ,    0.0003221...],
[ 728.        ,    0.0003001...],
[ 729.        ,    0.0002796...],
[ 730.        ,    0.0002604...],
[ 731.        ,    0.0002423...],
[ 732.        ,    0.0002254...],
[ 733.        ,    0.0002095...],
[ 734.        ,    0.0001947...],
[ 735.        ,    0.0001809...],
[ 736.        ,    0.0001680...],
[ 737.        ,    0.0001560...],
[ 738.        ,    0.0001449...],
[ 739.        ,    0.0001345...],
[ 740.        ,    0.0001249...],
[ 741.        ,    0.0001159...],
[ 742.        ,    0.0001076...],
[ 743.        ,    0.0000999...],
[ 744.        ,    0.0000927...],
[ 745.        ,    0.0000862...],
[ 746.        ,    0.0000801...],
[ 747.        ,    0.0000745...],
[ 748.        ,    0.0000693...],
[ 749.        ,    0.0000646...],
[ 750.        ,    0.0000602...],
[ 751.        ,    0.0000561...],
[ 752.        ,    0.0000523...],
[ 753.        ,    0.0000488...],
[ 754.        ,    0.0000456...],
[ 755.        ,    0.0000425...],
[ 756.        ,    0.0000397...],
[ 757.        ,    0.0000370...],
[ 758.        ,    0.0000346...],
[ 759.        ,    0.0000322...],
[ 760.        ,    0.0000301...],
[ 761.        ,    0.0000281...],
[ 762.        ,    0.0000262...],
```

```
                    [ 763.        ,    0.0000244...],
                    [ 764.        ,    0.0000228...],
                    [ 765.        ,    0.0000213...],
                    [ 766.        ,    0.0000198...],
                    [ 767.        ,    0.0000185...],
                    [ 768.        ,    0.0000173...],
                    [ 769.        ,    0.0000161...],
                    [ 770.        ,    0.0000150...],
                    [ 771.        ,    0.0000140...],
                    [ 772.        ,    0.0000131...],
                    [ 773.        ,    0.0000122...],
                    [ 774.        ,    0.0000114...],
                    [ 775.        ,    0.0000106...],
                    [ 776.        ,    0.0000099...],
                    [ 777.        ,    0.0000092...],
                    [ 778.        ,    0.0000086...],
                    [ 779.        ,    0.0000080...],
                    [ 780.        ,    0.0000075...]],
                   interpolator=SpragueInterpolator,
                   interpolator_args={},
                   extrapolator=Extrapolator,
                   extrapolator_args={...})
```

Dataset

colour

LEFS	Aggregated luminous efficiency functions.
PHOTOPIC_LEFS	Photopic luminous efficiency functions.
SCOTOPIC_LEFS	Scotopic luminous efficiency functions.

colour.LEFS

colour.**LEFS = CaseInsensitiveMapping({u'CIE 1924 Photopic Standard Observer': ..., u'CIE 1964 Photopic 10 De**

Aggregated luminous efficiency functions.

References

- *[CVRd]*
- *[CVRf]*
- *[Wiks]*

LEFS [CaseInsensitiveMapping] **{'CIE 1924 Photopic Standard Observer', 'Judd Modified CIE 1951 Photopic Standard Observer', 'Judd-Vos Modified CIE 1978 Photopic Standard Observer', 'CIE 1964 Photopic 10 Degree Standard Observer', 'CIE 2008 2 Degree Physiologically Relevant LEF', 'CIE 2008 10 Degree Physiologically Relevant LEF', 'CIE 1951 Scotopic Standard Observer'}**

colour.PHOTOPIC_LEFS

colour.**PHOTOPIC_LEFS = CaseInsensitiveMapping({u'CIE 1924 Photopic Standard Observer': ..., u'CIE 1964 Photo**
Photopic luminous efficiency functions.

References

- *[CVRd]*
- *[CVRf]*

PHOTOPIC_LEFS [CaseInsensitiveMapping] **{'CIE 1924 Photopic Standard Observer', 'Judd Modified CIE 1951 Photopic Standard Observer', 'Judd-Vos Modified CIE 1978 Photopic Standard Observer', 'CIE 1964 Photopic 10 Degree Standard Observer', 'CIE 2008 2 Degree Physiologically Relevant LEF', 'CIE 2008 10 Degree Physiologically Relevant LEF'}**

Aliases:

- 'cie_2_1924': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 Photopic 10 Degree Standard Observer'

colour.SCOTOPIC_LEFS

colour.**SCOTOPIC_LEFS = CaseInsensitiveMapping({u'cie_1951': ..., u'CIE 1951 Scotopic Standard Observer': ...**
Scotopic luminous efficiency functions.

References

- *[CVRf]*

SCOTOPIC_LEFS [CaseInsensitiveMapping] **{'CIE 1951 Scotopic Standard Observer', }**

Aliases:

- 'cie_1951': 'CIE 1951 Scotopic Standard Observer'

Lightness Computation

colour

lightness(Y[, method])	Returns the *Lightness* L using given method.
LIGHTNESS_METHODS	Supported *Lightness* computations methods.

colour.lightness

colour.**lightness**(*Y*, *method=u'CIE 1976'*, ***kwargs*)
Returns the *Lightness* L using given method.

Parameters

- **Y** (numeric or array_like) – *luminance* Y.
- **method** (unicode, optional) – **{'CIE 1976', 'Glasser 1958', 'Wyszecki 1963', 'Fairchild 2010', 'Fairchild 2011'}**, Computation method.

Other Parameters

- **Y_n** (*numeric or array_like, optional*) – {colour.colorimetry.lightness_CIE1976()}, White reference *luminance* Y_n.
- **epsilon** (*numeric or array_like, optional*) – {colour.colorimetry.lightness_Fairchild2010(), colour.colorimetry.lightness_Fairchild2011()}, ϵ exponent.

Returns *Lightness* L.

Return type numeric or array_like

Notes

- Input *luminance* Y and optional Y_n are in domain [0, 100] or [0, ∞].
- Output *Lightness* L is in range [0, 100].

References

- *[FW10]*
- *[FC11]*
- *[GMRS58]*
- *[Lin03a]*
- *[Wikn]*
- *[Wys63]*
- *[WS00a]*

Examples

```
>>> lightness(10.08)
37.9856290...
>>> lightness(10.08, Y_n=100)
37.9856290...
>>> lightness(10.08, Y_n=95)
38.9165987...
>>> lightness(10.08, method='Glasser 1958')
36.2505626...
>>> lightness(10.08, method='Wyszecki 1963')
37.0041149...
>>> lightness(10.08 / 100, epsilon=1.836, method='Fairchild 2010')
...
24.9022902...
```

colour.LIGHTNESS_METHODS

`colour.LIGHTNESS_METHODS = CaseInsensitiveMapping({u'Lstar1976': ..., u'CIE 1976': ..., u'Fairchild 2011':`

Supported *Lightness* computations methods.

References

- *[FW10]*
- *[FC11]*
- *[GMRS58]*
- *[Lin03a]*
- *[Wys63]*
- *[WS00a]*

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 alii (1958)* method.

colour.colorimetry.lightness_Glasser1958

`colour.colorimetry.lightness_Glasser1958`(Y)

Returns the *Lightness* L of given *luminance* Y using *Glasser et alii (1958)* method.

Parameters **Y** (`numeric or array_like`) – *luminance* Y.

Returns *Lightness* L.

Return type numeric or array_like

Notes

- Input *luminance* Y is in domain [0, 100].
- Output *Lightness* L is in range [0, 100].

References

- *[GMRS58]*

Examples

```
>>> lightness_Glasser1958(10.08)
36.2505626...
```

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

- Input *luminance* Y is in domain [0, 100].
- Output *Lightness* W is in range [0, 100].

References

- *[Wys63]*

Examples

```
>>> lightness_Wyszecki1963(10.08)
37.0041149...
```

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.

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

- Input *luminance* Y and Y_n are in domain [0, 100].
- Output *Lightness* L^* is in range [0, 100].

References

- *[Lin03a]*
- *[WS00a]*

Examples

```
>>> lightness_CIE1976(10.08)
37.9856290...
```

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

- Input *luminance* Y is in domain [0, ∞].

References

- *[FW10]*

Examples

```
>>> lightness_Fairchild2010(10.08 / 100)
24.9022902...
```

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 accordingly to *Michealis-Menten* kinetics.

colour.colorimetry.lightness_Fairchild2011

`colour.colorimetry.`**`lightness_Fairchild2011`**(*Y*, *epsilon=0.71*, *method=u'hdr-CIELAB'*)

Computes *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Chen (2011)* method accordingly 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

Warning: The input domain of that definition is non standard!

Notes

- Input *luminance* Y is in domain [0, ∞].

References

- *[FC11]*

Examples

```
>>> lightness_Fairchild2011(10.08 / 100)
26.45950981...
>>> lightness_Fairchild2011(10.08 / 100, method='hdr-IPT')
...
26.3524672...
```

Luminance Computation

colour

luminance(LV[, method])	Returns the *luminance* Y of given *Lightness* L^* or given *Munsell* value V.
LUMINANCE_METHODS	Supported *luminance* computations methods.

colour.luminance

colour.**luminance**(*LV*, *method=u'CIE 1976'*, ***kwargs*)

Returns the *luminance* Y of given *Lightness* L^* or given *Munsell* value V.

Parameters

- **LV** (numeric or array_like) – *Lightness* L^* or *Munsell* value V.
- **method** (unicode, optional) – **{'CIE 1976', 'Newhall 1943', 'ASTM D1535-08', 'Fairchild 2010', 'Fairchild 2011'}**, Computation method.

Other Parameters

- **Y_n** (*numeric or array_like, optional*) – {colour.colorimetry.luminance_CIE1976()}, White reference *luminance* Y_n.
- **epsilon** (*numeric or array_like, optional*) – {colour.colorimetry.lightness_Fairchild2010(), colour.colorimetry.lightness_Fairchild2011()}, ϵ exponent.

Returns *luminance* Y.

Return type numeric or array_like

Notes

- Input *LV* is in domain [0, 100], [0, 10] or [0, 1] and optional *luminance* Y_n is in domain [0, 100].

- Output *luminance* Y is in range [0, 100] or [0, math:*infty*].

References

- *[ASTMInternational08]*
- *[FW10]*
- *[FC11]*
- *[Lin03a]*
- *[NNJ43]*
- *[Wikp]*
- *[WS00a]*

Examples

```
>>> luminance(37.98562910)
10.0800000...
>>> luminance(37.98562910, Y_n=100)
10.0800000...
>>> luminance(37.98562910, Y_n=95)
9.5760000...
>>> luminance(3.74629715, method='Newhall 1943')
10.4089874...
>>> luminance(3.74629715, method='ASTM D1535-08')
10.1488096...
>>> luminance(24.902290269546651, epsilon=1.836, method='Fairchild 2010')
...
0.1007999...
```

colour.LUMINANCE_METHODS

colour.**LUMINANCE_METHODS = CaseInsensitiveMapping({u'cie1976': ..., u'Newhall 1943': ..., u'ASTM D1535-08':**
Supported *luminance* computations methods.

References

- *[ASTMInternational08]*
- *[FW10]*
- *[FC11]*
- *[Lin03a]*
- *[NNJ43]*
- *[WS00a]*

LUMINANCE_METHODS [CaseInsensitiveMapping] **{'Newhall 1943', 'ASTM D1535-08', 'CIE 1976', 'Fairchild 2010'}**

Aliases:

- 'astm2008': 'ASTM D1535-08'
- 'cie1976': 'CIE 1976'

Newhall, Nickerson and Judd (1943)

`colour.colorimetry`

luminance_Newhall1943(V)	Returns the *luminance* R_Y of given *Munsell* value V using *Newhall et alii (1943)* method.

colour.colorimetry.luminance_Newhall1943

colour.colorimetry.**luminance_Newhall1943**(V)

Returns the *luminance* R_Y of given *Munsell* value V using *Newhall et alii (1943)* method.

Parameters **V** (numeric or array_like) – *Munsell* value V.

Returns *luminance* R_Y.

Return type numeric or array_like

Notes

- Input *Munsell* value V is in domain [0, 10].
- Output *luminance* R_Y is in range [0, 100].

References

- *[NNJ43]*

Examples

```
>>> luminance_Newhall1943(3.74629715382)
10.4089874...
```

CIE 1976

`colour.colorimetry`

luminance_CIE1976(Lstar[, Y_n])	Returns the *luminance* Y of given *Lightness* L^* with given reference white *luminance* Y_n.

colour.colorimetry.luminance_CIE1976

colour.colorimetry.**luminance_CIE1976**(*Lstar*, *Y_n=100*)

Returns the *luminance* Y of given *Lightness* L^* with given reference white *luminance* Y_n.

Parameters

- **Lstar** (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

- Input *Lightness* L^* and reference white *luminance* Y_n are in domain [0, 100].
- Output *luminance* Y is in range [0, 100].

References

- *[Lin03a]*
- *[WS00a]*

Examples

```
>>> luminance_CIE1976(37.98562910)
10.0800000...
>>> luminance_CIE1976(37.98562910, 95)
9.5760000...
```

ASTM D1535-08e1

colour.colorimetry

luminance_ASTMD153508(V)	Returns the *luminance* Y of given *Munsell* value V using *ASTM D1535-08e1* method.

colour.colorimetry.luminance_ASTMD153508

colour.colorimetry.**luminance_ASTMD153508**(*V*)

Returns the *luminance* Y of given *Munsell* value V using *ASTM D1535-08e1* method.

Parameters **V** (numeric or array_like) – *Munsell* value V.

Returns *luminance* Y.

Return type numeric or array_like

Notes

- Input *Munsell* value V is in domain [0, 10].
- Output *luminance* Y is in range [0, 100].

References

- *[ASTMInternational08]*

Examples

```
>>> luminance_ASTMD153508(3.74629715382)
10.1488096...
```

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

- Output *luminance* Y is in range [0, math:*infty*].

References

- *[FW10]*

Examples

```
>>> luminance_Fairchild2010(24.902290269546651, 1.836)
...
0.1007999...
```

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 accordingly to *Michealis-Menten* kinetics.

colour.colorimetry.luminance_Fairchild2011

`colour.colorimetry.`**`luminance_Fairchild2011`**(*L_hdr*, *epsilon=0.71*, *method=u'hdr-CIELAB'*)

Computes *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Chen (2011)* method accordingly 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

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

Notes

- Output *luminance* Y is in range [0, math:*infty*].

References

- *[FC11]*

Examples

```
>>> luminance_Fairchild2011(26.459509817572265)
0.1007999...
>>> luminance_Fairchild2011(26.352467267703549, method='hdr-IPT')
```

```
...
0.1007999...
```

Whiteness Computation

`colour`

`whiteness`([method])	Returns the *whiteness* W using given method.
`WHITENESS_METHODS`	Supported *whiteness* computations methods.

colour.whiteness

`colour.whiteness`(*method=u'CIE 2004'*, ***kwargs*)

Returns the *whiteness* W using given method.

Parameters method (`unicode`, `optional`) – **{'CIE 2004', 'Berger 1959', 'Taube 1960', 'Stensby 1968', 'ASTM E313', 'Ganz 1979', 'CIE 2004'}**, Computation method.

Other Parameters

- **XYZ** (*array_like*) – {`colour.colorimetry.whiteness_Berger1959()`, `colour.colorimetry.whiteness_Taube1960()`, `colour.colorimetry.whiteness_ASTME313()`}, *CIE XYZ* tristimulus values of sample.
- **XYZ_0** (*array_like*) – {`colour.colorimetry.whiteness_Berger1959()`, `colour.colorimetry.whiteness_Taube1960()`}, *CIE XYZ* tristimulus values of reference white.
- **Lab** (*array_like*) – {`colour.colorimetry.whiteness_Stensby1968()`}, *CIE L*a*b** colourspace array of sample.
- **xy** (*array_like*) – {`colour.colorimetry.whiteness_Ganz1979()`, `colour.colorimetry.whiteness_CIE2004()`}, Chromaticity coordinates *xy* of sample.
- **Y** (*numeric or array_like*) – {`colour.colorimetry.whiteness_Ganz1979()`, `colour.colorimetry.whiteness_CIE2004()`}, Tristimulus Y value of sample.
- **xy_n** (*array_like*) – {`colour.colorimetry.whiteness_CIE2004()`}, Chromaticity coordinates *xy_n* of perfect diffuser.
- **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

References

- *[CIET14804h]*
- *[WS00i]*
- *[XRP12]*

- *[Wikz]*

Examples

```
>>> xy = np.array([0.3167, 0.3334])
>>> Y = 100
>>> xy_n = np.array([0.3139, 0.3311])
>>> whiteness(xy=xy, Y=Y, xy_n=xy_n)
array([ 93.85...,  -1.305...])
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> method = 'Taube 1960'
>>> whiteness(XYZ=XYZ, XYZ_0=XYZ_0, method=method)
91.4071738...
```

colour.WHITENESS_METHODS

`colour.WHITENESS_METHODS = CaseInsensitiveMapping({u'ASTM E313': ..., u'Ganz 1979': ..., u'Stensby 1968': .`
Supported *whiteness* computations methods.

References

- *[CIET14804h]*
- *[XRP12]*

WHITENESS_METHODS [CaseInsensitiveMapping] **{'CIE 2004', 'Berger 1959', 'Taube 1960', 'Stensby 1968', 'ASTM E313', 'Ganz 1979', 'CIE 2004'}**

Aliases:

- 'cie2004': 'CIE 2004'

Berger (1959)

`colour.colorimetry`

`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

- Input *CIE XYZ* and *CIE XYZ_0* tristimulus values are in domain [0, 100].
- *Whiteness* WI values larger than 33.33 indicate a bluish white and values smaller than 33.33 indicate a yellowish white.

Warning: The input domain of that definition is non standard!

References

- *[XRP12]*

Examples

```
>>> 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

- Input *CIE XYZ* and *CIE XYZ_0* tristimulus values are in domain [0, 100].
- *Whiteness* WI values larger than 100 indicate a bluish white and values smaller than 100 indicate a yellowish white.

References

- *[XRP12]*

Examples

```
>>> 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

- Input *CIE L*a*b** colourspace array is in domain [0, 100].
- *Whiteness* WI values larger than 100 indicate a bluish white and values smaller than 100 indicate a yellowish white.

References

- *[XRP12]*

Examples

```
>>> 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

Warning: The input domain of that definition is non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].

References

- *[XRP12]*

Examples

```
>>> 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 *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 *xy* chromaticity coordinates using *Ganz and Griesser (1979)* method.

Parameters

- **xy** (array_like) – Chromaticity coordinates *xy* of sample.
- **Y** (numeric or array_like) – Tristimulus Y value of sample.

Returns *Whiteness* W and *tint* T.

Return type ndarray

Warning: The input domain of that definition is non standard!

Notes

- Input tristimulus Y value is in domain [0, 100].
- The formula coefficients are valid for *CIE Standard Illuminant D Series D65* and *CIE 1964 10 Degree Standard Observer*.
- Positive output *tint* T values indicate a greener tint while negative values indicate a redder tint.
- Whiteness differences of less than 5 Ganz units appear to be indistinguishable to the human eye.
- Tint differences of less than 0.5 Ganz units appear to be indistinguishable to the human eye.

References

- *[XRP12]*

Examples

```
>>> 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 *xy* chromaticity coordinates using *CIE 2004* method.

colour.colorimetry.whiteness_CIE2004

colour.colorimetry.**whiteness_CIE2004**(*xy, Y, xy_n, observer=u'CIE 1931 2 Degree Standard Observer'*)

Returns the *whiteness* W or W_{10} and *tint* T or T_{10} of given sample *xy* chromaticity coordinates using *CIE 2004* method.

Parameters

- **xy** (array_like) – Chromaticity coordinates *xy* of sample.
- **Y** (numeric or array_like) – Tristimulus Y value of sample.
- **xy_n** (array_like) – Chromaticity coordinates *xy_n* of perfect diffuser.
- **observer** (unicode, optional) – **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}**, *CIE Standard Observer* used for computations, *tint* T or T_{10} value is dependent on viewing field angular subtense.

Returns *Whiteness* W or W_{10} and *tint* T or T_{10} of given sample.

Return type ndarray

Warning: The input domain of that definition is non standard!

Notes

- Input tristimulus Y value is in domain [0, 100].
- 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

- *[CIET14804h]*

Examples

```
>>> 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* computations methods.

colour.yellowness

colour.**yellowness**(*XYZ*, *method=u'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

References

- [XRP12]

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({u'ASTM E313': ..., u'ASTM D1925': ...})**

Supported *yellowness* computations methods.

References

- [XRP12]

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

> **Warning:** The input domain of that definition is non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].

References

- [XRP12]

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

Warning: The input domain of that definition is non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].

References

- *[XRP12]*

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

CIE_E	*CIE* ϵ constant.
CIE_K	*CIE* κ constant.
K_M	Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.
KP_M	Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

colour.constants.CIE_E

colour.constants.**CIE_E = 0.008856451679035631**

CIE ϵ constant.

CIE_E : numeric

Notes

- The original *CIE* value for ϵ is $\epsilon = 0.008856$, Lindbloom (2003) has shown that this value is causing a discontinuity at the junction point of the two functions grafted together to create the *Lightness* L^* function.

  That discontinuity can be avoided by using the rational representation as follows: $\epsilon = 216\ /\ 24389$.

References

- *[Lin03a]*

colour.constants.CIE_K

colour.constants.**CIE_K = 903.2962962962963**

CIE κ constant.

CIE_K : numeric

Notes

- The original *CIE* value for κ is $\kappa = 903.3$, Lindbloom (2003) has shown that this value is causing a discontinuity at the junction point of the two functions grafted together to create the *Lightness* L^* function.

  That discontinuity can be avoided by using the rational representation as follows: $k = 24389\ /\ 27$.

References

- *[Lin03a]*

colour.constants.K_M

`colour.constants.K_M = 683.0`

Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.

K_M : numeric

Notes

- To be adequate for all practical applications the K_m value has been rounded from the original 683.002 value.

References

- *[WS00e]*

colour.constants.KP_M

`colour.constants.KP_M = 1700.0`

Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

KP_M : numeric

Notes

- To be adequate for all practical applications the K'_m value has been rounded from the original 1700.06 value.

References

- *[WS00e]*

CODATA

`colour.constants`

AVOGADRO_CONSTANT	Avogadro constant.
BOLTZMANN_CONSTANT	Boltzmann constant.
LIGHT_SPEED	Speed of light in vacuum.
PLANCK_CONSTANT	Planck constant.

colour.constants.AVOGADRO_CONSTANT

`colour.constants.AVOGADRO_CONSTANT = 6.02214179e+23`

Avogadro constant.

AVOGADRO_CONSTANT : numeric

colour.constants.BOLTZMANN_CONSTANT

`colour.constants.BOLTZMANN_CONSTANT = 1.38065e-23`

Boltzmann constant.

BOLTZMANN_CONSTANT : numeric

colour.constants.LIGHT_SPEED

`colour.constants.LIGHT_SPEED = 299792458.0`

Speed of light in vacuum.

LIGHT_SPEED : numeric

colour.constants.PLANCK_CONSTANT

`colour.constants.PLANCK_CONSTANT = 6.62607e-34`

Planck constant.

PLANCK_CONSTANT : numeric

Common

`colour.constants`

DEFAULT_FLOAT_DTYPE	alias of `float64`
EPSILON	
FLOATING_POINT_NUMBER_PATTERN	unicode(object=") -> unicode object
INTEGER_THRESHOLD	Integer threshold value.

colour.constants.DEFAULT_FLOAT_DTYPE

`colour.constants.DEFAULT_FLOAT_DTYPE`

alias of `float64`

colour.constants.EPSILON

`colour.constants.EPSILON = 2.220446049250313e-16`

colour.constants.FLOATING_POINT_NUMBER_PATTERN

`colour.constants.FLOATING_POINT_NUMBER_PATTERN = u'[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)?'`

unicode(object=") -> unicode object unicode(string[, encoding[, errors]]) -> unicode object

Create a new Unicode object from the given encoded string. encoding defaults to the current default string encoding. errors can be 'strict', 'replace' or 'ignore' and defaults to 'strict'.

colour.constants.INTEGER_THRESHOLD

colour.constants.**INTEGER_THRESHOLD = 0.001**

Integer threshold value.

INTEGER_THRESHOLD : numeric

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.
MultiSignal([data, domain, labels])	Defines the base class for multi-continuous signal, a container for multiple colour.continuous.Signal sub-class instances.

colour.continuous.AbstractContinuousFunction

class colour.continuous.**AbstractContinuousFunction**(*name=None*)

Defines the base class for abstract continuous function.

This is an ABCMeta abstract class that must be inherited by sub-classes.

The sub-classes are expected to implement the colour.continuous.AbstractContinuousFunction.function() method so that evaluating the function for any independent domain $x \in \mathbb{R}$ variable returns a corresponding range $y \in \mathbb{R}$ variable. A conventional implementation adopts an interpolating function encapsulated inside an extrapolating function. The resulting function independent domain, stored as discrete values in the colour.continuous.AbstractContinuousFunction.domain attribute corresponds with the function dependent and already known range stored in the colour.continuous.AbstractContinuousFunction.range attribute.

Parameters **name** (unicode, optional) – Continuous function name.

name

domain

range

interpolator

interpolator_args

extrapolator

extrapolator_args

function

__str__()

__repr__()

__hash__()

__getitem__()

__setitem__()

__contains__()

__len__()

__eq__()

__ne__()

__iadd__()

__add__()

__isub__()

__sub__()

__imul__()

__mul__()

__idiv__()

__div__()

__ipow__()

__pow__()

arithmetical_operation()

fill_nan()

domain_distance()

is_uniform()

copy()

__init__(*name=None*)

Methods

`__init__`([name])	
`arithmetical_operation`(a, operation[, in_place])	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.
`copy`()	Returns a copy of the sub-class instance, must be reimplemented by sub-classes.
`domain_distance`(a)	Returns the euclidean distance between given array and independent domain x closest element.

Continued on next page

Table 3.103 – continued from previous page

fill_nan([method, default])	Fill NaNs in independent domain x variable and corresponding range y variable using given method, must be reimplemented by sub-classes.
is_uniform()	Returns if independent domain x variable is uniform.

colour.continuous.Signal

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

Defines the base class for continuous signal.

The class implements the `Signal.function()` method so that evaluating the function for any independent domain $x \in \mathbb{R}$ variable returns a corresponding range $y \in \mathbb{R}$ variable. It adopts an interpolating function encapsulated inside an extrapolating function. The resulting function independent domain, stored as discrete values in the `colour.continuous.Signal.domain` attribute corresponds with the function dependent and already known range stored in the `colour.continuous.Signal.range` attribute.

Parameters

- **data** (Series or Signal or array_like or dict_like, optional) – Data to be stored in the continuous signal.
- **domain** (array_like, optional) – Values to initialise the `colour.continuous.Signal.domain` attribute with. If both data and domain arguments are defined, the latter with be used to initialise the `colour.continuous.Signal.domain` attribute.

Other Parameters

- **name** (*unicode, optional*) – Continuous signal name.
- **dtype** (*type, optional*) – **{np.float16, np.float32, np.float64, np.float128}**, Floating point data type.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function.

dtype

domain

range

interpolator

interpolator_args

extrapolator

extrapolator_args

function

__str__()

__repr__()

__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*)

Methods

`__init__`([data, domain])	

Continued on next page

Table 3.104 – continued from previous page

arithmetical_operation(a, operation[, in_place])	Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.
copy()	Returns a copy of the sub-class instance, must be reimplemented by sub-classes.
domain_distance(a)	Returns the euclidean distance between given array and independent domain x closest element.
fill_nan([method, default])	Fill NaNs in independent domain x variable and corresponding range y variable using given method.
is_uniform()	Returns if independent domain x variable is uniform.
signal_unpack_data([data, domain, dtype])	Unpack given data for continuous signal instantiation.
to_series()	Converts the continuous signal to a *Pandas* Series class instance.

colour.continuous.MultiSignal

class colour.continuous.**MultiSignal**(*data=None, domain=None, labels=None, **kwargs*)

Defines the base class for multi-continuous signal, a container for multiple colour.continuous.Signal sub-class instances.

Parameters

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

Other Parameters

- **name** (*unicode, optional*) – Multi-continuous signal name.
- **dtype** (*type, optional*) – **{np.float16, np.float32, np.float64, np.float128}**, Floating point data type.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.continuous.Signal sub-class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the colour.continuous.Signal sub-class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the colour.continuous.Signal sub-class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the colour.continuous.Signal sub-class instances.
- **signal_type** (*type, optional*) – The colour.continuous.Signal sub-class type used for instances.

dtype

domain

range

interpolator

interpolator_args

extrapolator

extrapolator_args

function

signals

labels

__str__()

__repr__()

__getitem__()

__setitem__()

__contains__()

__eq__()

__ne__()

arithmetical_operation()

multi_signal_unpack_data()

fill_nan()

to_dataframe()

Examples

Instantiation with implicit *domain* and a single signal:

```
>>> range_ = np.linspace(10, 100, 10)
>>> print(MultiSignal(range_))
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
```

Instantiation with explicit *domain* and a single signal:

```
>>> domain = np.arange(100, 1100, 100)
>>> print(MultiSignal(range_, domain))
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
```

```
[  400.    40.]
[  500.    50.]
[  600.    60.]
[  700.    70.]
[  800.    80.]
[  900.    90.]
[ 1000.   100.]]
```

Instantiation with multiple signals:

```
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> print(MultiSignal(range_, domain))
[[  100.    10.    20.    30.]
 [  200.    20.    30.    40.]
 [  300.    30.    40.    50.]
 [  400.    40.    50.    60.]
 [  500.    50.    60.    70.]
 [  600.    60.    70.    80.]
 [  700.    70.    80.    90.]
 [  800.    80.    90.   100.]
 [  900.    90.   100.   110.]
 [ 1000.   100.   110.   120.]]
```

Instantiation with a *dict*:

```
>>> print(MultiSignal(dict(zip(domain, range_))))
[[  100.    10.    20.    30.]
 [  200.    20.    30.    40.]
 [  300.    30.    40.    50.]
 [  400.    40.    50.    60.]
 [  500.    50.    60.    70.]
 [  600.    60.    70.    80.]
 [  700.    70.    80.    90.]
 [  800.    80.    90.   100.]
 [  900.    90.   100.   110.]
 [ 1000.   100.   110.   120.]]
```

Instantiation using a *Signal* sub-class:

```
>>> class NotSignal(Signal):
...     pass
```

```
>>> multi_signal = MultiSignal(range_, domain, signal_type=NotSignal)
>>> print(multi_signal)
[[  100.    10.    20.    30.]
 [  200.    20.    30.    40.]
 [  300.    30.    40.    50.]
 [  400.    40.    50.    60.]
 [  500.    50.    60.    70.]
 [  600.    60.    70.    80.]
 [  700.    70.    80.    90.]
 [  800.    80.    90.   100.]
 [  900.    90.   100.   110.]
 [ 1000.   100.   110.   120.]]
 >>> type(multi_signal.signals[0])
 <class 'multi_signal.NotSignal'>
```

Instantiation with a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...     print(MultiSignal(
...         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(MultiSignal(
...         DataFrame(data, domain)))
[[  100.    10.    20.    30.]
 [  200.    20.    30.    40.]
 [  300.    30.    40.    50.]
 [  400.    40.    50.    60.]
 [  500.    50.    60.    70.]
 [  600.    60.    70.    80.]
 [  700.    70.    80.    90.]
 [  800.    80.    90.   100.]
 [  900.    90.   100.   110.]
 [ 1000.   100.   110.   120.]]
```

Retrieving domain *y* variable for arbitrary range *x* variable:

```
>>> x = 150
>>> range_ = tstack([np.sin(np.linspace(0, 1, 10))] * 3)
>>> range_ += np.array([0.0, 0.25, 0.5])
>>> MultiSignal(range_, domain)[x]
array([ 0.0359701...,  0.2845447...,  0.5331193...])
>>> x = np.linspace(100, 1000, 3)
>>> MultiSignal(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
>>> MultiSignal(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x]
array([ 0.0555274...,  0.3055274...,  0.5555274...])
>>> x = np.linspace(100, 1000, 3)
>>> MultiSignal(
```

```
...     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*)

Methods

__init__([data, domain, labels])	
arithmetical_operation(a, operation[, in_place])	Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.
copy()	Returns a copy of the sub-class instance, must be reimplemented by sub-classes.
domain_distance(a)	Returns the euclidean distance between given array and independent domain x closest element.
fill_nan([method, default])	Fill NaNs in independent domain x variable and corresponding range y variable using given method.
is_uniform()	Returns if independent domain x variable is uniform.
multi_signal_unpack_data([data, domain, ...])	Unpack given data for multi-continuous signal instantiation.
to_dataframe()	Converts the continuous signal to a *Pandas* DataFrame class instance.

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.

colour.corresponding_chromaticities_prediction

colour.**corresponding_chromaticities_prediction**(*experiment=1*, *model=u'Von Kries'*, ***kwargs*)
Returns the corresponding chromaticities prediction for given chromatic adaptation model.

Parameters

- **experiment** (integer, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number.
- **model** (unicode, optional) – **{'Von Kries', 'CIE 1994', 'CMCCAT2000', 'Fairchild 1990'}**, Chromatic adaptation model.

Other Parameters transform (*unicode, optional*) – {colour.corresponding.corresponding_chromaticities_prediction_VonKries()}, **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, Chromatic adaptation transform.

Returns Corresponding chromaticities prediction.

Return type tuple

References

- *[Bre87]*
- *[CIET13294]*
- *[Fai91]*
- *[Fai13c]*
- *[Fai13b]*
- *[LLRH02]*
- *[WRC12a]*

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction(2, 'CMCCAT2000')
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr)
[((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({u'vonkries': ..., u'Von Krie`

Aggregated corresponding chromaticities prediction models.

References

- *[Bre87]*
- *[CIET13294]*
- *[Fai91]*
- *[Fai13c]*
- *[Fai13b]*
- *[LLRH02]*
- *[WRC12a]*

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

Aliases:

- 'vonkries': 'Von Kries'

Dataset

`colour`

BRENEMAN_EXPERIMENTS	*Breneman (1987)* experiments.
BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES	*Breneman (1987)* experiments primaries chromaticities.

colour.BRENEMAN_EXPERIMENTS

`colour.BRENEMAN_EXPERIMENTS = {1: (BrenemanExperimentResult(name=u'Illuminant', uvp_t=array([ 0.259, 0.526])`

References

- *[Bre87]*

BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES : dict

Fairchild (1990)

`colour.corresponding`

`corresponding_chromaticities_prediction_Fairchild1990([…])`	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`, `optional`) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number.

Returns Corresponding chromaticities prediction.

Return type tuple

References

- *[Bre87]*
- *[Fai91]*
- *[Fai13c]*

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_Fairchild1990(2)
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr)
[((0.207, 0.486), (0.2089528..., 0.4724034...)),
 ((0.449, 0.511), (0.4375652..., 0.5121030...)),
 ((0.263, 0.505), (0.2621362..., 0.4972538...)),
 ((0.322, 0.545), (0.3235312..., 0.5475665...)),
 ((0.316, 0.537), (0.3151390..., 0.5398333...)),
 ((0.265, 0.553), (0.2634745..., 0.5544335...)),
 ((0.221, 0.538), (0.2211595..., 0.5324470...)),
 ((0.135, 0.532), (0.1396949..., 0.5207234...)),
 ((0.145, 0.472), (0.1512288..., 0.4533041...)),
 ((0.163, 0.331), (0.1715691..., 0.3026264...)),
 ((0.176, 0.431), (0.1825792..., 0.4077892...)),
 ((0.244, 0.349), (0.2418904..., 0.3413401...))]
```

CIE 1994

colour.corresponding

corresponding_chromaticities_prediction_CIE1994([...])	Returns the corresponding chromaticities prediction for *CIE 1994* chromatic adaptation model.

colour.corresponding.corresponding_chromaticities_prediction_CIE1994

colour.corresponding.**corresponding_chromaticities_prediction_CIE1994**(*experiment=1*)

Returns the corresponding chromaticities prediction for *CIE 1994* chromatic adaptation model.

Parameters **experiment** (integer, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number.

Returns Corresponding chromaticities prediction.

Return type tuple

References

- *[Bre87]*
- *[CIET13294]*

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CIE1994(2)
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr)
[((0.207, 0.486), (0.2133909..., 0.4939794...)),
 ((0.449, 0.511), (0.4450345..., 0.5120939...)),
 ((0.263, 0.505), (0.2693262..., 0.5083212...)),
 ((0.322, 0.545), (0.3308593..., 0.5443940...)),
 ((0.316, 0.537), (0.3225195..., 0.5377826...)),
 ((0.265, 0.553), (0.2709737..., 0.5513666...)),
 ((0.221, 0.538), (0.2280786..., 0.5351592...)),
 ((0.135, 0.532), (0.1439436..., 0.5303576...)),
 ((0.145, 0.472), (0.1500743..., 0.4842895...)),
 ((0.163, 0.331), (0.1559955..., 0.3772379...)),
 ((0.176, 0.431), (0.1806318..., 0.4518475...)),
 ((0.244, 0.349), (0.2454445..., 0.4018004...))]
```

CMCCAT2000

colour.corresponding

corresponding_chromaticities_prediction_CMCCAT2000([...])	Returns the corresponding chromaticities prediction for *CMCCAT2000* chromatic adaptation model.

colour.corresponding.corresponding_chromaticities_prediction_CMCCAT2000

colour.corresponding.**corresponding_chromaticities_prediction_CMCCAT2000**(*experiment=1*)

Returns the corresponding chromaticities prediction for *CMCCAT2000* chromatic adaptation model.

Parameters **experiment** (integer, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number.

Returns Corresponding chromaticities prediction.

Return type tuple

References

- *[Bre87]*
- *[LLRH02]*
- *[WRC12a]*

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CMCCAT2000(2)
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr)
[((0.207, 0.486), (0.2083210..., 0.4727168...)),
 ((0.449, 0.511), (0.4459270..., 0.5077735...)),
 ((0.263, 0.505), (0.2640262..., 0.4955361...)),
 ((0.322, 0.545), (0.3316884..., 0.5431580...)),
 ((0.316, 0.537), (0.3222624..., 0.5357624...)),
 ((0.265, 0.553), (0.2710705..., 0.5501997...)),
 ((0.221, 0.538), (0.2261826..., 0.5294740...)),
 ((0.135, 0.532), (0.1439693..., 0.5190984...)),
 ((0.145, 0.472), (0.1494835..., 0.4556760...)),
 ((0.163, 0.331), (0.1563172..., 0.3164151...)),
 ((0.176, 0.431), (0.1763199..., 0.4127589...)),
 ((0.244, 0.349), (0.2287638..., 0.3499324...))]
```

Von Kries

colour.corresponding

corresponding_chromaticities_prediction_VonKries([…])	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=u'CAT02'*)

Returns the corresponding chromaticities prediction for *Von Kries* chromatic adaptation model using given transform.

Parameters

- **experiment** (integer, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number.
- **transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, Chromatic adaptation transform.

Returns Corresponding chromaticities prediction.

Return type tuple

References

- *[Bre87]*
- *[Fai13b]*

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_VonKries(2, 'Bradford')
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr)
[((0.207, 0.486), (0.2082014..., 0.4722922...)),
 ((0.449, 0.511), (0.4489102..., 0.5071602...)),
 ((0.263, 0.505), (0.2643545..., 0.4959631...)),
 ((0.322, 0.545), (0.3348730..., 0.5471220...)),
 ((0.316, 0.537), (0.3248758..., 0.5390589...)),
 ((0.265, 0.553), (0.2733105..., 0.5555028...)),
 ((0.221, 0.538), (0.2271480..., 0.5331317...)),
 ((0.135, 0.532), (0.1442730..., 0.5226804...)),
 ((0.145, 0.472), (0.1498745..., 0.4550785...)),
 ((0.163, 0.331), (0.1564975..., 0.3148795...)),
 ((0.176, 0.431), (0.1760593..., 0.4103772...)),
 ((0.244, 0.349), (0.2259805..., 0.3465291...))]
```

Colour Difference

Delta E

`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} computations 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'}** Computation method.

Other Parameters

- **textiles** (*bool, optional*) – {`colour.difference.delta_E_CIE1994()`, `colour.difference.delta_E_CIE2000()`}, 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 $k_L = k_C = k_H = 1,\ k_1 = 0.045,\ k_2 = 0.015$.
- **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

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...
>>> 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({'cie1994': ..., 'CIE 1994': ..., 'CAM02-LCD': ..., 'CMC':**
Supported ΔE_{ab} computations methods.

References

- *[Lin03b]*
- *[Lin11]*
- *[Lin09b]*
- *[Lin09c]*
- *[LCL06]*
- *[Mel13]*
- *[Wikh]*

DELTA_E_METHODS [CaseInsensitiveMapping] **{'CIE 1976', 'CIE 1994', 'CIE 2000', 'CMC', 'CAM02-LCD', 'CAM02-SCD', 'CAM02-UCS', 'CAM16-LCD', 'CAM16-SCD', 'CAM16-UCS'}**

Aliases:

- 'cie1976': 'CIE 1976'
- 'cie1994': 'CIE 1994'
- 'cie2000': 'CIE 2000'

CIE 1976

colour.difference

delta_E_CIE1976(Lab_1, Lab_2)	Returns the difference ΔE_{ab} between two given *CIE L*a*b** colourspace arrays using *CIE 1976* recommendation.

colour.difference.delta_E_CIE1976

colour.difference.**delta_E_CIE1976**(*Lab_1*, *Lab_2*)
Returns the difference ΔE_{ab} 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_{ab}.

Return type numeric or ndarray

References

- *[Lin03b]*

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_{ab} 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_{ab} 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_{ab}.

Return type numeric or ndarray

Notes

- *CIE 1994* colour differences are not symmetrical: difference between Lab_1 and Lab_2 may not be the same as difference between Lab_2 and Lab_1 thus one colour must be understood to be the reference against which a sample colour is compared.

References

- *[Lin11]*

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE1994(Lab_1, Lab_2)
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_{ab} 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_{ab} 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_{ab}.

Return type numeric or ndarray

Notes

- *CIE 2000* colour differences are not symmetrical: difference between `Lab_1` and `Lab_2` may not be the same as difference between `Lab_2` and `Lab_1` thus one colour must be understood to be the reference against which a sample colour is compared.
- Parametric factors $k_L = k_C = k_H = 1$ weights under *reference conditions*:
  - Illumination: D65 source
  - Illuminance: 1000 lx

- Observer: Normal colour vision
- Background field: Uniform, neutral gray with $L^* = 50$
- Viewing mode: Object
- Sample size: Greater than 4 degrees
- Sample separation: Direct edge contact
- Sample colour-difference magnitude: Lower than 5.0 ΔE_{ab}
- Sample structure: Homogeneous (without texture)

References

- *[Lin09b]*
- *[Mel13]*

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE2000(Lab_1, Lab_2)
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_{ab} 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_{ab} 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_{ab}.

Return type numeric or ndarray

References

- *[Lin09c]*

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CMC(Lab_1, Lab_2)
172.7047712...
```

Luo, Cui and Li (2006)

colour.difference

delta_E_CAM02LCD(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et alii (2017) CAM16-LCD* colourspaces $J'a'b'$ arrays.
delta_E_CAM02SCD(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et alii (2017) CAM16-SCD* colourspaces $J'a'b'$ arrays.
delta_E_CAM02UCS(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et alii (2017) CAM16-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 *Li et alii (2017) CAM16-LCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et alii (2017) CAM16-LCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et alii (2017) CAM16-LCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02LCD()` definition for an usage example.

colour.difference.delta_E_CAM02SCD

`colour.difference.`**`delta_E_CAM02SCD`**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et alii (2017) CAM16-SCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (`array_like`) – Standard / reference *Li et alii (2017) CAM16-SCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (`array_like`) – Sample / test *Li et alii (2017) CAM16-SCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02SCD()` definition for an usage example.

colour.difference.delta_E_CAM02UCS

`colour.difference.`**`delta_E_CAM02UCS`**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et alii (2017) CAM16-UCS* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (`array_like`) – Standard / reference *Li et alii (2017) CAM16-UCS* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (`array_like`) – Sample / test *Li et alii (2017) CAM16-UCS* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02UCS()` definition for an usage example.

Li, Li, Wang, Zu, Luo, Cui, Melgosa, Brill and Pointer (2017)

`colour.difference`

`delta_E_CAM16LCD`(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et alii (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 alii (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 alii (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 alii (2017) CAM16-LCD* colourspaces $J'a'b'$ arrays.

Parameters

- **`Jpapbp_1`** (`array_like`) – Standard / reference *Li et alii (2017) CAM16-LCD* colourspaces $J'a'b'$ array.
- **`Jpapbp_2`** (`array_like`) – Sample / test *Li et alii (2017) CAM16-LCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02LCD()` definition for an usage example.

colour.difference.delta_E_CAM16SCD

`colour.difference.`**`delta_E_CAM16SCD`**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et alii (2017) CAM16-SCD* colourspaces $J'a'b'$ arrays.

Parameters

- **`Jpapbp_1`** (`array_like`) – Standard / reference *Li et alii (2017) CAM16-SCD* colourspaces $J'a'b'$ array.

- **Jpapbp_2** (array_like) – Sample / test *Li et alii (2017) CAM16-SCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02SCD()` definition for an usage example.

colour.difference.delta_E_CAM16UCS

`colour.difference.`**`delta_E_CAM16UCS`**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et alii (2017) CAM16-UCS* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et alii (2017) CAM16-UCS* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et alii (2017) CAM16-UCS* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02UCS()` definition for an usage example.

Input and Output

- *Image Data*
- *CSV Tabular Data*
- *IES TM-27-14 Data*
- *X-Rite Data*

Image Data

colour

read_image(path[, bit_depth])	Reads given image using *OpenImageIO*.
write_image(image, path[, bit_depth])	Writes given image using *OpenImageIO*.

colour.read_image

colour.**read_image**(*path*, *bit_depth=u'float32'*)
Reads given image using *OpenImageIO*.

Parameters

- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Image bit_depth.

Returns Image as a ndarray.

Return type ndarray

Notes

- For convenience, single channel images are squeezed to 2d arrays.

Examples

```
>>> import os
>>> path = os.path.join('tests', 'resources', 'CMSTestPattern.exr')
>>> image = read_image(path)
```

colour.write_image

colour.**write_image**(*image*, *path*, *bit_depth=u'float32'*)
Writes given image using *OpenImageIO*.

Parameters

- **image** (array_like) – Image data.
- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Image bit_depth.

Returns Definition success.

Return type bool

Examples

```
>>> import os
>>> path = os.path.join('tests', 'resources', 'CMSTestPattern.exr')
>>> image = read_image(path)
>>> path = os.path.join('tests', 'resources', 'CMSTestPattern.png')
>>> write_image(image, path, 'uint8')
True
```

CSV Tabular Data

colour

read_spds_from_csv_file(path[, delimiter, …])	Reads the spectral data from given *CSV* file and return its content as an *OrderedDict* of colour.SpectralPowerDistribution classes.
read_spectral_data_from_csv_file(path[, …])	Reads the spectral data from given *CSV* file in the following form:
write_spds_to_csv_file(spds, path[, …])	Writes the given spectral power distributions to given *CSV* file.

colour.read_spds_from_csv_file

colour.**read_spds_from_csv_file**(*path, delimiter=u', ', fields=None, default=0*)

Reads the spectral data from given *CSV* file and return its content as an *OrderedDict* of colour.SpectralPowerDistribution 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.SpectralPowerDistribution 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')
>>> spds = read_spds_from_csv_file(csv_file)
>>> print(tuple(spds.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):
```

```
...     spds['1']
SpectralPowerDistribution([[ 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],
                           [ 605.   ,    0.142],
                           [ 610.   ,    0.146],
                           [ 615.   ,    0.15 ],
                           [ 620.   ,    0.154],
                           [ 625.   ,    0.158],
                           [ 630.   ,    0.163],
                           [ 635.   ,    0.167],
                           [ 640.   ,    0.173],
                           [ 645.   ,    0.18 ],
                           [ 650.   ,    0.188],
                           [ 655.   ,    0.196],
                           [ 660.   ,    0.204],
```

```
                         [ 665.   ,    0.213],
                         [ 670.   ,    0.222],
                         [ 675.   ,    0.231],
                         [ 680.   ,    0.242],
                         [ 685.   ,    0.251],
                         [ 690.   ,    0.261],
                         [ 695.   ,    0.271],
                         [ 700.   ,    0.282],
                         [ 705.   ,    0.294],
                         [ 710.   ,    0.305],
                         [ 715.   ,    0.318],
                         [ 720.   ,    0.334],
                         [ 725.   ,    0.354],
                         [ 730.   ,    0.372],
                         [ 735.   ,    0.392],
                         [ 740.   ,    0.409],
                         [ 745.   ,    0.42 ],
                         [ 750.   ,    0.436],
                         [ 755.   ,    0.45 ],
                         [ 760.   ,    0.462],
                         [ 765.   ,    0.465],
                         [ 770.   ,    0.448],
                         [ 775.   ,    0.432],
                         [ 780.   ,    0.421]],
                        interpolator=SpragueInterpolator,
                        interpolator_args={},
                        extrapolator=Extrapolator,
                        extrapolator_args={...})
```

colour.read_spectral_data_from_csv_file

colour.**read_spectral_data_from_csv_file**(*path*, *delimiter=u', '*, *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 power 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')
>>> spds_data = read_spectral_data_from_csv_file(csv_file)
>>> pprint(list(spds_data.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']
```

colour.write_spds_to_csv_file

colour.**write_spds_to_csv_file**(*spds*, *path*, *delimiter=u', '*, *fields=None*)

Writes the given spectral power distributions to given *CSV* file.

Parameters

- **spds** (dict) – Spectral power 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 power

distributions *dict*.

Returns Definition success.

Return type bool

Raises RuntimeError – If the given spectral power distributions have different shapes.

IES TM-27-14 Data

colour

IES_TM2714_Spd([path, header, . . .])	Defines a *IES TM-27-14* spectral power distribution.

colour.IES_TM2714_Spd

class colour.**IES_TM2714_Spd**(*path=None*, *header=None*, *spectral_quantity=None*, *reflection_geometry=None*, *transmission_geometry=None*, *bandwidth_FWHM=None*, *bandwidth_corrected=None*)

Defines a *IES TM-27-14* spectral power distribution.

This class can read and write *IES TM-27-14* spectral data XML files.

Parameters

- **path** (unicode, optional) – Spectral data XML file path.
- **header** (IES_TM2714_Header, optional) – *IES TM-27-14* spectral power distribution header.
- **spectral_quantity** (unicode, optional) – **{'flux', 'absorptance', 'transmittance', 'reflectance', 'intensity', 'irradiance', 'radiance', 'exitance', 'R-Factor', 'T-Factor', 'relative', 'other'}**, Quantity of measurement for each element of the spectral data.
- **reflection_geometry** (unicode, optional) – **{'di:8', 'de:8', '8:di', '8:de', 'd:d', 'd:0', '45a:0', '45c:0', '0:45a', '45x:0', '0:45x', 'other'}**, Spectral reflectance factors geometric conditions.
- **transmission_geometry** (unicode, optional) – **{'0:0', 'di:0', 'de:0', '0:di', '0:de', 'd:d', 'other'}**, Spectral transmittance factors geometric conditions.
- **bandwidth_FWHM** (numeric, optional) – Spectroradiometer full-width half-maximum bandwidth in nanometers.
- **bandwidth_corrected** (bool, optional) – Specifies if bandwidth correction has been applied to the measured data.

Notes

Reflection Geometry

- di:8: Diffuse / eight-degree, specular component included.
- de:8: Diffuse / eight-degree, specular component excluded.
- 8:di: Eight-degree / diffuse, specular component included.
- 8:de: Eight-degree / diffuse, specular component excluded.

- d:d: Diffuse / diffuse.
- d:0: Alternative diffuse.
- 45a:0: Forty-five degree annular / normal.
- 45c:0: Forty-five degree circumferential / normal.
- 0:45a: Normal / forty-five degree annular.
- 45x:0: Forty-five degree directional / normal.
- 0:45x: Normal / forty-five degree directional.
- other: User-specified in comments.

Transmission Geometry

- 0:0: Normal / normal.
- di:0: Diffuse / normal, regular component included.
- de:0: Diffuse / normal, regular component excluded.
- 0:di: Normal / diffuse, regular component included.
- 0:de: Normal / diffuse, regular component excluded.
- d:d: Diffuse / diffuse.
- other: User-specified in comments.

mapping

path

header

spectral_quantity

reflection_geometry

transmission_geometry

bandwidth_FWHM

bandwidth_corrected

read()

write()

References

- *[IESCCommitteeTM2714WGroup14]*

Examples

```
>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), 'tests', 'resources')
>>> spd = IES_TM2714_Spd(join(directory, 'Fluorescent.spdx'))
>>> spd.read()
True
>>> spd.header.manufacturer
```

```
'Unknown'
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> spd[501.7]
0.0950000...
```

`__init__`(*path=None*, *header=None*, *spectral_quantity=None*, *reflection_geometry=None*, *transmission_geometry=None*, *bandwidth_FWHM=None*, *bandwidth_corrected=None*)

Methods

`__init__`([path, header, spectral_quantity, ...])	
`align`(shape[, interpolator, ...])	Aligns the spectral power distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
`arithmetical_operation`(a, operation[, in_place])	Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.
`clone`()	
`copy`()	Returns a copy of the sub-class instance, must be reimplemented by sub-classes.
`domain_distance`(a)	Returns the euclidean distance between given array and independent domain x closest element.
`extrapolate`(shape[, extrapolator, ...])	Extrapolates the spectral power distribution in-place according to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.
`fill_nan`([method, default])	Fill NaNs in independent domain x variable and corresponding range y variable using given method.
`get`()	
`interpolate`(shape[, interpolator, ...])	Interpolates the spectral power distribution in-place according to *CIE 167:2005* recommendation or given interpolation arguments.
`is_uniform`()	Returns if independent domain x variable is uniform.
`normalise`([factor])	Normalises the spectral power distribution using given normalization factor.
`read`()	Reads and parses the spectral data XML file path.
`signal_unpack_data`([data, domain, dtype])	Unpack given data for continuous signal instantiation.
`to_series`()	Converts the continuous signal to a *Pandas* `Series` class instance.
`trim`(shape)	Trims the spectral power distribution wavelengths to given spectral shape.
`trim_wavelengths`(shape)	
`write`()	Write the spd spectral data to XML file path.
`zeros`()	

X-Rite Data

`colour`

read_spds_from_xrite_file(path)	Reads the spectral data from given *X-Rite* file and returns it as an *OrderedDict* of colour.SpectralPowerDistribution classes.

colour.read_spds_from_xrite_file

colour.**read_spds_from_xrite_file**(*path*)

Reads the spectral data from given *X-Rite* file and returns it as an *OrderedDict* of colour.SpectralPowerDistribution classes.

Parameters **path** (unicode) – Absolute *X-Rite* file path.

Returns colour.SpectralPowerDistribution classes of given *X-Rite* file.

Return type OrderedDict

Notes

- This parser is minimalistic and absolutely not bullet proof.

Examples

```
>>> import os
>>> from pprint import pprint
>>> xrite_file = os.path.join(os.path.dirname(__file__), 'tests',
...                           'resources',
...                           'xrite_digital_colour_checker.txt')
>>> spds_data = read_spds_from_xrite_file(xrite_file)
>>> pprint(list(spds_data.keys()))
['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10']
```

Colour Models

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 *xy* chromaticity coordinates from given *CIE XYZ* tristimulus values.
`xy_to_XYZ`(xy)	Returns the *CIE XYZ* tristimulus values from given *xy* chromaticity coordinates.
`xyY_to_xy`(xyY)	Converts from *CIE xyY* colourspace to *xy* chromaticity coordinates.
`xy_to_xyY`(xy[, Y])	Converts from *xy* chromaticity coordinates to *CIE xyY* colourspace by extending the array last dimension with Y Luminance.

colour.XYZ_to_xyY

`colour.`**`XYZ_to_xyY`**(*XYZ*, *illuminant=array([0.3457, 0.3585])*)

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

- Input *CIE XYZ* tristimulus values are in domain [0, 1].
- Output *CIE xyY* colourspace array is in range [0, 1].

References

- *[Lin03c]*
- *[Wikc]*

Examples

```
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_to_xyY(XYZ)
array([ 0.2641477...,  0.3777000...,  0.1008    ])
```

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

- Input *CIE xyY* colourspace array is in domain [0, 1].
- Output *CIE XYZ* tristimulus values are in range [0, 1].

References

- *[Lin09d]*
- *[Wikc]*

Examples

```
>>> xyY = np.array([0.26414772, 0.37770001, 0.10080000])
>>> xyY_to_XYZ(xyY)
array([ 0.0704953...,  0.1008    ,  0.0955831...])
```

colour.XYZ_to_xy

colour.**XYZ_to_xy**(*XYZ*, *illuminant=array([0.3457, 0.3585])*)
Returns the *xy* chromaticity coordinates from given *CIE XYZ* tristimulus values.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant* chromaticity coordinates.

Returns *xy* chromaticity coordinates.

Return type ndarray

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 1].
- Output *xy* chromaticity coordinates are in range [0, 1].

References

- *[Wikc]*

Examples

```
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_to_xy(XYZ)
array([ 0.2641477...,  0.3777000...])
```

colour.xy_to_XYZ

colour.**xy_to_XYZ**(*xy*)
Returns the *CIE XYZ* tristimulus values from given *xy* chromaticity coordinates.

Parameters **xy** (array_like) – *xy* chromaticity coordinates.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

- Input *xy* chromaticity coordinates are in domain [0, 1].
- Output *CIE XYZ* tristimulus values are in range [0, 1].

References

- *[Wikc]*

Examples

```
>>> xy = np.array([0.26414772, 0.37770001])
>>> xy_to_XYZ(xy)
array([ 0.6993585...,  1.        ,  0.9482453...])
```

colour.xyY_to_xy

`colour.xyY_to_xy(xyY)`

Converts from *CIE xyY* colourspace to *xy* chromaticity coordinates.

`xyY` argument with last dimension being equal to 2 will be assumed to be a *xy* chromaticity coordinates argument and will be returned directly by the definition.

Parameters **xyY** (`array_like`) – *CIE xyY* colourspace array or *xy* chromaticity coordinates.

Returns *xy* chromaticity coordinates.

Return type ndarray

Notes

- Input *CIE xyY* colourspace array is in domain [0, 1].
- Output *xy* chromaticity coordinates are in range [0, 1].

References

- *[Wikc]*

Examples

```
>>> xyY = np.array([0.26414772, 0.37770001, 0.10080000])
>>> xyY_to_xy(xyY)
array([ 0.2641477...,  0.3777000...])
>>> xy = np.array([0.26414772, 0.37770001])
>>> xyY_to_xy(xy)
array([ 0.2641477...,  0.3777000...])
```

colour.xy_to_xyY

colour.**xy_to_xyY**(*xy*, *Y=1*)

Converts from *xy* chromaticity coordinates to *CIE xyY* colourspace by extending the array last dimension with Y Luminance.

xy argument with last dimension being equal to 3 will be assumed to be a *CIE xyY* colourspace array argument and will be returned directly by the definition.

Parameters

- **xy** (array_like) – *xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y** (numeric, optional) – Optional Y Luminance value used to construct the *CIE xyY* colourspace array, otherwise the Y Luminance will be set to 1.

Returns *CIE xyY* colourspace array.

Return type ndarray

Notes

- 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().
- Input *xy* chromaticity coordinates are in domain [0, 1].
- Output *CIE xyY* colourspace array is in range [0, 1].

References

- *[Wikc]*

Examples

```
>>> xy = np.array([0.26414772, 0.37770001])
>>> xy_to_xyY(xy)
array([ 0.2641477...,  0.3777000...,  1.        ])
>>> xy = np.array([0.26414772, 0.37770001, 0.10080000])
>>> xy_to_xyY(xy)
array([ 0.2641477...,  0.3777000...,  0.1008...])
>>> xy = np.array([0.26414772, 0.37770001])
>>> xy_to_xyY(xy, 100)
array([   0.2641477...,    0.3777000...,  100.        ])
```

CIE L*a*b* Colourspace

colour

XYZ_to_Lab(XYZ[, illuminant])	Converts from *CIE XYZ* tristimulus values to *CIE L*a*b** colourspace.

Continued on next page

Table 3.125 – continued from previous page

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([0.3457, 0.3585])*)

Converts from *CIE XYZ* tristimulus values to *CIE L*a*b** colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE L*a*b** colourspace array.

Return type ndarray

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 1].
- Input *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Output *Lightness* L^* is in range [0, 100].

References

- *[CIET14804f]*

Examples

```
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_to_Lab(XYZ)
array([ 37.9856291..., -23.6290768...,  -4.4174661...])
```

colour.Lab_to_XYZ

colour.**Lab_to_XYZ**(*Lab*, *illuminant=array([0.3457, 0.3585])*)

Converts from *CIE L*a*b** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Lab** (array_like) – *CIE L*a*b** colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

- Input *Lightness* L^* is in domain [0, 100].
- Input *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Output *CIE XYZ* tristimulus values are in range [0, 1].

References

- *[CIET14804f]*

Examples

```
>>> Lab = np.array([37.98562910, -23.62907688, -4.41746615])
>>> Lab_to_XYZ(Lab)
array([ 0.0704953...,  0.1008    ,  0.0955831...])
```

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

- *Lightness* L^* is in domain [0, 100].

References

- *[CIET14804f]*

Examples

```
>>> Lab = np.array([37.98562910, -23.62907688, -4.41746615])
>>> Lab_to_LCHab(Lab)
array([  37.9856291...,   24.0384542...,  190.5892337...])
```

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

- *Lightness* L^* is in domain [0, 100].

References

- *[CIET14804f]*

Examples

```
>>> LCHab = np.array([37.98562910, 24.03845422, 190.58923377])
>>> LCHab_to_Lab(LCHab)
array([ 37.9856291..., -23.6290768...,  -4.4174661...])
```

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.
Luv_uv_to_xy(uv)	Returns the *xy* chromaticity coordinates from given *CIE L*u*v** colourspace uv^p chromaticity coordinates.

colour.XYZ_to_Luv

colour.**XYZ_to_Luv**(*XYZ, illuminant=array([0.3457, 0.3585])*)

Converts from *CIE XYZ* tristimulus values to *CIE L*u*v** colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.

- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE L*u*v** colourspace array.

Return type ndarray

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 1].
- Input *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Output L^* is in range [0, 100].

References

- *[CIET14804f]*
- *[Wikg]*

Examples

```
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_to_Luv(XYZ)
array([ 37.9856291..., -28.8021959...,  -1.3580070...])
```

colour.Luv_to_XYZ

colour.**Luv_to_XYZ**(*Luv, illuminant=array([0.3457, 0.3585])*)

Converts from *CIE L*u*v** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Luv** (array_like) – *CIE L*u*v** colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

- Input L^* is in domain [0, 100].
- Input *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Output *CIE XYZ* tristimulus values are in range [0, 1].

References

- *[CIET14804f]*
- *[Wikg]*

Examples

```
>>> Luv = np.array([37.9856291 , -28.80219593,  -1.35800706])
>>> Luv_to_XYZ(Luv)
array([ 0.0704953...,  0.1008    ,  0.0955831...])
```

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

- Input / output L^* is in domain / range [0, 100].

References

- *[CIET14804f]*

Examples

```
>>> Luv = np.array([37.9856291 , -28.80219593,  -1.35800706])
>>> Luv_to_LCHuv(Luv)
array([  37.9856291...,   28.8341927...,  182.6994640...])
```

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

- Input / output L^* is in domain / range [0, 100].

References

- *[CIET14804f]*

Examples

```
>>> LCHuv = np.array([37.98562910, 28.83419279, 182.69946404])
>>> LCHuv_to_Luv(LCHuv)
array([ 37.9856291..., -28.8021959...,  -1.3580070...])
```

colour.Luv_to_uv

colour.**Luv_to_uv**(*Luv, illuminant=array([0.3457, 0.3585])*)
Returns the uv^p chromaticity coordinates from given *CIE L*u*v** colourspace array.

Parameters

- **Luv** (array_like) – *CIE L*u*v** colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns uv^p chromaticity coordinates.

Return type ndarray

Notes

- Input L^* is in domain [0, 100].
- Input *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Output uv^p chromaticity coordinates are in range [0, 1].

References

- *[CIET14804e]*

Examples

```
>>> Luv = np.array([37.9856291 , -28.80219593,  -1.35800706])
>>> Luv_to_uv(Luv)
array([ 0.1508531...,  0.4853297...])
```

colour.Luv_uv_to_xy

colour.**Luv_uv_to_xy**(*uv*)

Returns the *xy* chromaticity coordinates from given *CIE L*u*v** colourspace uv^p chromaticity coordinates.

Parameters **uv** (array_like) – *CIE L*u*v* u"v"* chromaticity coordinates.

Returns *xy* chromaticity coordinates.

Return type ndarray

Notes

- Input uv^p chromaticity coordinates are in domain [0, 1].
- Output *xy* is in range [0, 1].

References

- *[Wikx]*

Examples

```
>>> uv = np.array([0.150853098829857, 0.485329708543180])
>>> Luv_uv_to_xy(uv)
array([ 0.2641477...,  0.3777000...])
```

CIE UCS Colourspace

colour

XYZ_to_UCS(XYZ)	Converts from *CIE XYZ* tristimulus values to *CIE UCS* colourspace.
UCS_to_XYZ(UVW)	Converts from *CIE UCS* colourspace to *CIE XYZ* tristimulus values.
UCS_to_uv(UVW)	Returns the *uv* chromaticity coordinates from given *CIE UCS* colourspace array.
UCS_uv_to_xy(uv)	Returns the *xy* chromaticity coordinates from given *CIE UCS* colourspace *uv* chromaticity coordinates.

colour.XYZ_to_UCS

colour.**XYZ_to_UCS**(*XYZ*)

Converts from *CIE XYZ* tristimulus values to *CIE UCS* colourspace.

Parameters **XYZ** (array_like) – *CIE XYZ* tristimulus values.

Returns *CIE UCS* colourspace array.

Return type ndarray

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 1].
- Output *CIE UCS* colourspace array is in range [0, 1].

References

- *[Wikv]*
- *[Wikd]*

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_to_UCS(XYZ)
array([ 0.0469968...,  0.1008    ,  0.1637439...])
```

colour.UCS_to_XYZ

colour.**UCS_to_XYZ**(*UVW*)

Converts from *CIE UCS* colourspace to *CIE XYZ* tristimulus values.

Parameters **UVW** (array_like) – *CIE UCS* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

- Input *CIE UCS* colourspace array is in domain [0, 1].
- Output *CIE XYZ* tristimulus values are in range [0, 1].

References

- *[Wikv]*
- *[Wikd]*

Examples

```
>>> import numpy as np
>>> UVW = np.array([0.04699689, 0.10080000, 0.16374390])
>>> UCS_to_XYZ(UVW)
array([ 0.0704953...,  0.1008    ,  0.0955831...])
```

colour.UCS_to_uv

colour.**UCS_to_uv**(*UVW*)

Returns the *uv* chromaticity coordinates from given *CIE UCS* colourspace array.

Parameters **UVW** (array_like) – *CIE UCS* colourspace array.

Returns *uv* chromaticity coordinates.

Return type ndarray

Notes

- Input *CIE UCS* colourspace array is in domain [0, 1].
- Output *uv* chromaticity coordinates are in range [0, 1].

References

- *[Wikv]*

Examples

```
>>> import numpy as np
>>> UCS = np.array([0.04699689, 0.10080000, 0.16374390])
>>> UCS_to_uv(UCS)
array([ 0.1508530...,  0.3235531...])
```

colour.UCS_uv_to_xy

colour.**UCS_uv_to_xy**(*uv*)

Returns the *xy* chromaticity coordinates from given *CIE UCS* colourspace *uv* chromaticity coordinates.

Parameters **uv** (array_like) – *CIE UCS uv* chromaticity coordinates.

Returns *xy* chromaticity coordinates.

Return type ndarray

Notes

- Input *uv* chromaticity coordinates are in domain [0, 1].
- Output *xy* chromaticity coordinates are in range [0, 1].

References

- *[Wikv]*

Examples

```
>>> import numpy as np
>>> uv = np.array([0.150853087327666, 0.323553137295440])
>>> UCS_uv_to_xy(uv)
array([ 0.2641477...,  0.3777000...])
```

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.

colour.XYZ_to_UVW

colour.**XYZ_to_UVW**(*XYZ*, *illuminant=array([0.3457, 0.3585])*)

Converts from *CIE XYZ* tristimulus values to *CIE 1964 U*V*W** colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE 1964 U*V*W** colourspace array.

Return type ndarray

Warning: The input domain and output range of that definition are non standard!

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].
- Input *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Output *CIE 1964 U*V*W** colourspace array is in range [0, 100].

References

- *[Wike]*

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313]) * 100
>>> XYZ_to_UVW(XYZ)
array([-28.0579733...,  -0.8819449...,  37.0041149...])
```

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([96.38, 100., 82.45])*, *K_ab=array([173.51, 58.48])*)

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

- Input *CIE XYZ* and reference *illuminant* tristimulus values are in domain [0, 100].
- Output *Lightness* L^* is in range [0, 100].

References

- *[Hun08a]*

Examples

```
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313]) * 100
>>> D50 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D50']
>>> XYZ_to_Hunter_Lab(XYZ, D50.XYZ_n, D50.K_ab)
array([ 31.7490157..., -15.1146262...,  -2.7866075...])
```

colour.Hunter_Lab_to_XYZ

colour.**Hunter_Lab_to_XYZ**(*Lab, XYZ_n=array([96.38, 100., 82.45]), K_ab=array([173.51, 58.48])*)

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

- Input *Lightness* L^* is in domain [0, 100].
- Input *CIE XYZ* and reference *illuminant* tristimulus values are in domain [0, 100].
- Output *CIE XYZ* tristimulus values are in range [0, 100].

References

- *[Hun08a]*

Examples

```
>>> Lab = np.array([31.74901573, -15.11462629, -2.78660758])
>>> D50 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D50']
>>> Hunter_Lab_to_XYZ(Lab, D50.XYZ_n, D50.K_ab)
array([  7.049534,  10.08    ,   9.558313])
```

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

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 100].

References

- *[Hun08b]*

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.

colour.XYZ_to_Hunter_Rdab

colour.**XYZ_to_Hunter_Rdab**(*XYZ*, *XYZ_n=array([96.38, 100., 82.45])*, *K_ab=array([173.51, 58.48])*)
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

- Input *CIE XYZ* and reference *illuminant* tristimulus values are in domain [0, 100].

References

- *[Hun12]*

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313]) * 100
>>> D50 = HUNTERLAB_ILLUMINANTS[
```

```
...     'CIE 1931 2 Degree Standard Observer']['D50']
>>> XYZ_to_Hunter_Rdab(XYZ, D50.XYZ_n, D50.K_ab)
...
array([ 10.08      , -18.6765376...,  -3.4432992...])
```

Luo, Cui and Li (2006)

`colour`

JMh_CIECAM02_to_CAM02LCD(JMh)	Converts from *CIECAM02* JMh correlates array to *Luo et alii (2006) CAM02-LCD* colourspace $J'a'b'$ array.
CAM02LCD_to_JMh_CIECAM02(Jpapbp)	Converts from *Luo et alii (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 alii (2006) CAM02-SCD* colourspace $J'a'b'$ array.
CAM02SCD_to_JMh_CIECAM02(Jpapbp)	Converts from *Luo et alii (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 alii (2006) CAM02-UCS* colourspace $J'a'b'$ array.
CAM02UCS_to_JMh_CIECAM02(Jpapbp)	Converts from *Luo et alii (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 alii (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CIECAM02* correlates array JMh.

Returns *Luo et alii (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Return type ndarray

References

- *[LCL06]*

Examples

```
>>> from colour.appearance import (
...     CIECAM02_VIEWING_CONDITIONS,
...     XYZ_to_CIECAM02)
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
```

```
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = XYZ_to_CIECAM02(
...     XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02LCD(JMh)
array([ 54.9043313...,  -0.0845039...,  -0.0685483...])
```

colour.CAM02LCD_to_JMh_CIECAM02

colour.**CAM02LCD_to_JMh_CIECAM02**(*Jpapbp*)

Converts from *Luo et alii (2006) CAM02-LCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Luo et alii (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array JMh.

Return type ndarray

References

- *[LCL06]*

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 alii (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CIECAM02* correlates array JMh.

Returns *Luo et alii (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Return type ndarray

References

- *[LCL06]*

Examples

```
>>> from colour.appearance import (
...     CIECAM02_VIEWING_CONDITIONS,
...     XYZ_to_CIECAM02)
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = XYZ_to_CIECAM02(
...     XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02SCD(JMh)
array([ 54.9043313...,  -0.0843617...,  -0.0684329...])
```

colour.CAM02SCD_to_JMh_CIECAM02

colour.**CAM02SCD_to_JMh_CIECAM02**(*Jpapbp*)

Converts from *Luo et alii (2006) CAM02-SCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Luo et alii (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array JMh.

Return type ndarray

References

- *[LCL06]*

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 alii (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CIECAM02* correlates array JMh.

Returns *Luo et alii (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Return type ndarray

References

- *[LCL06]*

Examples

```
>>> from colour.appearance import (
...     CIECAM02_VIEWING_CONDITIONS,
...     XYZ_to_CIECAM02)
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = XYZ_to_CIECAM02(
...     XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02UCS(JMh)
array([ 54.9043313...,  -0.0844236...,  -0.0684831...])
```

colour.CAM02UCS_to_JMh_CIECAM02

colour.**CAM02UCS_to_JMh_CIECAM02**(*Jpapbp*)

Converts from *Luo et alii (2006) CAM02-UCS* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Luo et alii (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array JMh.

Return type ndarray

References

- *[LCL06]*

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])
```

Li, Li, Wang, Zu, Luo, Cui, Melgosa, Brill and Pointer (2017)

colour

JMh_CAM16_to_CAM16LCD	Converts from *CAM16* JMh correlates array to *Li et alii (2017) CAM16-LCD* colourspace $J'a'b'$ array.
CAM16LCD_to_JMh_CAM16	Converts from *Li et alii (2017) CAM16-LCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.
JMh_CAM16_to_CAM16SCD	Converts from *CAM16* JMh correlates array to *Li et alii (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Continued on next page

Table 3.132 – continued from previous page

CAM16SCD_to_JMh_CAM16	Converts from *Li et alii (2017) CAM16-SCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.
JMh_CAM16_to_CAM16UCS	Converts from *CAM16* JMh correlates array to *Li et alii (2017) CAM16-UCS* colourspace $J'a'b'$ array.
CAM16UCS_to_JMh_CAM16	Converts from *Li et alii (2017) CAM16-UCS* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

colour.JMh_CAM16_to_CAM16LCD

colour.**JMh_CAM16_to_CAM16LCD = <functools.partial object>**

Converts from *CAM16* JMh correlates array to *Li et alii (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CAM16* correlates array JMh.

Returns *Li et alii (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Return type ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to colour.JMh_CIECAM02_to_CAM02LCD() definition for an usage example.

colour.CAM16LCD_to_JMh_CAM16

colour.**CAM16LCD_to_JMh_CAM16 = <functools.partial object>**

Converts from *Li et alii (2017) CAM16-LCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Li et alii (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Returns *CAM16* correlates array JMh.

Return type ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to colour.CAM02LCD_to_JMh_CIECAM02() definition for an usage example.

colour.JMh_CAM16_to_CAM16SCD

colour.**JMh_CAM16_to_CAM16SCD = <functools.partial object>**

Converts from *CAM16* JMh correlates array to *Li et alii (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CAM16* correlates array JMh.

Returns *Li et alii (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Return type ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to `colour.JMh_CIECAM02_to_CAM02SCD()` definition for an usage example.

colour.CAM16SCD_to_JMh_CAM16

colour.**CAM16SCD_to_JMh_CAM16 = <functools.partial object>**

Converts from *Li et alii (2017) CAM16-SCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Li et alii (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Returns *CAM16* correlates array JMh.

Return type ndarray

References

- *[LLW+17]*

Notes

- This docstring is automatically generated, please refer to `colour.CAM02SCD_to_JMh_CIECAM02()` definition for an usage example.

colour.JMh_CAM16_to_CAM16UCS

colour.**JMh_CAM16_to_CAM16UCS = <functools.partial object>**

Converts from *CAM16* JMh correlates array to *Li et alii (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CAM16* correlates array JMh.

Returns *Li et alii (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Return type ndarray

References

- [LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.JMh_CIECAM02_to_CAM02UCS()` definition for an usage example.

colour.CAM16UCS_to_JMh_CAM16

`colour.CAM16UCS_to_JMh_CAM16 = <functools.partial object>`

Converts from *Li et alii (2017) CAM16-UCS* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Li et alii (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Returns *CAM16* correlates array JMh.

Return type ndarray

References

- [LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.CAM02UCS_to_JMh_CIECAM02()` definition for an usage example.

IPT Colourspace

colour

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

- Input *CIE XYZ* tristimulus values needs to be adapted for *CIE Standard Illuminant D Series D65*.

References

- [Fai13d]

Examples

```
>>> XYZ = np.array([0.96907232, 1.00000000, 1.12179215])
>>> XYZ_to_IPT(XYZ)
array([ 1.0030082...,  0.0190691..., -0.0136929...])
```

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

References

- [Fai13d]

Examples

```
>>> IPT = np.array([1.00300825, 0.01906918, -0.01369292])
>>> IPT_to_XYZ(IPT)
array([ 0.9690723...,  1.        ,  1.1217921...])
```

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

References

- [Fai13d]

Examples

```
>>> IPT = np.array([0.96907232, 1, 1.12179215])
>>> IPT_hue_angle(IPT)
48.2852074...
```

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=u'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 in domain [0, 1].
- **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

- Input *CIE XYZ* tristimulus values needs to be adapted for *CIE Standard Illuminant D Series D65*.

References

- *[FW10]*
- *[FC11]*

Examples

```
>>> XYZ = np.array([0.96907232, 1.00000000, 1.12179215])
>>> XYZ_to_hdr_IPT(XYZ)
array([ 93.5317473...,   1.8564156...,  -1.3292254...])
>>> XYZ_to_hdr_IPT(XYZ, method='Fairchild 2010')
array([ 94.6592917...,   0.3804177...,  -0.2673118...])
```

colour.hdr_IPT_to_XYZ

`colour.`**`hdr_IPT_to_XYZ`**(*IPT_hdr*, *Y_s=0.2*, *Y_abs=100*, *method=u'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 in domain [0, 1].
- **`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

References

- *[FW10]*
- *[FC11]*

Examples

```
>>> IPT_hdr = np.array([93.53174734, 1.85641567, -1.32922546])
>>> hdr_IPT_to_XYZ(IPT_hdr)
array([ 0.9690723...,  1.        ,  1.1217921...])
>>> IPT_hdr = np.array([94.65929175, 0.38041773, -0.26731187])
>>> hdr_IPT_to_XYZ(IPT_hdr, method='Fairchild 2010')
...
array([ 0.9690723...,  1.        ,  1.1217921...])
```

colour.HDR_IPT_METHODS

`colour.`**`HDR_IPT_METHODS = (u'Fairchild 2010', u'Fairchild 2011')`**
Supported *hdr-IPT* colourspace computation methods.

References

- *[FW10]*
- *[FC11]*

HDR_IPT_METHODS [tuple] **{'Fairchild 2011', 'Fairchild 2010'}**

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([0.3457, 0.3585]), Y_s=0.2, Y_abs=100, method=u'Fairchild 2011'*)

Converts from *CIE XYZ* tristimulus values to *hdr-CIELAB* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround in domain [0, 1].
- **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

- 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.
- Input *CIE XYZ* tristimulus values are in domain [0, math:*infty*].
- Input *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].

References

- *[FW10]*
- *[FC11]*

Examples

```
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_to_hdr_CIELab(XYZ)
array([ 26.4646106..., -24.613326 ...,  -4.8479681...])
>>> XYZ_to_hdr_CIELab(XYZ, method='Fairchild 2010')
array([ 24.9020664..., -46.8312760..., -10.1427484...])
```

colour.hdr_CIELab_to_XYZ

colour.**hdr_CIELab_to_XYZ**(*Lab_hdr, illuminant=array([0.3457, 0.3585]), Y_s=0.2, Y_abs=100, method=u'Fairchild 2011'*)

Converts from *hdr-CIELAB* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Lab_hdr** (array_like) – *hdr-CIELAB* colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround in domain [0, 1].
- **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

- Input *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Output *CIE XYZ* tristimulus values are in range [0, math:*infty*].

References

- *[FW10]*
- *[FC11]*

Examples

```
>>> Lab_hdr = np.array([26.46461067, -24.613326, -4.84796811])
>>> hdr_CIELab_to_XYZ(Lab_hdr)
array([ 0.0704953...,  0.1008    ,  0.0955831...])
>>> Lab_hdr = np.array([24.90206646, -46.83127607, -10.14274843])
>>> hdr_CIELab_to_XYZ(Lab_hdr, method='Fairchild 2010')
...
array([ 0.0704953...,  0.1008    ,  0.0955831...])
```

colour.HDR_CIELAB_METHODS

colour.**HDR_CIELAB_METHODS = (u'Fairchild 2010', u'Fairchild 2011')**
Supported *hdr-CIELAB* colourspace computation methods.

References

- *[FW10]*
- *[FC11]*

HDR_CIELAB_METHODS [tuple] **{'Fairchild 2011', 'Fairchild 2010'}**

RGB Colourspace and Transformations

colour

XYZ_to_RGB(XYZ, illuminant_XYZ, . . . [, . . .])	Converts from *CIE XYZ* tristimulus values to given *RGB* colourspace.
RGB_to_XYZ(RGB, illuminant_RGB, . . . [, . . .])	Converts from given *RGB* colourspace to *CIE XYZ* tristimulus values.
RGB_to_RGB(RGB, input_colourspace, . . . [, . . .])	Converts from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.
RGB_to_RGB_matrix(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, XYZ_to_RGB_matrix, chromatic_adaptation_transform=u'CAT02', encoding_cctf=None*)
Converts from *CIE XYZ* tristimulus values to given *RGB* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant_XYZ** (array_like) – *CIE XYZ* tristimulus values *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

- **illuminant_RGB** (array_like) – *RGB* colourspace *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **XYZ_to_RGB_matrix** (array_like) – *Normalised primary matrix*.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.
- **encoding_cctf** (object, optional) – Encoding colour component transfer function (Encoding CCTF) or opto-electronic transfer function (OETF / OECF).

Returns *RGB* colourspace array.

Return type ndarray

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 1].
- Input *illuminant_XYZ xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Input *illuminant_RGB xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Output *RGB* colourspace array is in range [0, 1].

Examples

```
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> illuminant_XYZ = np.array([0.34570, 0.35850])
>>> illuminant_RGB = np.array([0.31270, 0.32900])
>>> chromatic_adaptation_transform = 'Bradford'
>>> XYZ_to_RGB_matrix = np.array(
...     [[3.24062548, -1.53720797, -0.49862860],
...      [-0.96893071, 1.87575606, 0.04151752],
...      [0.05571012, -0.20402105, 1.05699594]]
... )
>>> XYZ_to_RGB(XYZ, illuminant_XYZ, illuminant_RGB, XYZ_to_RGB_matrix,
...            chromatic_adaptation_transform)
array([ 0.0110015...,  0.1273504...,  0.1163271...])
```

colour.RGB_to_XYZ

colour.**RGB_to_XYZ**(*RGB*, *illuminant_RGB*, *illuminant_XYZ*, *RGB_to_XYZ_matrix*, *chromatic_adaptation_transform=u'CAT02'*, *decoding_cctf=None*)

Converts from given *RGB* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **illuminant_RGB** (array_like) – *RGB* colourspace *illuminant* chromaticity coordinates or *CIE xyY* colourspace array.

- **illuminant_XYZ** (array_like) – *CIE XYZ* tristimulus values *illuminant* chromaticity coordinates or *CIE xyY* colourspace array.
- **RGB_to_XYZ_matrix** (array_like) – *Normalised primary matrix*.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.
- **decoding_cctf** (object, optional) – Decoding colour component transfer function (Decoding CCTF) or electro-optical transfer function (EOTF / EOCF).

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

- Input *RGB* colourspace array is in domain [0, 1].
- Input *illuminant_RGB xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Input *illuminant_XYZ xy* chromaticity coordinates or *CIE xyY* colourspace array are in domain [0, ∞].
- Output *CIE XYZ* tristimulus values are in range [0, 1].

Examples

```
>>> RGB = np.array([0.01100154,  0.12735048,  0.11632713])
>>> illuminant_RGB = np.array([0.31270, 0.32900])
>>> illuminant_XYZ = np.array([0.34570, 0.35850])
>>> chromatic_adaptation_transform = 'Bradford'
>>> RGB_to_XYZ_matrix = np.array(
...     [[0.41240000, 0.35760000, 0.18050000],
...      [0.21260000, 0.71520000, 0.07220000],
...      [0.01930000, 0.11920000, 0.95050000]]
... )
>>> RGB_to_XYZ(RGB, illuminant_RGB, illuminant_XYZ, RGB_to_XYZ_matrix,
...            chromatic_adaptation_transform)
array([ 0.0704953...,  0.1008    ,  0.0955831...])
```

colour.RGB_to_RGB

colour.**RGB_to_RGB**(*RGB, input_colourspace, output_colourspace, chromatic_adaptation_transform=u'CAT02', apply_decoding_cctf=False, apply_encoding_cctf=False*)

Converts from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **input_colourspace** (RGB_Colourspace) – *RGB* input colourspace.

- **output_colourspace** (RGB_Colourspace) – *RGB* output colourspace.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.
- **apply_decoding_cctf** (bool, optional) – Apply input colourspace decoding colour component transfer function / electro-optical transfer function.
- **apply_encoding_cctf** (bool, optional) – Apply output colourspace encoding colour component transfer function / opto-electronic transfer function.

Returns *RGB* colourspace array.

Return type ndarray

Notes

- Input / output *RGB* colourspace arrays are in domain / range [0, 1].
- Input / output *RGB* colourspace arrays are assumed to be representing linear light values.

Examples

```
>>> from colour.models import sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE
>>> RGB = np.array([0.01103742, 0.12734226, 0.11632971])
>>> RGB_to_RGB(RGB, sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE)
...
array([ 0.0643561...,  0.1157331...,  0.1158069...])
```

colour.RGB_to_RGB_matrix

colour.**RGB_to_RGB_matrix**(*input_colourspace*, *output_colourspace*, *chromatic_adaptation_transform=u'CAT02'*)

Computes the matrix M converting from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.

Parameters

- **input_colourspace** (RGB_Colourspace) – *RGB* input colourspace.
- **output_colourspace** (RGB_Colourspace) – *RGB* output colourspace.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.

Returns Conversion matrix M.

Return type ndarray

Examples

```
>>> from colour.models import sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE
>>> RGB_to_RGB_matrix(sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE)
...
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=u'CAT02', apply_encoding_cctf=True*)

Converts from *CIE XYZ* tristimulus values to *sRGB* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Source illuminant chromaticity coordinates.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.
- **apply_encoding_cctf** (bool, optional) – Apply *sRGB* encoding colour component transfer function / opto-electronic transfer function.

Returns *sRGB* colour array.

Return type ndarray

Notes

- Input *CIE XYZ* tristimulus values are in domain [0, 1].

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> XYZ_to_sRGB(XYZ)
array([ 0.1749881...,  0.3881947...,  0.3216031...])
```

colour.sRGB_to_XYZ

colour.**sRGB_to_XYZ**(*RGB, illuminant=array([0.3127, 0.329]), chromatic_adaptation_method=u'CAT02', apply_decoding_cctf=True*)

Converts from *sRGB* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **RGB** (array_like) – *sRGB* colourspace array.
- **illuminant** (array_like, optional) – Source illuminant chromaticity coordinates.
- **chromatic_adaptation_method** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* method.
- **apply_decoding_cctf** (bool, optional) – Apply *sRGB* decoding colour component transfer function / electro-optical transfer function.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

- Input *RGB* colourspace array is in domain [0, 1].

Examples

```
>>> import numpy as np
>>> RGB = np.array([0.17498172, 0.38818743, 0.32159978])
>>> sRGB_to_XYZ(RGB)
array([ 0.0704953...,  0.1008...,  0.0955831...])
```

RGB Colourspace Derivation

colour

normalised_primary_matrix(primaries, whitepoint)	Returns the *normalised primary matrix* 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)	Returns the *primaries* and *whitepoint* xy chromaticity coordinates using given *normalised primary matrix*.
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*)

Returns the *normalised primary matrix* using given *primaries* and *whitepoint* xy chromaticity coordinates.

Parameters

- **`primaries`** (`array_like, (3, 2)`) – Primaries xy chromaticity coordinates.
- **`whitepoint`** (`array_like`) – Illuminant / whitepoint xy chromaticity coordinates.

Returns *Normalised primary matrix.*

Return type ndarray, (3, 3)

References

- *[SocietyoMPaTEngineers93]*

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=u'CAT02'*)

Chromatically adapts given *primaries* xy chromaticity coordinates from test `whitepoint_t` to reference `whitepoint_r`.

Parameters

- **`primaries`** (`array_like, (3, 2)`) – Primaries xy chromaticity coordinates.
- **`whitepoint_t`** (`array_like`) – Test illuminant / whitepoint xy chromaticity coordinates.
- **`whitepoint_r`** (`array_like`) – Reference illuminant / whitepoint xy chromaticity coordinates.
- **`chromatic_adaptation_transform`** (`unicode, optional`) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.

Returns Chromatically adapted primaries xy chromaticity coordinates.

Return type ndarray

Examples

```
>>> p = np.array([0.64, 0.33, 0.30, 0.60, 0.15, 0.06])
>>> w_t = np.array([0.31270, 0.32900])
>>> w_r = np.array([0.34570, 0.35850])
>>> chromatic_adaptation_transform = 'Bradford'
>>> chromatically_adapted_primaries(p, w_t, w_r,
...                                 chromatic_adaptation_transform)
...
array([[ 0.6484414...,  0.3308533...],
       [ 0.3211951...,  0.5978443...],
       [ 0.1558932...,  0.0660492...]])
```

colour.primaries_whitepoint

colour.**primaries_whitepoint**(*npm*)

Returns the *primaries* and *whitepoint* xy chromaticity coordinates using given *normalised primary matrix*.

Parameters **npm** (array_like, (3, 3)) – *Normalised primary matrix*.

Returns *Primaries* and *whitepoint* xy chromaticity coordinates.

Return type tuple

References

- *[Tri15]*

Examples

```
>>> npm = np.array([[9.52552396e-01, 0.00000000e+00, 9.36786317e-05],
...                 [3.43966450e-01, 7.28166097e-01, -7.21325464e-02],
...                 [0.00000000e+00, 0.00000000e+00, 1.00882518e+00]])
>>> p, w = 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([40.6, 4.2, 67.4])
>>> 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)
12.1616018...
```

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 dataset from colour.models.dataset.aces_rgb, etc....
RGB_COLOURSPACES	Aggregated *RGB* colourspaces.

colour.RGB_Colourspace

class colour.**RGB_Colourspace**(*name*, *primaries*, *whitepoint*, *illuminant=None*, *RGB_to_XYZ_matrix=None*, *XYZ_to_RGB_matrix=None*, *encoding_cctf=None*, *decoding_cctf=None*, *use_derived_RGB_to_XYZ_matrix=False*, *use_derived_XYZ_to_RGB_matrix=False*)

Implements support for the *RGB* colourspaces dataset from `colour.models.dataset.aces_rgb`, etc....

Colour science literature related to *RGB* colourspaces and encodings defines their dataset using different degree of precision or rounding. While instances where a whitepoint is being defined with a value different than its canonical agreed one are rare, it is however very common to have normalised primary matrices rounded at different decimals. This can yield large discrepancies in computations.

Such an occurrence is the *V-Gamut* colourspace white paper, that defines the *V-Gamut* to *ITU-R BT.709* conversion matrix as follows:

```
[[ 1.806576 -0.695697 -0.110879]
 [-0.170090  1.305955 -0.135865]
 [-0.025206 -0.154468  1.179674]]
```

Computing this matrix using *ITU-R BT.709* colourspace derived normalised primary matrix yields:

```
[[ 1.8065736 -0.6956981 -0.1108786]
 [-0.1700890  1.3059548 -0.1358648]
 [-0.0252057 -0.1544678  1.1796737]]
```

The latter matrix is almost equals with the former, however performing the same computation using *IEC 61966-2-1:1999 sRGB* colourspace normalised primary matrix introduces severe disparities:

```
[[ 1.8063853 -0.6956147 -0.1109453]
 [-0.1699311  1.3058387 -0.1358616]
 [-0.0251630 -0.1544899  1.1797117]]
```

In order to provide support for both literature defined dataset and accurate computations enabling transformations without loss of precision, the `colour.RGB_Colourspace` class provides two sets of transformation matrices:

- Instantiation transformation matrices
- Derived transformation matrices

Upon instantiation, the `colour.RGB_Colourspace` class stores the given `RGB_to_XYZ_matrix` and `XYZ_to_RGB_matrix` arguments and also computes their derived counterpart using the `primaries` and `whitepoint` arguments.

Whether the initialisation or derived matrices are used in subsequent computations is dependent on the `colour.RGB_Colourspace.use_derived_RGB_to_XYZ_matrix` and `colour.RGB_Colourspace.use_derived_XYZ_to_RGB_matrix` attributes values.

Parameters

- **name** (`unicode`) – *RGB* colourspace name.
- **primaries** (`array_like`) – *RGB* colourspace primaries.
- **whitepoint** (`array_like`) – *RGB* colourspace whitepoint.
- **illuminant** (`unicode, optional`) – *RGB* colourspace whitepoint name as illuminant.

- **RGB_to_XYZ_matrix** (array_like, optional) – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.
- **XYZ_to_RGB_matrix** (array_like, optional) – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.
- **encoding_cctf** (object, optional) – Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF) that maps estimated tristimulus values in a scene to $R'G'B'$ video component signal value.
- **decoding_cctf** (object, optional) – Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF) that maps an $R'G'B'$ video component signal value to tristimulus values at the display.
- **use_derived_RGB_to_XYZ_matrix** (bool, optional) – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.
- **use_derived_XYZ_to_RGB_matrix** (bool, optional) – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

name

primaries

whitepoint

illuminant

RGB_to_XYZ_matrix

XYZ_to_RGB_matrix

encoding_cctf

decoding_cctf

use_derived_RGB_to_XYZ_matrix

use_derived_XYZ_to_RGB_matrix

__str__()

__repr__()

use_derived_transformation_matrices()

Notes

- The normalised primary matrix defined by colour.RGB_Colourspace.RGB_to_XYZ_matrix attribute is treated as the prime matrix from which the inverse will be calculated as required by the internal derivation mechanism. This behaviour has been chosen in accordance with literature where commonly a *RGB* colourspace is defined by its normalised primary matrix as it is directly computed from the chosen primaries and whitepoint.

References

- *[InternationalECommission99]*
- *[Pan14]*

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> RGB_to_XYZ_matrix = np.identity(3)
>>> XYZ_to_RGB_matrix = np.identity(3)
>>> colourspace = RGB_Colourspace('RGB Colourspace', p, whitepoint, 'D60',
...                               RGB_to_XYZ_matrix, XYZ_to_RGB_matrix)
>>> colourspace.RGB_to_XYZ_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.XYZ_to_RGB_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_transformation_matrices(True)
True
>>> colourspace.RGB_to_XYZ_matrix
array([[  9.5255239...e-01,   0.0000000...e+00,   9.3678631...e-05],
       [  3.4396645...e-01,   7.2816609...e-01,  -7.2132546...e-02],
       [  0.0000000...e+00,   0.0000000...e+00,   1.0088251...e+00]])
>>> colourspace.XYZ_to_RGB_matrix
array([[  1.0498110...e+00,   0.0000000...e+00,  -9.7484540...e-05],
       [ -4.9590302...e-01,   1.3733130...e+00,   9.8240036...e-02],
       [  0.0000000...e+00,   0.0000000...e+00,   9.9125201...e-01]])
>>> colourspace.use_derived_RGB_to_XYZ_matrix = False
>>> colourspace.RGB_to_XYZ_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_XYZ_to_RGB_matrix = False
>>> colourspace.XYZ_to_RGB_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

__init__(*name*, *primaries*, *whitepoint*, *illuminant=None*, *RGB_to_XYZ_matrix=None*, *XYZ_to_RGB_matrix=None*, *encoding_cctf=None*, *decoding_cctf=None*, *use_derived_RGB_to_XYZ_matrix=False*, *use_derived_XYZ_to_RGB_matrix=False*)

Methods

`__init__`(name, primaries, whitepoint[, …])	
`use_derived_transformation_matrices`([usage])	Enables or disables usage of both derived transformations matrices, the normalised primary matrix and its inverse in subsequent computations.

colour.RGB_COLOURSPACES

colour.**RGB_COLOURSPACES = CaseInsensitiveMapping({u'ACEScc': ..., u'ACEScg': ..., u'Protune Native': ..., u**

Aggregated *RGB* colourspaces.

RGB_COLOURSPACES : CaseInsensitiveMapping

Aliases:

- 'aces': ACES_2065_1_COLOURSPACE.name
- 'adobe1998': ADOBE_RGB_1998_COLOURSPACE.name
- 'prophoto': PROPHOTO_RGB_COLOURSPACE.name

colour.models

ACES_2065_1_COLOURSPACE	*ACES2065-1* colourspace, base encoding, used for exchange of full fidelity
ACES_CC_COLOURSPACE	*ACEScc* colourspace, a working space for color correctors, target for ASC-CDL
ACES_CCT_COLOURSPACE	*ACEScct* colourspace, an alternative working space for colour correctors,
ACES_PROXY_COLOURSPACE	*ACESproxy* colourspace, a lightweight encoding for transmission over HD-SDI
ACES_CG_COLOURSPACE	*ACEScg* colourspace, a working space for paint/compositor applications that
ADOBE_RGB_1998_COLOURSPACE	*Adobe RGB (1998)* colourspace.
ADOBE_WIDE_GAMUT_RGB_COLOURSPACE	*Adobe Wide Gamut RGB* colourspace.
ALEXA_WIDE_GAMUT_COLOURSPACE	*ALEXA Wide Gamut* colourspace.
APPLE_RGB_COLOURSPACE	*Apple RGB* colourspace.
BEST_RGB_COLOURSPACE	*Best RGB* colourspace.
BETA_RGB_COLOURSPACE	*Beta RGB* colourspace.
BT470_525_COLOURSPACE	*ITU-R BT.470 - 525* colourspace.
BT470_625_COLOURSPACE	*ITU-R BT.470 - 625* colourspace.
BT709_COLOURSPACE	*ITU-R BT.709* colourspace.
BT2020_COLOURSPACE	*ITU-R BT.2020* colourspace.
CIE_RGB_COLOURSPACE	*CIE RGB* colourspace.
CINEMA_GAMUT_COLOURSPACE	*Cinema Gamut* colourspace.
COLOR_MATCH_RGB_COLOURSPACE	*ColorMatch RGB* colourspace.
DCI_P3_COLOURSPACE	*DCI-P3* colourspace.
DCI_P3_P_COLOURSPACE	*DCI-P3+* colourspace.
DON_RGB_4_COLOURSPACE	*Don RGB 4* colourspace.
ECI_RGB_V2_COLOURSPACE	*ECI RGB v2* colourspace.
EKTA_SPACE_PS_5_COLOURSPACE	*Ekta Space PS 5* colourspace.
PROTUNE_NATIVE_COLOURSPACE	*Protune Native* colourspace.
MAX_RGB_COLOURSPACE	*Max RGB* colourspace.
NTSC_COLOURSPACE	*NTSC* colourspace.
PAL_SECAM_COLOURSPACE	*Pal/Secam* colourspace.
RED_COLOR_COLOURSPACE	*REDcolor* colourspace.
RED_COLOR_2_COLOURSPACE	*REDcolor2* colourspace.
RED_COLOR_3_COLOURSPACE	*REDcolor3* colourspace.
RED_COLOR_4_COLOURSPACE	*REDcolor4* colourspace.
RED_WIDE_GAMUT_RGB_COLOURSPACE	*REDWideGamutRGB* colourspace.
DRAGON_COLOR_COLOURSPACE	*DRAGONcolor* colourspace.
DRAGON_COLOR_2_COLOURSPACE	*DRAGONcolor2* colourspace.
ROMM_RGB_COLOURSPACE	*ROMM RGB* colourspace.
RIMM_RGB_COLOURSPACE	*RIMM RGB* colourspace. In cases in which it is necessary to identify a
ERIMM_RGB_COLOURSPACE	*ERIMM RGB* colourspace.

Continued on next page

Table 3.141 – continued from previous page

PROPHOTO_RGB_COLOURSPACE	*ProPhoto RGB* colourspace, an alias colourspace for *ROMM RGB*.
RUSSELL_RGB_COLOURSPACE	*Russell RGB* colourspace.
SMPTE_240M_COLOURSPACE	*SMPTE 240M* colourspace.
S_GAMUT_COLOURSPACE	*S-Gamut* colourspace.
S_GAMUT3_COLOURSPACE	*S-Gamut3* colourspace.
S_GAMUT3_CINE_COLOURSPACE	*S-Gamut3.Cine* colourspace.
sRGB_COLOURSPACE	*sRGB* colourspace.
V_GAMUT_COLOURSPACE	*V-Gamut* colourspace.
XTREME_RGB_COLOURSPACE	*Xtreme RGB* colourspace.

colour.models.ACES_2065_1_COLOURSPACE

`colour.models.ACES_2065_1_COLOURSPACE = RGB_Colourspace(ACES2065-1, [[ 7.34700000e-01, 2.65300000e-01], [ 0.0`

ACES2065-1 colourspace, base encoding, used for exchange of full fidelity images and archiving.

References

- [TheAoMPAaSciencesSciencea TCouncilAcademyCESACESPSubcommittee14c]
- [TheAoMPAaSciencesSciencea TCouncilAcademyCESACESPSubcommittee14d]
- [TheAoMPAaSciencesSciencea TCouncilAcademyCESACESPSubcommittee]

ACES_2065_1_COLOURSPACE : RGB_Colourspace

colour.models.ACES_CC_COLOURSPACE

`colour.models.ACES_CC_COLOURSPACE = RGB_Colourspace(ACEScc, [[ 0.713, 0.293], [ 0.165, 0.83 ], [ 0.128, 0.04`

ACEScc colourspace, a working space for color correctors, target for ASC-CDL values created on-set.

References

- [TheAoMPAaSciencesSciencea TCouncilAcademyCESACESPSubcommittee14c]
- [TheAoMPAaSciencesSciencea TCouncilAcademyCESACESPSubcommittee14d]
- [TheAoMPAaSciencesSciencea TCouncilAcademyCESACESPSubcommittee14b]
- [TheAoMPAaSciencesSciencea TCouncilAcademyCESACESPSubcommittee]

ACES_CC_COLOURSPACE : RGB_Colourspace

colour.models.ACES_CCT_COLOURSPACE

`colour.models.ACES_CCT_COLOURSPACE = RGB_Colourspace(ACEScct, [[ 0.713, 0.293], [ 0.165, 0.83 ], [ 0.128, 0.0`

ACEScct colourspace, an alternative working space for colour correctors, intended to be transient and internal to software or hardware systems, and is specifically not intended for interchange or archiving.

References

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

ACES_CCT_COLOURSPACE : RGB_Colourspace

colour.models.ACES_PROXY_COLOURSPACE

`colour.models.`**`ACES_PROXY_COLOURSPACE = RGB_Colourspace(ACESproxy, [[ 0.713, 0.293], [ 0.165, 0.83 ], [ 0.128,`**

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

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

ACES_PROXY_COLOURSPACE : RGB_Colourspace

colour.models.ACES_CG_COLOURSPACE

`colour.models.`**`ACES_CG_COLOURSPACE = RGB_Colourspace(ACEScg, [[ 0.713, 0.293], [ 0.165, 0.83 ], [ 0.128, 0.04`**

ACEScg colourspace, a working space for paint/compositor applications that don't support ACES2065-1 or ACEScc.

References

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee15]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

ACES_CG_COLOURSPACE : RGB_Colourspace

colour.models.ADOBE_RGB_1998_COLOURSPACE

`colour.models.`**`ADOBE_RGB_1998_COLOURSPACE = RGB_Colourspace(Adobe RGB (1998), [[ 0.64, 0.33], [ 0.21, 0.71],`**

Adobe RGB (1998) colourspace.

References

- *[AdobeSystems05]*

ADOBE_RGB_1998_COLOURSPACE : RGB_Colourspace

colour.models.ADOBE_WIDE_GAMUT_RGB_COLOURSPACE

`colour.models.`**`ADOBE_WIDE_GAMUT_RGB_COLOURSPACE = RGB_Colourspace(Adobe Wide Gamut RGB, [[ 0.7347, 0.2653], [`**

Adobe Wide Gamut RGB colourspace.

References

- *[Wik{]*

ADOBE_WIDE_GAMUT_RGB_COLOURSPACE : RGB_Colourspace

colour.models.ALEXA_WIDE_GAMUT_COLOURSPACE

`colour.models.`**`ALEXA_WIDE_GAMUT_COLOURSPACE = RGB_Colourspace(ALEXA Wide Gamut, [[ 0.684 , 0.313 ], [ 0.221 ,`**

ALEXA Wide Gamut colourspace.

References

- *[ARR12]*

ALEXA_WIDE_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.APPLE_RGB_COLOURSPACE

`colour.models.`**`APPLE_RGB_COLOURSPACE = RGB_Colourspace(Apple RGB, [[ 0.625, 0.34 ], [ 0.28 , 0.595], [ 0.155,`**

Apple RGB colourspace.

References

- *[SBS99]*

APPLE_RGB_COLOURSPACE : RGB_Colourspace

colour.models.BEST_RGB_COLOURSPACE

`colour.models.`**`BEST_RGB_COLOURSPACE = RGB_Colourspace(Best RGB, [[ 0.73519164, 0.26480836], [ 0.21533613, 0.77`**

Best RGB colourspace.

References

- *[Huta]*

BEST_RGB_COLOURSPACE : RGB_Colourspace

colour.models.BETA_RGB_COLOURSPACE

`colour.models.BETA_RGB_COLOURSPACE = RGB_Colourspace(Beta RGB, [[ 0.6888, 0.3112], [ 0.1986, 0.7551], [ 0.126`
Beta RGB colourspace.

References

- *[Lin14]*

BETA_RGB_COLOURSPACE : RGB_Colourspace

colour.models.BT470_525_COLOURSPACE

`colour.models.BT470_525_COLOURSPACE = RGB_Colourspace(ITU-R BT.470 - 525, [[ 0.67, 0.33], [ 0.21, 0.71], [ 0.`
ITU-R BT.470 - 525 colourspace.

References

- *[InternationalTUnion98]*

BT470_525_COLOURSPACE : RGB_Colourspace

colour.models.BT470_625_COLOURSPACE

`colour.models.BT470_625_COLOURSPACE = RGB_Colourspace(ITU-R BT.470 - 625, [[ 0.64, 0.33], [ 0.29, 0.6 ], [ 0.`
ITU-R BT.470 - 625 colourspace.

References

- *[InternationalTUnion98]*

BT470_625_COLOURSPACE : RGB_Colourspace

colour.models.BT709_COLOURSPACE

`colour.models.BT709_COLOURSPACE = RGB_Colourspace(ITU-R BT.709, [[ 0.64, 0.33], [ 0.3 , 0.6 ], [ 0.15, 0.06]`
ITU-R BT.709 colourspace.

References

- [InternationalTUnion15b]

BT709_COLOURSPACE : RGB_Colourspace

colour.models.BT2020_COLOURSPACE

colour.models.**BT2020_COLOURSPACE = RGB_Colourspace(ITU-R BT.2020, [[0.708, 0.292], [0.17 , 0.797], [0.131,**
ITU-R BT.2020 colourspace.

References

- [InternationalTUnion15a]

BT2020_COLOURSPACE : RGB_Colourspace

colour.models.CIE_RGB_COLOURSPACE

colour.models.**CIE_RGB_COLOURSPACE = RGB_Colourspace(CIE RGB, [[0.73474284, 0.26525716], [0.27377903, 0.717**
CIE RGB colourspace.

References

- [FBH97]

CIE_RGB_COLOURSPACE : RGB_Colourspace

colour.models.CINEMA_GAMUT_COLOURSPACE

colour.models.**CINEMA_GAMUT_COLOURSPACE = RGB_Colourspace(Cinema Gamut, [[0.74, 0.27], [0.17, 1.14], [0.08,**
Cinema Gamut colourspace.

References

- [Can14]

CINEMA_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.COLOR_MATCH_RGB_COLOURSPACE

colour.models.**COLOR_MATCH_RGB_COLOURSPACE = RGB_Colourspace(ColorMatch RGB, [[0.63 , 0.34], [0.295, 0.605**
ColorMatch RGB colourspace.

References

- *[Lin14]*

COLOR_MATCH_RGB_COLOURSPACE : RGB_Colourspace

colour.models.DCI_P3_COLOURSPACE

colour.models.**DCI_P3_COLOURSPACE = RGB_Colourspace(DCI-P3, [[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.06]**
DCI-P3 colourspace.

References

- *[DigitalCInitiatives07]*
- *[HewlettPDCompany09]*

DCI_P3_COLOURSPACE : RGB_Colourspace

colour.models.DCI_P3_P_COLOURSPACE

colour.models.**DCI_P3_P_COLOURSPACE = RGB_Colourspace(DCI-P3+, [[0.74, 0.27], [0.22, 0.78], [0.09, -0.09]],**
DCI-P3+ colourspace.

References

- *[Can14]*

DCI_P3_P_COLOURSPACE : RGB_Colourspace

colour.models.DON_RGB_4_COLOURSPACE

colour.models.**DON_RGB_4_COLOURSPACE = RGB_Colourspace(Don RGB 4, [[0.69612069, 0.29956897], [0.21468298, 0.**
Don RGB 4 colourspace.

References

- *[Hutb]*

DON_RGB_4_COLOURSPACE : RGB_Colourspace

colour.models.ECI_RGB_V2_COLOURSPACE

colour.models.**ECI_RGB_V2_COLOURSPACE = RGB_Colourspace(ECI RGB v2, [[0.67010309, 0.32989691], [0.20990566,**
ECI RGB v2 colourspace.

References

- *[EuropeanCInitiative02]*

ECI_RGB_V2_COLOURSPACE : RGB_Colourspace

colour.models.EKTA_SPACE_PS_5_COLOURSPACE

```
colour.models.EKTA_SPACE_PS_5_COLOURSPACE = RGB_Colourspace(Ekta Space PS 5, [[ 0.69473684, 0.30526316], [ 0.
```

Ekta Space PS 5 colourspace.

References

- *[Hol]*

EKTA_SPACE_PS_5_COLOURSPACE : RGB_Colourspace

colour.models.PROTUNE_NATIVE_COLOURSPACE

```
colour.models.PROTUNE_NATIVE_COLOURSPACE = RGB_Colourspace(Protune Native, [[ 0.69848046, 0.19302645], [ 0.32
```

Protune Native colourspace.

References

- *[GDM16]*
- *[Man15]*

PROTUNE_NATIVE_COLOURSPACE : RGB_Colourspace

colour.models.MAX_RGB_COLOURSPACE

```
colour.models.MAX_RGB_COLOURSPACE = RGB_Colourspace(Max RGB, [[ 0.73413379, 0.26586621], [ 0.10039113, 0.8996
```

Max RGB colourspace.

References

- *[Hutc]*

MAX_RGB_COLOURSPACE : RGB_Colourspace

colour.models.NTSC_COLOURSPACE

```
colour.models.NTSC_COLOURSPACE = RGB_Colourspace(NTSC, [[ 0.67, 0.33], [ 0.21, 0.71], [ 0.14, 0.08]], [ 0.310
```

NTSC colourspace.

References

- [InternationalTUnion98]

NTSC_COLOURSPACE : RGB_Colourspace

colour.models.PAL_SECAM_COLOURSPACE

colour.models.**PAL_SECAM_COLOURSPACE = RGB_Colourspace(Pal/Secam, [[0.64, 0.33], [0.29, 0.6], [0.15, 0.06]**

Pal/Secam colourspace.

References

- [InternationalTUnion98]

PAL_SECAM_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_COLOURSPACE

colour.models.**RED_COLOR_COLOURSPACE = RGB_Colourspace(REDcolor, [[0.699747 , 0.32904693], [0.30426404, 0.62**

REDcolor colourspace.

References

- [Man15]
- [SonyImageworks12]

RED_COLOR_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_2_COLOURSPACE

colour.models.**RED_COLOR_2_COLOURSPACE = RGB_Colourspace(REDcolor2, [[0.87868251, 0.32496401], [0.30088871,**

REDcolor2 colourspace.

References

- [Man15]
- [SonyImageworks12]

RED_COLOR_2_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_3_COLOURSPACE

colour.models.**RED_COLOR_3_COLOURSPACE = RGB_Colourspace(REDcolor3, [[0.70118104, 0.32901416], [0.3006003 ,**

REDcolor3 colourspace.

References

- *[Man15]*
- *[SonyImageworks12]*

RED_COLOR_3_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_4_COLOURSPACE

colour.models.**RED_COLOR_4_COLOURSPACE = RGB_Colourspace(REDcolor4, [[0.70118059, 0.3290137], [0.3006004 ,**
REDcolor4 colourspace.

References

- *[Man15]*
- *[SonyImageworks12]*

RED_COLOR_4_COLOURSPACE : RGB_Colourspace

colour.models.RED_WIDE_GAMUT_RGB_COLOURSPACE

colour.models.**RED_WIDE_GAMUT_RGB_COLOURSPACE = RGB_Colourspace(REDWideGamutRGB, [[0.780308, 0.304253], [0.1**
REDWideGamutRGB colourspace.

References

- *[Man15]*
- *[Nat16]*
- *[SonyImageworks12]*

RED_WIDE_GAMUT_RGB_COLOURSPACE : RGB_Colourspace

colour.models.DRAGON_COLOR_COLOURSPACE

colour.models.**DRAGON_COLOR_COLOURSPACE = RGB_Colourspace(DRAGONcolor, [[0.75304422, 0.32783058], [0.2995702**
DRAGONcolor colourspace.

References

- *[Man15]*
- *[SonyImageworks12]*

DRAGON_COLOR_COLOURSPACE : RGB_Colourspace

colour.models.DRAGON_COLOR_2_COLOURSPACE

`colour.models.DRAGON_COLOR_2_COLOURSPACE = RGB_Colourspace(DRAGONcolor2, [[ 0.75304449, 0.32783103], [ 0.2995`

DRAGONcolor2 colourspace.

References

- *[Man15]*
- *[SonyImageworks12]*

DRAGON_COLOR_2_COLOURSPACE : RGB_Colourspace

colour.models.ROMM_RGB_COLOURSPACE

`colour.models.ROMM_RGB_COLOURSPACE = RGB_Colourspace(ROMM RGB, [[ 7.34700000e-01, 2.65300000e-01], [ 1.59600`

ROMM RGB colourspace.

References

- *[ANS03]*
- *[SWG00]*

ROMM_RGB_COLOURSPACE : RGB_Colourspace

colour.models.RIMM_RGB_COLOURSPACE

`colour.models.RIMM_RGB_COLOURSPACE = RGB_Colourspace(RIMM RGB, [[ 7.34700000e-01, 2.65300000e-01], [ 1.59600`

RIMM RGB colourspace. In cases in which it is necessary to identify a specific precision level, the notation *RIMM8 RGB*, *RIMM12 RGB* and *RIMM16 RGB* is used.

References

- *[SWG00]*

RIMM_RGB_COLOURSPACE : RGB_Colourspace

colour.models.ERIMM_RGB_COLOURSPACE

`colour.models.ERIMM_RGB_COLOURSPACE = RGB_Colourspace(ERIMM RGB, [[ 7.34700000e-01, 2.65300000e-01], [ 1.596`

ERIMM RGB colourspace.

References

- *[SWG00]*

ERIMM_RGB_COLOURSPACE : RGB_Colourspace

colour.models.PROPHOTO_RGB_COLOURSPACE

colour.models.**PROPHOTO_RGB_COLOURSPACE = RGB_Colourspace(ProPhoto RGB, [[7.34700000e-01, 2.65300000e-01], [**

ProPhoto RGB colourspace, an alias colourspace for *ROMM RGB*.

References

- *[ANS03]*
- *[SWG00]*

PROPHOTO_RGB_COLOURSPACE : RGB_Colourspace

colour.models.RUSSELL_RGB_COLOURSPACE

colour.models.**RUSSELL_RGB_COLOURSPACE = RGB_Colourspace(Russell RGB, [[0.69, 0.31], [0.18, 0.77], [0.1 , 0**

Russell RGB colourspace.

References

- *[Cot]*

RUSSELL_RGB_COLOURSPACE : RGB_Colourspace

colour.models.SMPTE_240M_COLOURSPACE

colour.models.**SMPTE_240M_COLOURSPACE = RGB_Colourspace(SMPTE 240M, [[0.63 , 0.34], [0.31 , 0.595], [0.155**

SMPTE 240M colourspace.

References

- *[SocietyoMPaTEngineers99]*
- *[SocietyoMPaTEngineers04]*

SMPTE_240M_COLOURSPACE : RGB_Colourspace

colour.models.S_GAMUT_COLOURSPACE

colour.models.**S_GAMUT_COLOURSPACE = RGB_Colourspace(S-Gamut, [[0.73 , 0.28], [0.14 , 0.855], [0.1 , -0.05**

S-Gamut colourspace.

References

- *[GDY+]*
- *[SonyCorporationb]*

S_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.S_GAMUT3_COLOURSPACE

`colour.models.S_GAMUT3_COLOURSPACE = RGB_Colourspace(S-Gamut3, [[ 0.73 , 0.28 ], [ 0.14 , 0.855], [ 0.1 , -0.`
S-Gamut3 colourspace.

References

- *[SonyCorporationc]*

S_GAMUT3_COLOURSPACE : RGB_Colourspace

colour.models.S_GAMUT3_CINE_COLOURSPACE

`colour.models.S_GAMUT3_CINE_COLOURSPACE = RGB_Colourspace(S-Gamut3.Cine, [[ 0.766, 0.275], [ 0.225, 0.8 ], [`
S-Gamut3.Cine colourspace.

References

- *[SonyCorporationa]*

S_GAMUT3_CINE_COLOURSPACE : RGB_Colourspace

colour.models.sRGB_COLOURSPACE

`colour.models.sRGB_COLOURSPACE = RGB_Colourspace(sRGB, [[ 0.64, 0.33], [ 0.3 , 0.6 ], [ 0.15, 0.06]], [ 0.31`
sRGB colourspace.

References

- *[InternationalECommission99]*
- *[InternationalTUnion15b]*

sRGB_COLOURSPACE : RGB_Colourspace

colour.models.V_GAMUT_COLOURSPACE

`colour.models.V_GAMUT_COLOURSPACE = RGB_Colourspace(V-Gamut, [[ 0.73 , 0.28 ], [ 0.165, 0.84 ], [ 0.1 , -0.0`
V-Gamut colourspace.

References

- *[Pan14]*

V_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.XTREME_RGB_COLOURSPACE

colour.models.**XTREME_RGB_COLOURSPACE = RGB_Colourspace(Xtreme RGB, [[1., 0.], [0., 1.], [0., 0.]], [0.345**

Xtreme RGB colourspace.

References

- *[Hutd]*

XTREME_RGB_COLOURSPACE : RGB_Colourspace

Ancillary Objects

colour.models

spectral_to_aces_relative_exposure_values(spd)	Converts given spectral power distribution to *ACES2065-1* colourspace relative exposure values.
ACES_RICD	Implements support for the *CIE RGB* colour matching functions.

colour.models.spectral_to_aces_relative_exposure_values

colour.models.**spectral_to_aces_relative_exposure_values**(*spd, illuminant=SpectralPowerDistribution(name='D60', ...)*)

Converts given spectral power distribution to *ACES2065-1* colourspace relative exposure values.

Parameters

- **spd** (SpectralPowerDistribution) – Spectral power distribution.
- **illuminant** (SpectralPowerDistribution, optional) – *Illuminant* spectral power distribution.

Returns *ACES2065-1* colourspace relative exposure values array.

Return type ndarray, (3,)

Notes

- Output *ACES2065-1* colourspace relative exposure values array is in range [0, 1].

References

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

Examples

```
>>> from colour import COLOURCHECKERS_SPDS
>>> spd = COLOURCHECKERS_SPDS['ColorChecker N Ohta']['dark skin']
>>> spectral_to_aces_relative_exposure_values(spd)
array([ 0.1187697...,  0.0870866...,  0.0589442...])
```

colour.models.ACES_RICD

colour.models.**ACES_RICD = RGB_ColourMatchingFunctions(name='ACES RICD', ...)**

Implements support for the *CIE RGB* colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignal or MultiSpectralPowerDistribution or array_like or dict_like, optional) – Data to be stored in the multi-spectral power distribution.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralPowerDistribution 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.SpectralPowerDistribution class instances.

Other Parameters

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

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).

Continued on next page

Table 3.143 – continued from previous page

OETFS	Supported opto-electrical transfer functions (OETFs / OECFs).
oetf_reverse(value[, function])	Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given reverse opto-electronic transfer function (OETF / OECF).
OETFS_REVERSE	Supported reverse opto-electrical transfer functions (OETFs / OECFs).

colour.oetf

colour.**oetf**(*value*, *function='sRGB'*, ***kwargs*)

Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given opto-electronic transfer function (OETF / OECF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'sRGB', 'ARIB STD-B67', 'DCI-P3', 'DICOM GSDF', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'ProPhoto RGB', 'RIMM RGB', 'ROMM RGB', 'SMPTE 240M', 'ST 2084'}**, Opto-electronic transfer function (OETF / OECF).

Other Parameters

- **E_clip** (*numeric, optional*) – {colour.models.oetf_RIMMRGB()}, Maximum exposure level.
- **I_max** (*numeric, optional*) – {colour.models.oetf_ROMMRGB(), colour.models.oetf_RIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **L_p** (*numeric, optional*) – {colour.models.oetf_ST2084()}, Display peak luminance cd/m^2.
- **is_12_bits_system** (*bool*) – {colour.models.oetf_BT2020()}, *ITU-R BT.2020 alpha* and *beta* constants are used if system is not 12-bit.
- **r** (*numeric, optional*) – {colour.models.oetf_ARIBSTDB67()}, Video level corresponding to reference white level.

Returns $R'G'B'$ video component signal value.

Return type numeric or ndarray

Examples

```
>>> oetf(0.18)
0.4613561...
>>> oetf(0.18, function='ITU-R BT.2020')
0.4090077...
>>> oetf(0.18, function='ST 2084', L_p=1000)
...
0.1820115...
```

colour.OETFS

`colour.OETFS = CaseInsensitiveMapping({'ITU-R BT.2020': ..., 'ITU-R BT.2100 PQ': ..., 'SMPTE 240M': ..., 'DC`

Supported opto-electrical transfer functions (OETFs / OECFs).

OETFS [CaseInsensitiveMapping] **{'sRGB', 'ARIB STD-B67', 'DCI-P3', 'DICOM GSDF', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'ProPhoto RGB', 'RIMM RGB', 'ROMM RGB', 'SMPTE 240M', 'ST 2084'}**

colour.oetf_reverse

colour.**oetf_reverse**(*value*, *function='sRGB'*, ***kwargs*)

Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given reverse opto-electronic transfer function (OETF / OECF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'sRGB', 'ARIB STD-B67', 'ITU-R BT.2100 HLD', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709'}**, Reverse opto-electronic transfer function (OETF / OECF).

Other Parameters r (*numeric, optional*) – {colour.models.oetf_ARIBSTDB67()}, Video level corresponding to reference white level.

Returns Tristimulus values at the display.

Return type numeric or ndarray

Examples

```
>>> oetf_reverse(0.461356129500442)
0.1...
>>> oetf_reverse(
...     0.409007728864150, function='ITU-R BT.601')
0.1...
```

colour.OETFS_REVERSE

`colour.OETFS_REVERSE = CaseInsensitiveMapping({'ITU-R BT.2100 PQ': ..., 'ITU-R BT.2100 HLD': ..., 'sRGB': ..`

Supported reverse opto-electrical transfer functions (OETFs / OECFs).

OETFS_REVERSE [CaseInsensitiveMapping] **{'sRGB', 'ARIB STD-B67', 'ITU-R BT.2100 HLD', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709'}**

colour.models

oetf_ARIBSTDB67(E[, r])	Defines *ARIB STD-B67 (Hybrid Log-Gamma)* opto-electrical transfer function (OETF / OECF).
oetf_reverse_ARIBSTDB67(E_p[, r])	Defines *ARIB STD-B67 (Hybrid Log-Gamma)* reverse opto-electrical transfer function (OETF / OECF).

Continued on next page

Table 3.144 – continued from previous page

oetf_DCIP3(XYZ)	Defines the *DCI-P3* colourspace opto-electronic transfer function (OETF / OECF).
oetf_DICOMGSDF(L)	Defines the *DICOM - Grayscale Standard Display Function* opto-electronic transfer function (OETF / OECF).
oetf_BT2020(E[, is_12_bits_system])	Defines *Recommendation ITU-R BT.2020* opto-electrical transfer function (OETF / OECF).
oetf_BT2100_HLG(E)	Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF / OECF).
oetf_reverse_BT2100_HLG(E)	Defines *Recommendation ITU-R BT.2100 Reference HLG* reverse opto-electrical transfer function (OETF / OECF).
oetf_BT2100_PQ(E)	Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-electrical transfer function (OETF / OECF).
oetf_reverse_BT2100_PQ(E_p)	Defines *Recommendation ITU-R BT.2100 Reference PQ* reverse opto-electrical transfer function (OETF / OECF).
oetf_BT601(L)	Defines *Recommendation ITU-R BT.601-7* opto-electronic transfer function (OETF / OECF).
oetf_reverse_BT601(E)	Defines *Recommendation ITU-R BT.601-7* reverse opto-electronic transfer function (OETF / OECF).
oetf_BT709(L)	Defines *Recommendation ITU-R BT.709-6* opto-electronic transfer function (OETF / OECF).
oetf_reverse_BT709(V)	Defines *Recommendation ITU-R BT.709-6* reverse opto-electronic transfer function (OETF / OECF).
oetf_ProPhotoRGB(X[, I_max])	Defines the *ROMM RGB* encoding opto-electronic transfer function (OETF / OECF).
oetf_RIMMRGB(X[, I_max, E_clip])	Defines the *RIMM RGB* encoding opto-electronic transfer function (OETF / OECF).
oetf_ROMMRGB(X[, I_max])	Defines the *ROMM RGB* encoding opto-electronic transfer function (OETF / OECF).
oetf_SMPTE240M(L_c)	Defines *SMPTE 240M* opto-electrical transfer function (OETF / OECF).
oetf_ST2084(C[, L_p])	Defines *SMPTE ST 2084:2014* optimised perceptual opto-electronic transfer function (OETF / OECF).
oetf_sRGB(L)	Defines the *sRGB* colourspace opto-electronic transfer function (OETF / OECF).
oetf_reverse_sRGB(V)	Defines the *sRGB* colourspace reverse opto-electronic transfer function (OETF / OECF).

colour.models.oetf_ARIBSTDB67

colour.models.**oetf_ARIBSTDB67**(*E*, *r=0.5*)

Defines *ARIB STD-B67 (Hybrid Log-Gamma)* opto-electrical transfer function (OETF / OECF).

Parameters

- **E** (numeric or array_like) – Voltage normalized 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.

Returns Resulting non-linear signal E'.

Return type numeric or ndarray

References

- *[AssociationoRIaBusinesses15]*

Examples

```
>>> oetf_ARIBSTDB67(0.18)
0.2121320...
```

colour.models.oetf_reverse_ARIBSTDB67

colour.models.**oetf_reverse_ARIBSTDB67**(*E_p*, *r=0.5*)
Defines *ARIB STD-B67 (Hybrid Log-Gamma)* reverse opto-electrical transfer function (OETF / OECF).

Parameters

- **E_p** (numeric or array_like) – Non-linear signal E'.
- **r** (numeric, optional) – Video level corresponding to reference white level.

Returns Voltage E normalized 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

References

- *[AssociationoRIaBusinesses15]*

Examples

```
>>> oetf_reverse_ARIBSTDB67(0.212132034355964)
0.1799999...
```

colour.models.oetf_DCIP3

colour.models.**oetf_DCIP3**(*XYZ*)
Defines the *DCI-P3* colourspace opto-electronic transfer function (OETF / OECF).

Parameters **XYZ** (numeric or array_like) – *CIE XYZ* tristimulus values.

Returns Non-linear *CIE XYZ'* tristimulus values.

Return type numeric or ndarray

References

- *[DigitalCInitiatives07]*

Examples

```
>>> oetf_DCIP3(0.18)
461.9922059...
```

colour.models.oetf_DICOMGSDF

colour.models.**oetf_DICOMGSDF**(*L*)

Defines the *DICOM - Grayscale Standard Display Function* opto-electronic transfer function (OETF / OECF).

Parameters **L** (numeric or array_like) – *Luminance* L.

Returns Just-Noticeable Difference (JND) Index, j in domain 1 to 1023.

Return type numeric or ndarray

References

- *[NationalEMAssociation04]*

Examples

```
>>> oetf_DICOMGSDF(130.065284012159790)
511.9964806...
```

colour.models.oetf_BT2020

colour.models.**oetf_BT2020**(*E, is_12_bits_system=False*)

Defines *Recommendation ITU-R BT.2020* opto-electrical transfer function (OETF / OECF).

Parameters

- **E** (numeric or array_like) – Voltage E normalized by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera colour channel R, G, B.
- **is_12_bits_system** (bool) – *BT.709 alpha* and *beta* constants are used if system is not 12-bit.

Returns Resulting non-linear signal E'.

Return type numeric or ndarray

References

- *[InternationalTUnion15a]*

Examples

```
>>> oetf_BT2020(0.18)
0.4090077...
```

colour.models.oetf_BT2100_HLG

colour.models.**oetf_BT2100_HLG**(E)

Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF / OECF).

The OETF maps relative scene linear light into the non-linear *HLG* signal value.

Parameters **E** (numeric or array_like) – E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure, normalized to the range [0, 1].

Returns E is the resulting non-linear signal R', G', B' in the range [0, 1].

Return type numeric or ndarray

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> oetf_BT2100_HLG(0.18 / 12)
0.2121320...
```

colour.models.oetf_reverse_BT2100_HLG

colour.models.**oetf_reverse_BT2100_HLG**(E)

Defines *Recommendation ITU-R BT.2100 Reference HLG* reverse opto-electrical transfer function (OETF / OECF).

Parameters **E_p** (numeric or array_like) – E is the resulting non-linear signal R', G', B' in the range [0, 1].

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, normalized to the range [0, 1].

Return type numeric or ndarray

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> oetf_reverse_BT2100_HLG(0.212132034355964)
0.0149999...
```

colour.models.oetf_BT2100_PQ

colour.models.**oetf_BT2100_PQ**(E)

Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-electrical transfer function (OETF / OECF).

The OETF maps relative scene linear light into the non-linear *PQ* signal value.

Parameters **E** (numeric or array_like) – $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure. The values E, R_S, G_S, B_S, Y_S, I_S are in the range [0, 1].

Returns E is the resulting non-linear signal (R', G', B') in the range [0, 1].

Return type numeric or ndarray

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> oetf_BT2100_PQ(0.1)
0.7247698...
```

colour.models.oetf_reverse_BT2100_PQ

colour.models.**oetf_reverse_BT2100_PQ**(E_p)

Defines *Recommendation ITU-R BT.2100 Reference PQ* reverse opto-electrical transfer function (OETF / OECF).

Parameters **E_p** (numeric or array_like) – E is the resulting non-linear signal (R', G', B') in the range [0, 1].

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. The values E, R_S, G_S, B_S, Y_S, I_S are in the range [0, 1].

Return type numeric or ndarray

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> oetf_reverse_BT2100_PQ(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 Es.

Return type numeric or ndarray

References

- *[InternationalTUnion11b]*

Examples

```
>>> oetf_BT601(0.18)
0.4090077...
```

colour.models.oetf_reverse_BT601

colour.models.**oetf_reverse_BT601**(*E*)

Defines *Recommendation ITU-R BT.601-7* reverse opto-electronic transfer function (OETF / OECF).

Parameters **E** (numeric or array_like) – Electrical signal E.

Returns Corresponding *luminance* L of the image.

Return type numeric or ndarray

References

- *[InternationalTUnion11b]*

Examples

```
>>> oetf_reverse_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

References

- *[InternationalTUnion15b]*

Examples

```
>>> oetf_BT709(0.18)
0.4090077...
```

colour.models.oetf_reverse_BT709

`colour.models.oetf_reverse_BT709(V)`

Defines *Recommendation ITU-R BT.709-6* reverse opto-electronic transfer function (OETF / OECF).

Parameters **V** (numeric or array_like) – Electrical signal V.

Returns Corresponding *luminance* L of the image.

Return type numeric or ndarray

References

- *[InternationalTUnion15b]*

Examples

```
>>> oetf_reverse_BT709(0.409007728864150)
0.1...
```

colour.models.oetf_ProPhotoRGB

`colour.models.oetf_ProPhotoRGB(X, I_max=255)`

Defines the *ROMM RGB* encoding opto-electronic transfer function (OETF / OECF).

Parameters

- **X** (numeric or array_like) – Linear data X_{ROMM}.
- **I_max** (numeric, optional) – Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.

Returns Non-linear data X'_{ROMM}.

Return type numeric or ndarray

References

- *[ANS03]*
- *[SWG00]*

Examples

```
>>> oetf_ROMMRGB(0.18)
98.3564133...
```

colour.models.oetf_RIMMRGB

colour.models.**oetf_RIMMRGB**(*X*, *I_max=255*, *E_clip=2.0*)

Defines the *RIMM RGB* encoding opto-electronic transfer function (OETF / OECF).

RIMM RGB encoding non-linearity is based on that specified by *Recommendation ITU-R BT.709-6*.

Parameters

- **X** (numeric or array_like) – Linear data X_{RIMM}.
- **I_max** (numeric, optional) – Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **E_clip** (numeric, optional) – Maximum exposure level.

Returns Non-linear data X'_{RIMM}.

Return type numeric or ndarray

References

- *[SWG00]*

Examples

```
>>> oetf_RIMMRGB(0.18)
74.3768017...
```

colour.models.oetf_ROMMRGB

colour.models.**oetf_ROMMRGB**(*X*, *I_max=255*)

Defines the *ROMM RGB* encoding opto-electronic transfer function (OETF / OECF).

Parameters

- **X** (numeric or array_like) – Linear data X_{ROMM}.

- **I_max** (numeric, optional) – Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.

Returns Non-linear data X'_{ROMM}.

Return type numeric or ndarray

References

- *[ANS03]*
- *[SWG00]*

Examples

```
>>> oetf_ROMMRGB(0.18)
98.3564133...
```

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 normalized to the system reference white.

Returns Video signal output V_c of the reference camera normalized to the system reference white.

Return type numeric or ndarray

References

- *[SocietyoMPaTEngineers99]*

Examples

```
>>> oetf_SMPTE240M(0.18)
0.4022857...
```

colour.models.oetf_ST2084

colour.models.**oetf_ST2084**(*C*, *L_p=10000*)

Defines *SMPTE ST 2084:2014* optimised perceptual opto-electronic transfer function (OETF / OECF).

Parameters

- **C** (numeric or array_like) – Target optical output C in cd/m^2 of the ideal reference display.
- **L_p** (numeric, optional) – Display peak luminance cd/m^2.

Returns Color value abbreviated as N, normalized to the range [0, 1], 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

References

- *[MDolbyLaboratories14]*
- *[SocietyoMPaTEngineers14]*

Examples

```
>>> oetf_ST2084(10.0, 1000)
0.5080784...
```

colour.models.oetf_sRGB

`colour.models.oetf_sRGB(L)`

Defines the *sRGB* colourspace opto-electronic transfer function (OETF / OECF).

Parameters L (`numeric or array_like`) – *Luminance* L of the image.

Returns Corresponding electrical signal V.

Return type numeric or ndarray

References

- *[InternationalECommission99]*
- *[InternationalTUnion15b]*

Examples

```
>>> oetf_sRGB(0.18)
0.4613561...
```

colour.models.oetf_reverse_sRGB

`colour.models.oetf_reverse_sRGB(V)`

Defines the *sRGB* colourspace reverse opto-electronic transfer function (OETF / OECF).

Parameters V (`numeric or array_like`) – Electrical signal V.

Returns Corresponding *luminance* L of the image.

Return type numeric or ndarray

References

- *[InternationalECommission99]*
- *[InternationalTUnion15b]*

Examples

```
>>> oetf_reverse_sRGB(0.461356129500442)
0.1...
```

Ancillary Objects

colour

function_gamma(a[, exponent, …])	Defines a typical gamma encoding / decoding function.
function_linear(a)	Defines a typical linear encoding / decoding function, essentially a pass-through function.

colour.function_gamma

colour.**function_gamma**(*a*, *exponent=1*, *negative_number_handling=u'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

```
>>> function_gamma(0.18, 2.2)
0.0229932...
>>> function_gamma(-0.18, 2.0)
0.0323999...
>>> function_gamma(-0.18, 2.2)
nan
>>> function_gamma(-0.18, 2.2, 'Mirror')
-0.0229932...
>>> function_gamma(-0.18, 2.2, 'Preserve')
-0.1...
>>> function_gamma(-0.18, 2.2, 'Clamp')
0.0
```

colour.function_linear

colour.**function_linear**(*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

```
>>> function_linear(0.18)
0.18
```

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_reverse(value[, function])	Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given reverse electro-optical transfer function (EOTF / EOCF).
EOTFS_REVERSE	Supported reverse electro-optical transfer functions (EOTFs / EOCFs).

colour.eotf

colour.**eotf**(*value*, *function='ITU-R BT.1886'*, ***kwargs*)

Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given electro-optical transfer function (EOTF / EOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.1886', 'DCI-P3', 'DICOM GSDF', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ProPhoto RGB', 'RIMM RGB', 'ROMM RGB', 'SMPTE 240M', 'ST 2084'}**, Electro-optical transfer function (EOTF / EOCF).

Other Parameters

- **E_clip** (*numeric, optional*) – {colour.models.eotf_RIMMRGB()}, Maximum exposure level.
- **I_max** (*numeric, optional*) – {colour.models.eotf_ROMMRGB(), colour.models.eotf_RIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **L_B** (*numeric, optional*) – {colour.models.eotf_BT1886(), colour.models.eotf_BT2100_HLG()}, Screen luminance for black.
- **L_W** (*numeric, optional*) – {colour.models.eotf_BT1886(), colour.models.eotf_BT2100_HLG()}, Screen luminance for white.
- **L_p** (*numeric, optional*) – {colour.models.eotf_ST2084()}, Display peak luminance cd/m^2.
- **gamma** (*numeric, optional*) – {colour.models.eotf_BT2100_HLG()}, System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **is_12_bits_system** (*bool*) – {colour.models.eotf_BT2020()}, *ITU-R BT.2020 alpha* and *beta* constants are used if system is not 12-bit.

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({'ITU-R BT.2020': ..., 'ST 2084': ..., 'DCI-P3': ..., 'ITU-R BT.2100**

Supported electro-optical transfer functions (EOTFs / EOCFs).

EOTFS [CaseInsensitiveMapping] **{'DCI-P3', 'DICOM GSDF', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ProPhoto RGB', 'RIMM RGB', 'ROMM RGB', 'SMPTE 240M', 'ST 2084'}**

colour.eotf_reverse

colour.**eotf_reverse**(*value*, *function='ITU-R BT.1886'*, ***kwargs*)

Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given reverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.1886', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}**, Reverse electro-optical transfer function (EOTF / EOCF).

Other Parameters

- **L_B** (*numeric, optional*) – {colour.models.eotf_reverse_BT1886(), colour.models.eotf_reverse_BT2100_HLG()}, Screen luminance for black.
- **L_W** (*numeric, optional*) – {colour.models.eotf_reverse_BT1886(), colour.models.eotf_reverse_BT2100_HLG()}, Screen luminance for white.
- **gamma** (*numeric, optional*) – {colour.models.eotf_BT2100_HLG()}, System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.

Returns $R'G'B'$ video component signal value.

Return type numeric or ndarray

Examples

```
>>> eotf_reverse(0.11699185725296059)
0.4090077...
>>> eotf_reverse(
...     0.11699185725296059, function='ITU-R BT.1886')
0.4090077...
```

colour.EOTFS_REVERSE

colour.**EOTFS_REVERSE = CaseInsensitiveMapping({'ITU-R BT.1886': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-R BT.2100**

Supported reverse electro-optical transfer functions (EOTFs / EOCFs).

EOTFS_REVERSE [CaseInsensitiveMapping] **{'ITU-R BT.1886', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}**

colour.models

eotf_DCIP3(XYZ_p)	Defines the *DCI-P3* colourspace electro-optical transfer function (EOTF / EOCF).
eotf_DICOMGSDF(J)	Defines the *DICOM - Grayscale Standard Display Function* 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_reverse_BT1886(L[, L_B, L_W])	Defines *Recommendation ITU-R BT.1886* reverse electro-optical transfer function (EOTF / EOCF).

Continued on next page

Table 3.147 – continued from previous page

eotf_BT2020(E_p[, is_12_bits_system])	Defines *Recommendation ITU-R BT.2020* electro-optical transfer function (EOTF / EOCF).
eotf_BT2100_HLG(E_p[, L_B, L_W, gamma])	Defines *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).
eotf_reverse_BT2100_HLG(F_D[, L_B, L_W, gamma])	Defines *Recommendation ITU-R BT.2100 Reference HLG* reverse electro-optical transfer function (EOTF / EOCF).
eotf_BT2100_PQ(E_p)	Defines *Recommendation ITU-R BT.2100 Reference PQ* electro-optical transfer function (EOTF / EOCF).
eotf_reverse_BT2100_PQ(F_D)	Defines *Recommendation ITU-R BT.2100 Reference PQ* reverse electro-optical transfer function (EOTF / EOCF).
eotf_ProPhotoRGB(X_p[, I_max])	Defines the *ROMM RGB* encoding electro-optical transfer function (EOTF / EOCF).
eotf_RIMMRGB(X_p[, I_max, E_clip])	Defines the *RIMM RGB* encoding electro-optical transfer function (EOTF / EOCF).
eotf_ROMMRGB(X_p[, I_max])	Defines the *ROMM RGB* encoding electro-optical transfer function (EOTF / EOCF).
eotf_SMPTE240M(V_r)	Defines *SMPTE 240M* electro-optical transfer function (EOTF / EOCF).
eotf_ST2084(N[, L_p])	Defines *SMPTE ST 2084:2014* optimised perceptual electro-optical transfer function (EOTF / EOCF).

colour.models.eotf_DCIP3

colour.models.**eotf_DCIP3**(*XYZ_p*)

Defines the *DCI-P3* colourspace electro-optical transfer function (EOTF / EOCF).

Parameters **XYZ_p** (numeric or array_like) – Non-linear *CIE XYZ'* tristimulus values.

Returns *CIE XYZ* tristimulus values.

Return type numeric or ndarray

References

- *[DigitalCInitiatives07]*

Examples

```
>>> eotf_DCIP3(461.99220597484737)
0.18...
```

colour.models.eotf_DICOMGSDF

colour.models.**eotf_DICOMGSDF**(*J*)

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 range 1 to 1023.

Returns Corresponding *luminance* L.

Return type numeric or ndarray

References

- *[NationalEMAssociation04]*

Examples

```
>>> eotf_DICOMGSDF(512)
130.0652840...
```

colour.models.eotf_BT1886

colour.models.**eotf_BT1886**(*V*, *L_B=0*, *L_W=1*)

Defines *Recommendation ITU-R BT.1886* electro-optical transfer function (EOTF / EOCF).

Parameters

- **V** (numeric or array_like) – Input video signal level (normalized, 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

References

- *[InternationalTUnion11a]*

Examples

```
>>> eotf_BT1886(0.409007728864150)
0.1169918...
```

colour.models.eotf_reverse_BT1886

colour.models.**eotf_reverse_BT1886**(*L*, *L_B=0*, *L_W=1*)

Defines *Recommendation ITU-R BT.1886* reverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **L** (numeric or array_like) – Screen luminance in cd/m^2.
- **L_B** (numeric, optional) – Screen luminance for black.
- **L_W** (numeric, optional) – Screen luminance for white.

Returns Input video signal level (normalized, black at $V = 0$, to white at $V = 1$.

Return type numeric or ndarray

References

- *[InternationalTUnion11a]*

Examples

```
>>> eotf_reverse_BT1886(0.11699185725296059)
0.4090077...
```

colour.models.eotf_BT2020

colour.models.**eotf_BT2020**(*E_p, is_12_bits_system=False*)

Defines *Recommendation ITU-R BT.2020* electro-optical transfer function (EOTF / EOCF).

Parameters

- **E_p** (numeric or array_like) – Non-linear signal E'.
- **is_12_bits_system** (bool) – *BT.709 alpha* and *beta* constants are used if system is not 12-bit.

Returns Resulting voltage E.

Return type numeric or ndarray

References

- *[InternationalTUnion15a]*

Examples

```
>>> eotf_BT2020(0.705515089922121)
0.4999999...
```

colour.models.eotf_BT2100_HLG

colour.models.**eotf_BT2100_HLG**(*E_p, L_B=0, L_W=1000, gamma=None*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).

The EOTF maps the non-linear *HLG* signal into display light.

Parameters

- **E_p** (numeric or array_like) – E' denotes a non-linear colour value R', G', B' or L', M', S' in *HLG* space in range [0, 1].
- **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 $1000 cd/m^2$.

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

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> eotf_BT2100_HLG(0.212132034355964)
6.4760398...
```

colour.models.eotf_reverse_BT2100_HLG

colour.models.**eotf_reverse_BT2100_HLG**(*F_D*, *L_B=0*, *L_W=1000*, *gamma=None*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* reverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **F_D** (numeric or array_like) – Luminance F_D of a displayed linear component R_D, G_D, B_D or Y_D or I_D, in cd/m^2.
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2.
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000 cd/m^2$.

Returns E' denotes a non-linear colour value R', G', B' or L', M', S' in *HLG* space in range [0, 1].

Return type numeric or ndarray

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> eotf_reverse_BT2100_HLG(6.476039825649814)
0.2121320...
```

colour.models.eotf_BT2100_PQ

colour.models.**eotf_BT2100_PQ**(*E_p*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* electro-optical transfer function (EOTF / EOCF).

The EOTF maps the non-linear *PQ* signal into display light.

Parameters **E_p** (numeric or array_like) – E' denotes a non-linear colour value R', G', B' or L', M', S' in *PQ* space [0, 1].

Returns F_D is the luminance of a displayed linear component R_D, G_D, B_D or Y_D or I_D, in cd/m^2.

Return type numeric or ndarray

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> eotf_BT2100_PQ(0.724769816665726)
779.9883608...
```

colour.models.eotf_reverse_BT2100_PQ

colour.models.**eotf_reverse_BT2100_PQ**(*F_D*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* reverse electro-optical transfer function (EOTF / EOCF).

Parameters **F_D** (numeric or array_like) – F_D is the luminance of a displayed linear component R_D, G_D, B_D or Y_D or I_D, in cd/m^2.

Returns E' denotes a non-linear colour value R', G', B' or L', M', S' in *PQ* space [0, 1].

Return type numeric or ndarray

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> eotf_reverse_BT2100_PQ(779.988360834085370)
0.7247698...
```

colour.models.eotf_ProPhotoRGB

colour.models.**eotf_ProPhotoRGB**(*X_p*, *I_max=255*)

Defines the *ROMM RGB* encoding electro-optical transfer function (EOTF / EOCF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{ROMM}.
- **I_max** (numeric, optional) – Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.

Returns Linear data X_{ROMM}.

Return type numeric or ndarray

References

- *[ANS03]*
- *[SWG00]*

Examples

```
>>> eotf_ROMMRGB(98.356413311540095)
0.1...
```

colour.models.eotf_RIMMRGB

colour.models.**eotf_RIMMRGB**(*X_p*, *I_max=255*, *E_clip=2.0*)

Defines the *RIMM RGB* encoding electro-optical transfer function (EOTF / EOCF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{RIMM}.
- **I_max** (numeric, optional) – Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **E_clip** (numeric, optional) – Maximum exposure level.

Returns Linear data X_{RIMM}.

Return type numeric or ndarray

References

- *[SWG00]*

Examples

```
>>> eotf_RIMMRGB(74.37680178131521)
0.1...
```

colour.models.eotf_ROMMRGB

colour.models.**eotf_ROMMRGB**(*X_p*, *I_max=255*)

Defines the *ROMM RGB* encoding electro-optical transfer function (EOTF / EOCF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{ROMM}.
- **I_max** (numeric, optional) – Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.

Returns Linear data X_{ROMM}.

Return type numeric or ndarray

References

- *[ANS03]*
- *[SWG00]*

Examples

```
>>> eotf_ROMMRGB(98.356413311540095)
0.1...
```

colour.models.eotf_SMPTE240M

colour.models.**eotf_SMPTE240M**(*V_r*)

Defines *SMPTE 240M* electro-optical transfer function (EOTF / EOCF).

Parameters V_r (numeric or array_like) – Video signal level V_r driving the reference reproducer normalized to the system reference white.

Returns Light output L_r from the reference reproducer normalized to the system reference white.

Return type numeric or ndarray

References

- *[SocietyoMPaTEngineers99]*

Examples

```
>>> eotf_SMPTE240M(0.402285796753870)
0.1...
```

colour.models.eotf_ST2084

colour.models.**eotf_ST2084**(*N*, *L_p=10000*)

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, normalized to the range [0, 1], 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) – Display peak luminance cd/m^2.

Returns Target optical output C in cd/m^2 of the ideal reference display.

Return type numeric or ndarray

References

- *[MDolbyLaboratories14]*
- *[SocietyoMPaTEngineers14]*

Examples

```
>>> eotf_ST2084(0.508078421517399)
100.0000000...
```

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_reverse(value[, function])	Maps relative display linear light to scene linear light using given reverse opto-optical transfer function (OOTF / OOCF).
OOTFS_REVERSE	Supported reverse 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 PQ': ..., 'ITU-R BT.2100 HLG': ...})**

Supported opto-optical transfer functions (OOTFs / OOCFs).

OOTFS [CaseInsensitiveMapping] **{'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}**

colour.ootf_reverse

colour.**ootf_reverse**(*value*, *function='ITU-R BT.2100 PQ'*, ***kwargs*)

Maps relative display linear light to scene linear light using given reverse opto-optical transfer function (OOTF / OOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}** Reverse opto-optical transfer function (OOTF / OOCF).

Other Parameters

- **L_B** (*numeric, optional*) – {colour.models.ootf_reverse_BT2100_HLG()}, L_B is the display luminance for black in cd/m^2.
- **L_W** (*numeric, optional*) – {colour.models.ootf_reverse_BT2100_HLG()}, L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (*numeric, optional*) – {colour.models.ootf_reverse_BT2100_HLG()}, System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.

Returns Luminance of scene linear light.

Return type numeric or ndarray

Examples

```
>>> ootf_reverse(779.988360834115840)
0.1000000...
>>> ootf_reverse(
...     63.095734448019336, function='ITU-R BT.2100 HLG')
0.1000000...
```

colour.OOTFS_REVERSE

colour.**OOTFS_REVERSE = CaseInsensitiveMapping({'ITU-R BT.2100 PQ': ..., 'ITU-R BT.2100 HLG': ...})**
Supported reverse opto-optical transfer functions (OOTFs / OOCFs).

OOTFS_REVERSE [CaseInsensitiveMapping] **{'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}**

colour.models

ootf_BT2100_HLG(E[, L_B, L_W, gamma])	Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).
ootf_reverse_BT2100_HLG(F_D[, L_B, L_W, gamma])	Defines *Recommendation ITU-R BT.2100 Reference HLG* reverse opto-optical transfer function (OOTF / OOCF).
ootf_BT2100_PQ(E)	Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-optical transfer function (OOTF / OOCF).
ootf_reverse_BT2100_PQ(F_D)	Defines *Recommendation ITU-R BT.2100 Reference PQ* reverse opto-optical transfer function (OOTF / OOCF).

colour.models.ootf_BT2100_HLG

colour.models.**ootf_BT2100_HLG**(*E*, *L_B=0*, *L_W=1000*, *gamma=None*)
Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).

The OOTF maps relative scene linear light to display linear light.

Parameters

- **E** (numeric or array_like) – E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure, normalized to the range [0, 1].
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2.
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.

Returns F_D is the luminance of a displayed linear component $R_D, G_D, or B_D$, in cd/m^2.

Return type numeric or ndarray

References

- *[Bor17]*

- *[InternationalTUnion16]*

Examples

```
>>> ootf_BT2100_HLG(0.1)
63.0957344...
```

colour.models.ootf_reverse_BT2100_HLG

colour.models.**ootf_reverse_BT2100_HLG**(*F_D*, *L_B=0*, *L_W=1000*, *gamma=None*)
Defines *Recommendation ITU-R BT.2100 Reference HLG* reverse opto-optical transfer function (OOTF / OOCF).

Parameters

- **F_D** (numeric or array_like) – F_D is the luminance of a displayed linear component $R_D, G_D, or B_D$, in cd/m^2.
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2.
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000 cd/m^2$.

Returns E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure, normalized to the range [0, 1].

Return type numeric or ndarray

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> ootf_reverse_BT2100_HLG(63.095734448019336)
0.1000000...
```

colour.models.ootf_BT2100_PQ

colour.models.**ootf_BT2100_PQ**(*E*)
Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-optical transfer function (OOTF / OOCF).

The OOTF maps relative scene linear light to display linear light.

Parameters **E** (numeric or array_like) – $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure. The values E, R_S, G_S, B_S, Y_S, I_S are in the range [0, 1].

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

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> ootf_BT2100_PQ(0.1)
779.9883608...
```

colour.models.ootf_reverse_BT2100_PQ

colour.models.**ootf_reverse_BT2100_PQ**(*F_D*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* reverse opto-optical transfer function (OOTF / OOCF).

Parameters F_D (numeric or array_like) – F_D is the luminance of a displayed linear component (R_D, G_D, B_D; Y_D; 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. The values E, R_S, G_S, B_S, Y_S, I_S are in the range [0, 1].

Return type numeric or ndarray

References

- *[Bor17]*
- *[InternationalTUnion16]*

Examples

```
>>> ootf_reverse_BT2100_PQ(779.988360834115840)
0.1000000...
```

Log Encoding and Decoding Curves

colour

log_encoding_curve(value[, curve])	Encodes linear-light values to $R'G'B'$ video component signal value using given *log* curve.
LOG_ENCODING_CURVES	Supported *log* encoding curves.

Continued on next page

Table 3.150 – continued from previous page

log_decoding_curve(value[, curve])	Decodes $R'G'B'$ video component signal value to linear-light values using given *log* curve.
LOG_DECODING_CURVES	Supported *log* decoding curves.

colour.log_encoding_curve

colour.**log_encoding_curve**(*value*, *curve='Cineon'*, ***kwargs*)

Encodes linear-light values to $R'G'B'$ video component signal value using given *log* curve.

Parameters

- **value** (numeric or array_like) – Value.
- **curve** (unicode, optional) – **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'ERIMM RGB', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'V-Log', 'ViperLog'}**, Computation curve.

Other Parameters

- **EI** (*int, optional*) – {colour.models.log_encoding_ALEXALogC()}, Ei.
- **E_clip** (*numeric, optional*) – {colour.models.log_encoding_ERIMMRGB()}, Maximum exposure limit.
- **E_min** (*numeric, optional*) – {colour.models.log_encoding_ERIMMRGB()}, Minimum exposure limit.
- **I_max** (*numeric, optional*) – {colour.models.log_encoding_ERIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **bit_depth** (*unicode, optional*) – {colour.models.log_encoding_ACESproxy(), colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2()}, **{8, 10, 12}**, Bit depth used for conversion, *ACESproxy* uses **{10, 12}**.
- **black_offset** (*numeric or array_like*) – {colour.models.log_encoding_Cineon(), colour.models.log_encoding_Panalog(), colour.models.log_encoding_REDLog(), colour.models.log_encoding_REDLogFilm()}, Black offset.
- **density_per_code_value** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Density per code value.
- **firmware** (*unicode, optional*) – {colour.models.log_encoding_ALEXALogC()}, **{'SUP 3.x', 'SUP 2.x'}**, Alexa firmware version.
- **in_reflection** (*bool, optional*) – {colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2()}, Whether the light level x to a camera is reflection.
- **linear_reference** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Linear reference.
- **log_reference** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Log reference.
- **out_legal** (*bool, optional*) – {colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2(), colour.models.log_encoding_SLog3()}, Whether the non-linear *Sony S-Log*, *Sony S-Log2* or *Sony S-Log3* data y is encoded in legal range.

- **negative_gamma** (*numeric or array_like*) – {`colour.models.log_encoding_PivotedLog()`}, Negative gamma.
- **method** (*unicode, optional*) – {`colour.models.log_encoding_ALEXALogC()`}, **{'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}**, Conversion method.

Returns *Log* value.

Return type numeric or ndarray

Examples

```
>>> log_encoding_curve(0.18)
0.4573196...
>>> log_encoding_curve(0.18, curve='ACEScc')
0.4135884...
>>> log_encoding_curve(0.18, curve='PLog', log_reference=400)
...
0.3910068...
>>> log_encoding_curve(0.18, curve='S-Log')
0.3849708...
```

colour.LOG_ENCODING_CURVES

`colour.LOG_ENCODING_CURVES = CaseInsensitiveMapping({'ACEScc': ..., 'ACEScct': ..., 'Log3G12': ..., 'Log3G`

Supported *log* encoding curves.

LOG_ENCODING_CURVES [CaseInsensitiveMapping] **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'ERIMM RGB', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'V-Log', 'ViperLog'}**

colour.log_decoding_curve

`colour.log_decoding_curve`(*value*, *curve='Cineon'*, ***kwargs*)

Decodes $R'G'B'$ video component signal value to linear-light values using given *log* curve.

Parameters

- **value** (`numeric or array_like`) – Value.
- **curve** (`unicode, optional`) – **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'ERIMM RGB', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'V-Log', 'ViperLog'}**, Computation curve.

Other Parameters

- **EI** (*int, optional*) – {`colour.models.log_decoding_ALEXALogC()`}, Ei.
- **E_clip** (*numeric, optional*) – {`colour.models.log_decoding_ERIMMRGB()`}, Maximum exposure limit.
- **E_min** (*numeric, optional*) – {`colour.models.log_decoding_ERIMMRGB()`}, Minimum exposure limit.

- **I_max** (*numeric, optional*) – {colour.models.log_decoding_ERIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **bit_depth** (*int, optional*) – {colour.models.log_decoding_ACESproxy(), colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2()}, **{8, 10, 12}**, Bit depth used for conversion, *ACESproxy* uses **{10, 12}**.
- **black_offset** (*numeric or array_like*) – {colour.models.log_decoding_Cineon(), colour.models.log_decoding_Panalog(), colour.models.log_decoding_REDLog(), colour.models.log_decoding_REDLogFilm()}, Black offset.
- **density_per_code_value** (*numeric or array_like*) – {colour.models.log_decoding_PivotedLog()}, Density per code value.
- **firmware** (*unicode, optional*) – {colour.models.log_decoding_ALEXALogC()}, **{'SUP 3.x', 'SUP 2.x'}**, Alexa firmware version.
- **in_legal** (*bool, optional*) – {colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2(), colour.models.log_decoding_SLog3()}, Whether the non-linear *Sony S-Log*, *Sony S-Log2* or *Sony S-Log3* data y is encoded in legal range.
- **linear_reference** (*numeric or array_like*) – {colour.models.log_decoding_PivotedLog()}, Linear reference.
- **log_reference** (*numeric or array_like*) – {colour.models.log_decoding_PivotedLog()}, Log reference.
- **negative_gamma** (*numeric or array_like*) – {colour.models.log_decoding_PivotedLog()}, Negative gamma.
- **out_reflection** (*bool, optional*) – {colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2()}, Whether the light level x to a camera is reflection.
- **method** (*unicode, optional*) – {colour.models.log_decoding_ALEXALogC()}, **{'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}**, Conversion method.

Returns *Log* value.

Return type numeric or ndarray

Examples

```
>>> log_decoding_curve(0.457319613085418)
0.1...
>>> log_decoding_curve(0.413588402492442, curve='ACEScc')
...
0.1...
>>> log_decoding_curve(0.391006842619746, curve='PLog', log_reference=400)
...
0.1...
>>> log_decoding_curve(0.376512722254600, curve='S-Log')
...
0.1...
```

colour.LOG_DECODING_CURVES

colour.**LOG_DECODING_CURVES = CaseInsensitiveMapping({'ACEScc': ..., 'ACEScct': ..., 'Log3G12': ..., 'Log3G**
Supported *log* decoding curves.

LOG_DECODING_CURVES [CaseInsensitiveMapping] **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'ERIMM RGB', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', '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.
log_encoding_ACEScct(lin_AP1)	Defines the *ACEScct* colourspace log encoding / opto-electronic transfer function.
log_decoding_ACEScct(ACEScct)	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[, bit_depth])	Defines the *ACESproxy* colourspace log decoding curve / electro-optical transfer function.
log_encoding_ALEXALogC(x[, firmware, method, EI])	Defines the *ALEXA Log C* log encoding curve / opto-electronic transfer function.
log_decoding_ALEXALogC(t[, firmware, method, EI])	Defines the *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[, I_max, E_min, E_clip])	Defines the *ERIMM RGB* log encoding curve / opto-electronic transfer function (OETF / OECF).
log_decoding_ERIMMRGB(X_p[, I_max, E_min, ...])	Defines the *ERIMM RGB* log decoding curve / electro-optical transfer function (EOTF / EOCF).
log_encoding_Log3G10(x[, legacy_curve])	Defines the *Log3G10* log encoding curve / opto-electronic transfer function.
log_decoding_Log3G10(y[, legacy_curve])	Defines the *Log3G10* log decoding curve / electro-optical transfer function.

Continued on next page

Table 3.151 – continued from previous page

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.
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, out_legal, ...])	Defines the *Sony S-Log* log encoding curve / opto-electronic transfer function.
log_decoding_SLog(y[, bit_depth, in_legal, ...])	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, in_legal, ...])	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, in_legal, ...])	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

References

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

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

References

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

Examples

```
>>> log_decoding_ACEScc(0.413588402492442)
0.1799999...
```

colour.models.log_encoding_ACEScct

colour.models.**log_encoding_ACEScct**(*lin_AP1*)

Defines the *ACEScct* colourspace log encoding / opto-electronic transfer function.

Parameters **lin_AP1** (numeric or array_like) – *lin_AP1* value.

Returns *ACEScct* non-linear value.

Return type numeric or ndarray

References

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

Examples

```
>>> log_encoding_ACEScct(0.18)
0.4135884...
```

colour.models.log_decoding_ACEScct

colour.models.**log_decoding_ACEScct**(*ACEScct*)

Defines the *ACEScct* colourspace log decoding / electro-optical transfer function.

Parameters **ACEScct** (numeric or array_like) – *ACEScct* non-linear value.

Returns *lin_AP1* value.

Return type numeric or ndarray

References

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

Examples

```
>>> log_decoding_ACEScct(0.413588402492442)
0.1799999...
```

colour.models.log_encoding_ACESproxy

colour.models.**log_encoding_ACESproxy**(*lin_AP1*, *bit_depth=10*)

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.

Returns *ACESproxy* non-linear value.

Return type numeric or ndarray

References

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

Examples

```
>>> log_encoding_ACESproxy(0.18)
426
```

colour.models.log_decoding_ACESproxy

colour.models.**log_decoding_ACESproxy**(*ACESproxy*, *bit_depth=10*)

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.

Returns *lin_AP1* value.

Return type numeric or ndarray

References

- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a]*
- *[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]*

Examples

```
>>> log_decoding_ACESproxy(426)
0.1792444...
```

colour.models.log_encoding_ALEXALogC

colour.models.**log_encoding_ALEXALogC**(*x*, *firmware=u'SUP 3.x'*, *method=u'Linear Scene Exposure Factor'*, *EI=800*)

Defines the *ALEXA Log C* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **firmware** (unicode, optional) – **{'SUP 3.x', 'SUP 2.x'}**, Alexa firmware version.
- **method** (unicode, optional) – **{'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}**, Conversion method.
- **EI** (int, optional) – Ei.

Returns *ALEXA Log C* encoded data t.

Return type numeric or ndarray

References

- *[ARR12]*

Examples

```
>>> log_encoding_ALEXALogC(0.18)
0.3910068...
```

colour.models.log_decoding_ALEXALogC

colour.models.**log_decoding_ALEXALogC**(*t*, *firmware=u'SUP 3.x'*, *method=u'Linear Scene Exposure Factor'*, *EI=800*)

Defines the *ALEXA Log C* log decoding curve / electro-optical transfer function.

Parameters

- **t** (numeric or array_like) – *ALEXA Log C* encoded data t.
- **firmware** (unicode, optional) – **{'SUP 3.x', 'SUP 2.x'}**, Alexa firmware version.
- **method** (unicode, optional) – **{'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}**, Conversion method.
- **EI** (int, optional) – Ei.

Returns Linear data x.

Return type numeric or ndarray

References

- *[ARR12]*

Examples

```
>>> log_decoding_ALEXALogC(0.391006832034084)
0.18...
```

colour.models.log_encoding_CanonLog2

colour.models.**log_encoding_CanonLog2**(*x, bit_depth=10, out_legal=True, in_reflection=True*)
Defines the *Canon Log 2* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the *Canon Log 2* non-linear data is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns *Canon Log 2* non-linear data.

Return type numeric or ndarray

References

- *[Can]*

Examples

```
>>> log_encoding_CanonLog2(0.18) * 100
39.8254694...
```

colour.models.log_decoding_CanonLog2

colour.models.**log_decoding_CanonLog2**(*clog2, bit_depth=10, in_legal=True, out_reflection=True*)
Defines the *Canon Log 2* log decoding curve / electro-optical transfer function.

Parameters

- **clog2** (numeric or array_like) – *Canon Log 2* non-linear data.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the *Canon Log 2* non-linear data is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Linear data x.

Return type numeric or ndarray

References

- *[Can]*

Examples

```
>>> log_decoding_CanonLog2(39.825469498316735 / 100)
0.1799999...
```

colour.models.log_encoding_CanonLog3

colour.models.**log_encoding_CanonLog3**(*x*, *bit_depth=10*, *out_legal=True*, *in_reflection=True*)
Defines the *Canon Log 3* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the *Canon Log 3* non-linear data is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns *Canon Log 3* non-linear data.

Return type numeric or ndarray

Notes

- Introspection of the grafting points by Shaw, N. (2018) shows that the *Canon Log 3* IDT was likely derived from its encoding curve as the later is grafted at *+/-0.014*:

```
>>> clog3 = 0.04076162
>>> (clog3 - 0.073059361) / 2.3069815
-0.014000000000000002
>>> clog3 = 0.105357102
>>> (clog3 - 0.073059361) / 2.3069815
0.013999999999999997
```

References

- *[Can]*

Examples

```
>>> log_encoding_CanonLog3(0.18) * 100
34.3389369...
```

colour.models.log_decoding_CanonLog3

colour.models.**log_decoding_CanonLog3**(*clog3*, *bit_depth=10*, *in_legal=True*, *out_reflection=True*)
Defines the *Canon Log 3* log decoding curve / electro-optical transfer function.

Parameters

- **clog3** (numeric or array_like) – *Canon Log 3* non-linear data.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the *Canon Log 3* non-linear data is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Linear data x.

Return type numeric or ndarray

References

- *[Can]*

Examples

```
>>> log_decoding_CanonLog3(34.338936938868677 / 100)
0.1800000...
```

colour.models.log_encoding_CanonLog

colour.models.**log_encoding_CanonLog**(*x*, *bit_depth=10*, *out_legal=True*, *in_reflection=True*)
Defines the *Canon Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the *Canon Log* non-linear data is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns *Canon Log* non-linear data.

Return type numeric or ndarray

References

- *[Tho12]*

Examples

```
>>> log_encoding_CanonLog(0.18) * 100
34.3389651...
```

colour.models.log_decoding_CanonLog

colour.models.**log_decoding_CanonLog**(*clog*, *bit_depth=10*, *in_legal=True*, *out_reflection=True*)
Defines the *Canon Log* log decoding curve / electro-optical transfer function.

Parameters

- **clog** (numeric or array_like) – *Canon Log* non-linear data.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the *Canon Log* non-linear data is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Linear data x.

Return type numeric or ndarray

References

- *[Tho12]*

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

References

- *[SonyImageworks12]*

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

References

- *[SonyImageworks12]*

Examples

```
>>> log_decoding_Cineon(0.457319613085418)
0.1799999...
```

colour.models.log_encoding_ERIMMRGB

`colour.models.log_encoding_ERIMMRGB(X, I_max=255, 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}.
- **I_max** (numeric, optional) – Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **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

References

- *[SWG00]*

Examples

```
>>> log_encoding_ERIMMRGB(0.18)
104.5633593...
```

colour.models.log_decoding_ERIMMRGB

colour.models.**log_decoding_ERIMMRGB**(*X_p*, *I_max=255*, *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}.
- **I_max** (numeric, optional) – Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **E_min** (numeric, optional) – Minimum exposure limit.
- **E_clip** (numeric, optional) – Maximum exposure limit.

Returns Linear data X_{ERIMM}.

Return type numeric or ndarray

References

- *[SWG00]*

Examples

```
>>> log_decoding_ERIMMRGB(104.56335932049294)
0.1...
```

colour.models.log_encoding_Log3G10

colour.models.**log_encoding_Log3G10**(*x*, *legacy_curve=False*)
Defines the *Log3G10* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **legacy_curve** (bool, optional) – Whether to use the v1 *Log3G10* log encoding curve. Default is *False*.

Returns Non-linear data y.

Return type numeric or ndarray

Notes

- The v1 *Log3G10* log encoding curve is the one used in *REDCINE-X beta 42*. *Resolve 12.5.2* also uses the v1 curve. *RED* is planning to use v2 *Log3G10* log encoding curve in the release version of the *RED SDK*. Use the *legacy_curve=True* argument to switch to the v1 curve for compatibility with the current (as of September 21, 2016) *RED SDK*.
- The intent of the v1 *Log3G10* log encoding curve is that zero maps to zero, 0.18 maps to 1/3, and 10 stops above 0.18 maps to 1.0. The name indicates this in a similar way to the naming conventions of *Sony HyperGamma* curves.

  The constants used in the functions do not in fact quite hit these values, but rather than use corrected constants, the functions here use the official *RED* values, in order to match the output of the *RED SDK*.

  For those interested, solving for constants which exactly hit 1/3 and 1.0 yields the following values:

```
B = 25 * (np.sqrt(4093.0) - 3) / 9
A = 1 / np.log10(B * 184.32 + 1)
```

  where the function takes the form:

```
Log3G10(x) = A * np.log10(B * x + 1)
```

  Similarly for *Log3G12*, the values which hit exactly 1/3 and 1.0 are:

```
B = 25 * (np.sqrt(16381.0) - 3) / 9
A = 1 / np.log10(B * 737.28 + 1)
```

References

- *[Nat16]*

Examples

```
>>> log_encoding_Log3G10(0.18, legacy_curve=True)
0.3333336...
>>> log_encoding_Log3G10(0.0)
0.0915514...
```

colour.models.log_decoding_Log3G10

`colour.models.`**`log_decoding_Log3G10`**(*y*, *legacy_curve=False*)

Defines the *Log3G10* log decoding curve / electro-optical transfer function.

Parameters

- **y** (`numeric or array_like`) – Non-linear data y.
- **legacy_curve** (`bool, optional`) – Whether to use the v1 *Log3G10* log encoding curve. Default is *False*.

Returns Linear data x.

Return type numeric or ndarray

References

- *[Nat16]*

Examples

```
>>> log_decoding_Log3G10(1.0 / 3, legacy_curve=True)
0.1799994...
>>> log_decoding_Log3G10(1.0)
184.3223476...
```

colour.models.log_encoding_Log3G12

colour.models.**log_encoding_Log3G12**(x)

Defines the *Log3G12* log encoding curve / opto-electronic transfer function.

Parameters **x** (numeric or array_like) – Linear data x.

Returns Non-linear data y.

Return type numeric or ndarray

References

- *[Nat16]*

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

References

- *[Nat16]*

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.

References

- *[SonyImageworks12]*

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.

References

- *[SonyImageworks12]*

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

References

- *[SonyImageworks12]*

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

References

- *[SonyImageworks12]*

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

References

- *[GDM16]*

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

References

- *[GDM16]*

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

References

- *[SonyImageworks12]*

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

References

- *[SonyImageworks12]*

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

References

- *[SonyImageworks12]*

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

References

- *[SonyImageworks12]*

Examples

```
>>> log_decoding_REDLogFilm(0.457319613085418)
0.1799999...
```

colour.models.log_encoding_SLog

colour.models.**log_encoding_SLog**(*x*, *bit_depth=10*, *out_legal=True*, *in_reflection=True*)
Defines the *Sony S-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Reflection or $IRE/100$ input light level x to a camera.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the non-linear *Sony S-Log* data y is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Non-linear *Sony S-Log* data y.

Return type numeric or ndarray

References

- *[SonyCorporation12]*

Examples

```
>>> log_encoding_SLog(0.18)
0.3849708...
>>> log_encoding_SLog(0.18, out_legal=False)
0.3765127...
>>> log_encoding_SLog(0.18, in_reflection=False)
0.3708204...
```

colour.models.log_decoding_SLog

colour.models.**log_decoding_SLog**(*y*, *bit_depth=10*, *in_legal=True*, *out_reflection=True*)
Defines the *Sony S-Log* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear *Sony S-Log* data y.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the non-linear *Sony S-Log* data y is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Reflection or $IRE/100$ input light level x to a camera.

Return type numeric or ndarray

References

- *[SonyCorporation12]*

Examples

```
>>> log_decoding_SLog(0.384970815928670)
0.1...
>>> log_decoding_SLog(0.376512722254600, in_legal=False)
...
0.1...
>>> log_decoding_SLog(0.370820482371268, out_reflection=False)
...
0.1...
```

colour.models.log_encoding_SLog2

colour.models.**log_encoding_SLog2**(*x, bit_depth=10, out_legal=True, in_reflection=True*)

Defines the *Sony S-Log2* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Reflection or $IRE/100$ input light level x to a camera.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the non-linear *Sony S-Log2* data y is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Non-linear *Sony S-Log2* data y.

Return type numeric or ndarray

References

- *[SonyCorporation12]*

Examples

```
>>> log_encoding_SLog2(0.18)
0.3395325...
>>> log_encoding_SLog2(0.18, out_legal=False)
0.3234495...
>>> log_encoding_SLog2(0.18, in_reflection=False)
0.3262865...
```

colour.models.log_decoding_SLog2

colour.models.**log_decoding_SLog2**(*y*, *bit_depth=10*, *in_legal=True*, *out_reflection=True*)
Defines the *Sony S-Log2* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear *Sony S-Log2* data y.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the non-linear *Sony S-Log2* data y is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Reflection or $IRE/100$ input light level x to a camera.

Return type numeric or ndarray

References

- *[SonyCorporation12]*

Examples

```
>>> log_decoding_SLog2(0.339532524633774)
0.1...
>>> log_decoding_SLog2(0.323449512215013, in_legal=False)
...
0.1...
>>> log_decoding_SLog2(0.326286538946799, out_reflection=False)
...
0.1...
```

colour.models.log_encoding_SLog3

colour.models.**log_encoding_SLog3**(*x*, *bit_depth=10*, *out_legal=True*, *in_reflection=True*)
Defines the *Sony S-Log3* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Reflection or $IRE/100$ input light level x to a camera.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the non-linear *Sony S-Log3* data y is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Non-linear *Sony S-Log3* data y.

Return type numeric or ndarray

References

- *[SonyCorporationc]*

Examples

```
>>> log_encoding_SLog3(0.18)
0.4105571...
>>> log_encoding_SLog3(0.18, out_legal=False)
0.4063926...
>>> log_encoding_SLog3(0.18, in_reflection=False)
0.3995079...
```

colour.models.log_decoding_SLog3

`colour.models.log_decoding_SLog3`(*y*, *bit_depth=10*, *in_legal=True*, *out_reflection=True*)

Defines the *Sony S-Log3* log decoding curve / electro-optical transfer function.

Parameters

- **y** (`numeric or array_like`) – Non-linear *Sony S-Log3* data y.
- **bit_depth** (`int, optional`) – Bit depth used for conversion.
- **in_legal** (`bool, optional`) – Whether the non-linear *Sony S-Log3* data y is encoded in legal range.
- **out_reflection** (`bool, optional`) – Whether the light level x to a camera is reflection.

Returns Reflection or $IRE/100$ input light level x to a camera.

Return type numeric or ndarray

References

- *[SonyCorporationc]*

Examples

```
>>> log_decoding_SLog3(0.410557184750733)
0.1...
>>> log_decoding_SLog3(0.406392694063927, in_legal=False)
...
0.1...
>>> log_decoding_SLog3(0.399507939606216, out_reflection=False)
...
0.1...
```

colour.models.log_encoding_VLog

colour.models.**log_encoding_VLog**(*L_in*, *bit_depth=10*, *out_legal=True*, *in_reflection=True*)
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_legal** (bool, optional) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level :math'L_{in}' to a camera is reflection.

Returns Non-linear data V_{out}.

Return type numeric or ndarray

References

- *[Pan14]*

Examples

```
>>> log_encoding_VLog(0.18)
0.4233114...
```

colour.models.log_decoding_VLog

colour.models.**log_decoding_VLog**(*V_out*, *bit_depth=10*, *in_legal=True*, *out_reflection=True*)
Defines the *Panasonic V-Log* log decoding curve / electro-optical transfer function.

Parameters

- **V_out** (numeric or array_like) – Non-linear data V_{out}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level :math'L_{in}' to a camera is reflection.

Returns Linear reflection data :math'L_{in}'.

Return type numeric or ndarray

References

- *[Pan14]*

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

References

- *[SonyImageworks12]*

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

References

- *[SonyImageworks12]*

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.
`YCBCR_WEIGHTS`	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.YCBCR_WEIGHTS` for presets. Default is *(0.2126, 0.0722)*, the weightings for *ITU-R BT.709*.
- **in_bits** (`int, optional`) – Bit depth for integer input, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Default is *10*.
- **in_legal** (`bool, optional`) – Whether to treat the input values as legal range. Default is *False*.
- **in_int** (`bool, optional`) – Whether to treat the input values as `in_bits` integer code values. Default is *False*.
- **out_bits** (`int, optional`) – Bit depth for integer output, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Ignored if `out_legal` and `out_int` are both *False*. Default is *8*.
- **out_legal** (`bool, optional`) – Whether to return legal range values. Default is *True*.
- **out_int** (`bool, optional`) – Whether to return values as `out_bits` integer code values. Default is *False*.

Other Parameters

- **in_range** (*array_like, optional*) – Array overriding the computed range such as *in_range = (RGB_min, RGB_max)*. If `in_range` is undefined, *RGB_min* and *RGB_max* will be computed using `colour.CV_range()` definition.

- **out_range** (*array_like, optional*) – Array overriding the computed range such as *out_range = (Y_min, Y_max, C_min, C_max)`. If "out_range" is undefined, *Y_min, Y_max*, *C_min* and *C_max* will be computed using `colour.models.rgb.ycbcr.YCbCr_ranges()` definition.

Returns *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 *[InternationalTUnion15a]*.

Notes

- 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 in range [0, 1] (`in_bits` is ignored) to a float *Y'CbCr* output array where *Y'* is in range [16 / 255, 235 / 255] and *Cb* and *Cr* are in range [16 / 255, 240./255]. The float values are calculated based on an [0, 255] integer range, but no 8-bit quantisation or clamping are performed.

References

- *[InternationalTUnion11c]*
- *[InternationalTUnion15b]*
- *[SocietyoMPaTEngineers99]*
- *[Wik|]*

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 *[InternationalTUnion11c]*:

```
>>> RGB = np.array([102, 0, 51])
>>> RGB_to_YCbCr(RGB, K=YCBCR_WEIGHTS['ITU-R BT.601'], in_range=(0, 255),
...              out_range=(0, 255, 0, 256), out_int=True)
array([ 36, 136, 175])
```

Note the use of 256 for the max *Cb / Cr* value, which is required so that the *Cb* and *Cr* output is centered about 128. Using 255 centres it about 127.5, meaning that there is no integer code value to represent achromatic colours. This does however create the possibility of output integer codes with value of 256, which cannot be stored in 8-bit integer representation. Recommendation ITU-T T.871 specifies these should be clamped to 255.

These JFIF JPEG ranges are also obtained as follows:

```
>>> RGB_to_YCbCr(RGB, K=YCBCR_WEIGHTS['ITU-R BT.601'], in_bits=8,
...              in_int=True, out_legal=False, out_int=True)
array([ 36, 136, 175])
```

colour.YCbCr_to_RGB

colour.**YCbCr_to_RGB**(*YCbCr, K=array([0.2126, 0.0722]), in_bits=8, in_legal=True, in_int=False, out_bits=10, out_legal=False, out_int=False, **kwargs*)

Converts an array of *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.YCBCR_WEIGHTS` for presets. Default is *(0.2126, 0.0722)*, the weightings for *ITU-R BT.709*.
- **in_bits** (int, optional) – Bit depth for integer input, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Default is *10*.
- **in_legal** (bool, optional) – Whether to treat the input values as legal range. Default is *False*.
- **in_int** (bool, optional) – Whether to treat the input values as `in_bits` integer code values. Default is *False*.
- **out_bits** (int, optional) – Bit depth for integer output, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Ignored if `out_legal` and `out_int` are both *False*. Default is *8*.
- **out_legal** (bool, optional) – Whether to return legal range values. Default is *True*.
- **out_int** (bool, optional) – Whether to return values as `out_bits` integer code values. Default is *False*.

Other Parameters

- **in_range** (*array_like, optional*) – Array overriding the computed range such as *in_range = (Y_min, Y_max, C_min, C_max)*. If `in_range` is undefined, *Y_min*, *Y_max*, *C_min* and *C_max* will be computed using `colour.models.rgb.ycbcr.YCbCr_ranges()` definition.
- **out_range** (*array_like, optional*) – Array overriding the computed range such as *out_range = (RGB_min, RGB_max)*. If `out_range` is undefined, *RGB_min* and *RGB_max* will be computed using `colour.CV_range()` definition.

Returns *R'G'B'* array of integer or float values.

Return type ndarray

> **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 *[InternationalTUnion15a]*.

References

- *[InternationalTUnion11c]*
- *[InternationalTUnion15b]*
- *[SocietyoMPaTEngineers99]*
- *[Wik|]*

Examples

```
>>> YCbCr = np.array([502, 512, 512])
>>> YCbCr_to_RGB(YCbCr, in_bits=10, in_legal=True, in_int=True)
array([ 0.5,  0.5,  0.5])
```

colour.YCBCR_WEIGHTS

`colour.`**`YCBCR_WEIGHTS = CaseInsensitiveMapping({u'ITU-R BT.2020': ..., u'ITU-R BT.601': ..., u'ITU-R BT.709'`**

Implements a case-insensitive mutable mapping / *dict* object.

Allows values retrieving from keys while ignoring the key case. The keys are expected to be unicode or string-like objects supporting the `str.lower()` method.

Parameters **`data`** (`dict`) – *dict* of data to store into the mapping at initialisation.

Other Parameters ****kwargs** (*dict, optional*) – Key / Value pairs to store into the mapping at initialisation.

`colour.`**`__setitem__`**`()`

`colour.`**`__getitem__`**`()`

`colour.`**`__delitem__`**`()`

`colour.`**`__contains__`**`()`

`colour.`**`__iter__`**`()`

colour.**__len__**()

colour.**__eq__**()

colour.**__ne__**()

colour.**__repr__**()

colour.**copy**()

colour.**lower_items**()

Warning: The keys are expected to be unicode or string-like objects.

References

- *[Rei]*

Examples

```
>>> methods = CaseInsensitiveMapping({'McCamy': 1, 'Hernandez': 2})
>>> methods['mccamy']
1
```

colour.RGB_to_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*.

Other Parameters **out_range** (*array_like, optional*) – Array overriding the computed range such as *out_range = (Y_min, Y_max, C_min, C_max)*. If out_range is undefined, *Y_min*, *Y_max*, *C_min* and *C_max* will be computed using colour.models.rgb.ycbcr.YCbCr_ranges() definition.

Returns *Yc'Cbc'Crc'* colour encoding array of integer or float values.

Return type ndarray

Warning: This definition is specifically for usage with *Recommendation ITU-R BT.2020* when adopting the constant luminance implementation.

References

- *[InternationalTUnion15a]*
- *[Wik|]*

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*.

Other Parameters **in_range** (*array_like, optional*) – Array overriding the computed range such as *in_range = (Y_min, Y_max, C_min, C_max)*. If `in_range` is undefined, *Y_min*, *Y_max*, *C_min* and *C_max* will be computed using `colour.models.rgb.ycbcr.YCbCr_ranges()` definition.

Returns *RGB* array of linear float values.

Return type ndarray

Warning: This definition is specifically for usage with *Recommendation ITU-R BT.2020* when adopting the constant luminance implementation.

References

- *[InternationalTUnion15a]*
- *[Wik|]*

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.])
```

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.

Returns IC_TC_P colour encoding array.

Return type ndarray

References

- *[Dol16]*
- *[LPY+16]*

Examples

```
>>> RGB = np.array([0.35181454, 0.26934757, 0.21288023])
>>> RGB_to_ICTCP(RGB)
array([ 0.0955407..., -0.0089063...,  0.0138928...])
```

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.

Returns *ITU-R BT.2020* colourspace array.

Return type ndarray

References

- *[Dol16]*
- *[LPY+16]*

Examples

```
>>> ICTCP = np.array([0.09554079, -0.00890639, 0.01389286])
>>> ICTCP_to_RGB(ICTCP)
array([ 0.3518145...,  0.2693475...,  0.2128802...])
```

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

References

- *[SH15]*

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

References

- *[SH15]*

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

- Input *RGB* colourspace array is in domain [0, 1].
- Output *HSV* colourspace array is in range [0, 1].

References

- *[Eash]*
- *[Smi78]*
- *[Wikj]*

Examples

```
>>> RGB = np.array([0.49019608, 0.98039216, 0.25098039])
>>> RGB_to_HSV(RGB)
array([ 0.2786738...,  0.744      ,  0.98039216])
```

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

- Input *HSV* colourspace array is in domain [0, 1].
- Output *RGB* colourspace array is in range [0, 1].

References

- *[Ease]*
- *[Smi78]*
- *[Wikj]*

Examples

```
>>> HSV = np.array([0.27867384, 0.74400000, 0.98039216])
>>> HSV_to_RGB(HSV)
array([ 0.4901960...,  0.9803921...,  0.2509803...])
```

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

- Input *RGB* colourspace array is in domain [0, 1].
- Output *HSL* colourspace array is in range [0, 1].

References

- *[Easg]*
- *[Smi78]*
- *[Wikj]*

Examples

```
>>> RGB = np.array([0.49019608, 0.98039216, 0.25098039])
>>> RGB_to_HSL(RGB)
array([ 0.2786738...,  0.9489796...,  0.6156862...])
```

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

- Input *HSL* colourspace array is in domain [0, 1].
- Output *RGB* colourspace array is in range [0, 1].

References

- *[Easd]*
- *[Smi78]*
- *[Wikj]*

Examples

```
>>> HSL = np.array([0.27867384, 0.94897959, 0.61568627])
>>> HSL_to_RGB(HSL)
array([ 0.4901960...,  0.9803921...,  0.2509803...])
```

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

- Input *RGB* colourspace array is in domain [0, 1].
- Output *CMY* colourspace array is in range [0, 1].

References

- *[Easf]*

Examples

```
>>> RGB = np.array([0.49019608, 0.98039216, 0.25098039])
>>> RGB_to_CMY(RGB)
array([ 0.5098039...,  0.0196078...,  0.7490196...])
```

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

- Input *CMY* colourspace array is in domain [0, 1].
- Output *RGB* colourspace array is in range [0, 1].

References

- *[Easb]*

Examples

```
>>> CMY = np.array([0.50980392, 0.01960784, 0.74901961])
>>> CMY_to_RGB(CMY)
array([ 0.4901960...,  0.9803921...,  0.2509803...])
```

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

- Input *CMY* colourspace array is in domain [0, 1].
- Output*CMYK* colourspace array is in range [0, 1].

References

- *[Easa]*

Examples

```
>>> CMY = np.array([0.50980392, 0.01960784, 0.74901961])
>>> CMY_to_CMYK(CMY)
array([ 0.5       ,  0.        ,  0.744     ,  0.0196078...])
```

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

- Input *CMYK* colourspace array is in domain [0, 1].
- Output *CMY* colourspace array is in range [0, 1].

References

- *[Easc]*

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

POINTER_GAMUT_BOUNDARIES	
POINTER_GAMUT_DATA	
POINTER_GAMUT_ILLUMINANT	

colour.POINTER_GAMUT_BOUNDARIES

colour.**POINTER_GAMUT_BOUNDARIES = array([[0.659, 0.316], [0.634, 0.351], [0.594, 0.391], [0.557, 0.427],**

colour.POINTER_GAMUT_DATA

colour.**POINTER_GAMUT_DATA = array([[15, 10, 0], [15, 15, 10], [15, 14, 20], ..., [90, 9, 330], [90, 4, 3**

colour.POINTER_GAMUT_ILLUMINANT

colour.**POINTER_GAMUT_ILLUMINANT = array([0.31005673, 0.3161457])**

Colour Notation Systems

Munsell Renotation System

`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`) – *Munsell* colour.

Returns *CIE xyY* colourspace array.

Return type ndarray, (3,)

Notes

- Output *CIE xyY* colourspace array is in range [0, 1].

References

- *[Cen]*
- *[Cen12]*

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   ,  0.746134...])
```

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

- Input *CIE xyY* colourspace array is in domain [0, 1].

References

- *[Cen]*
- *[Cen12]*

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({'real': ..., 'all': ..., 'Munsell Colours Real': ..., '192**

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 computations methods.

colour.munsell_value

colour.**munsell_value**(*Y*, *method=u'ASTM D1535-08'*)

Returns the *Munsell* value V of given *luminance* Y using given method.

Parameters

- **Y** (numeric or array_like) – *luminance* Y.
- **method** (unicode, optional) – **{'ASTM D1535-08', 'Priest 1920', 'Munsell 1933', 'Moon 1943', 'Saunderson 1944', 'Ladd 1955', 'McCamy 1987'}**, Computation method.

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

- Input Y is in domain [0, 100].
- Output V is in range [0, 10].

References

- *[ASTMInternational89]*
- *[Wikn]*

Examples

```
>>> munsell_value(10.08)
3.7344764...
>>> munsell_value(10.08, method='Priest 1920')
3.1749015...
>>> munsell_value(10.08, method='Munsell 1933')
3.7918355...
>>> munsell_value(10.08, method='Moon 1943')
3.7462971...
>>> munsell_value(10.08, method='Saunderson 1944')
3.6865080...
>>> munsell_value(10.08, method='Ladd 1955')
3.6952862...
>>> munsell_value(10.08, method='McCamy 1987')
array(3.7347235...)
```

colour.MUNSELL_VALUE_METHODS

colour.**MUNSELL_VALUE_METHODS = CaseInsensitiveMapping({u'Saunderson 1944': ..., u'McCamy 1987': ..., u'Ladd**

Supported *Munsell* value computations methods.

References

- *[ASTMInternational89]*
- *[Wikn]*

MUNSELL_VALUE_METHODS [CaseInsensitiveMapping] **{'Priest 1920', 'Munsell 1933', 'Moon 1943', 'Saunderson 1944', 'Ladd 1955', 'McCamy 1987', 'ASTM D1535-08'}**

Aliases:

- 'astm2008': 'ASTM D1535-08'

Priest, Gibson and MacNicholas (1920)

`colour.notation`

munsell_value_Priest1920(Y)	Returns the *Munsell* value V of given *luminance* Y using *Priest et alii (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 alii (1920)* method.

Parameters **Y** (numeric or array_like) – *luminance* Y.

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

- Input *Y* is in domain [0, 100].
- Output *V* is in range [0, 10].

References

- *[Wikn]*

Examples

```
>>> munsell_value_Priest1920(10.08)
3.1749015...
```

Munsell, Sloan and Godlove (1933)

`colour.notation`

munsell_value_Munsell1933(Y)	Returns the *Munsell* value V of given *luminance* Y using *Munsell et alii (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 alii (1933)* method.

Parameters **Y** (numeric or array_like) – *luminance* Y.

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

- Input Y is in domain [0, 100].
- Output V is in range [0, 10].

References

- *[Wikn]*

Examples

```
>>> munsell_value_Munsell1933(10.08)
3.7918355...
```

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

- Input Y is in domain [0, 100].
- Output V is in range [0, 10].

References

- [Wikn]

Examples

```
>>> munsell_value_Moon1943(10.08)
3.7462971...
```

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

- Input Y is in domain [0, 100].
- Output V is in range [0, 10].

References

- [Wikn]

Examples

```
>>> munsell_value_Saunderson1944(10.08)
3.6865080...
```

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

- Input Y is in domain [0, 100].
- Output V is in range [0, 10].

References

- *[Wikn]*

Examples

```
>>> munsell_value_Ladd1955(10.08)
3.6952862...
```

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

- Input Y is in domain [0, 100].
- Output V is in range [0, 10].

References

- *[ASTMInternational89]*

Examples

```
>>> munsell_value_McCamy1987(10.08)
array(3.7347235...)
```

ASTM D1535-08e1

colour.notation

munsell_value_ASTMD153508(Y)	Returns the *Munsell* value V of given *luminance* Y using a reverse lookup table from *ASTM D1535-08e1* method.

colour.notation.munsell_value_ASTMD153508

colour.notation.**munsell_value_ASTMD153508**(Y)

Returns the *Munsell* value V of given *luminance* Y using a reverse lookup table from *ASTM D1535-08e1* method.

Parameters **Y** (numeric or array_like) – *luminance* Y

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

- Input Y is in domain [0, 100].
- Output V is in range [0, 10].

References

- *[ASTMInternational89]*

Examples

```
>>> munsell_value_ASTMD153508(10.1488096782)
3.7462971...
```

Hexadecimal Triplet Notation

colour.notation

RGB_to_HEX(RGB)	Converts from *RGB* colourspace to hexadecimal triplet representation.
HEX_to_RGB(HEX)	Converts from hexadecimal triplet representation to *RGB* colourspace.

colour.notation.RGB_to_HEX

colour.notation.**RGB_to_HEX**(*RGB*)

Converts from *RGB* colourspace to hexadecimal triplet representation.

Parameters **RGB** (array_like) – *RGB* colourspace array.

Returns Hexadecimal triplet representation.

Return type unicode

Notes

- Input *RGB* colourspace array is in domain [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 triplet representation to *RGB* colourspace.

Parameters **HEX** (unicode or array_like) – Hexadecimal triplet representation.

Returns *RGB* colourspace array.

Return type ndarray

Notes

- Output *RGB* colourspace array is in range [0, 1].

Examples

```
>>> HEX = '#aaddff'
>>> HEX_to_RGB(HEX)
array([ 0.6666666...,  0.8666666...,  1.        ])
```

Optical Phenomena

Rayleigh Scattering

`colour`

`rayleigh_scattering`(wavelength[, ...])	Returns the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).
`rayleigh_scattering_spd`([shape, ...])	Returns the *Rayleigh* spectral power 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

- *[BWDS99]*
- *[Wiku]*

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8)
0.1004070...
```

colour.rayleigh_scattering_spd

colour.**rayleigh_scattering_spd**(*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 power distribution for given spectral shape.

Parameters

- **shape** (SpectralShape, optional) – Spectral shape used to create the *Rayleigh* scattering spectral power 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 power distribution.

Return type *SpectralPowerDistribution*

References

- *[BWDS99]*
- *[Wiku]*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     rayleigh_scattering_spd()
SpectralPowerDistribution([[ 360.        ,    0.5991013...],
                           [ 361.        ,    0.5921706...],
                           [ 362.        ,    0.5853410...],
                           [ 363.        ,    0.5786105...],
                           [ 364.        ,    0.5719774...],
                           [ 365.        ,    0.5654401...],
                           [ 366.        ,    0.5589968...],
                           [ 367.        ,    0.5526460...],
                           [ 368.        ,    0.5463860...],
                           [ 369.        ,    0.5402153...],
                           [ 370.        ,    0.5341322...],
                           [ 371.        ,    0.5281354...],
                           [ 372.        ,    0.5222234...],
                           [ 373.        ,    0.5163946...],
                           [ 374.        ,    0.5106476...],
                           [ 375.        ,    0.5049812...],
                           [ 376.        ,    0.4993939...],
                           [ 377.        ,    0.4938844...],
                           [ 378.        ,    0.4884513...],
                           [ 379.        ,    0.4830934...],
                           [ 380.        ,    0.4778095...],
                           [ 381.        ,    0.4725983...],
                           [ 382.        ,    0.4674585...],
                           [ 383.        ,    0.4623891...],
                           [ 384.        ,    0.4573889...],
                           [ 385.        ,    0.4524566...],
                           [ 386.        ,    0.4475912...],
                           [ 387.        ,    0.4427917...],
                           [ 388.        ,    0.4380568...],
                           [ 389.        ,    0.4333856...],
                           [ 390.        ,    0.4287771...],
                           [ 391.        ,    0.4242302...],
                           [ 392.        ,    0.4197439...],
                           [ 393.        ,    0.4153172...],
```

```
                [ 394.        ,    0.4109493...],
                [ 395.        ,    0.4066391...],
                [ 396.        ,    0.4023857...],
                [ 397.        ,    0.3981882...],
                [ 398.        ,    0.3940458...],
                [ 399.        ,    0.3899576...],
                [ 400.        ,    0.3859227...],
                [ 401.        ,    0.3819402...],
                [ 402.        ,    0.3780094...],
                [ 403.        ,    0.3741295...],
                [ 404.        ,    0.3702996...],
                [ 405.        ,    0.366519 ...],
                [ 406.        ,    0.3627868...],
                [ 407.        ,    0.3591025...],
                [ 408.        ,    0.3554651...],
                [ 409.        ,    0.3518740...],
                [ 410.        ,    0.3483286...],
                [ 411.        ,    0.344828 ...],
                [ 412.        ,    0.3413716...],
                [ 413.        ,    0.3379587...],
                [ 414.        ,    0.3345887...],
                [ 415.        ,    0.3312609...],
                [ 416.        ,    0.3279747...],
                [ 417.        ,    0.3247294...],
                [ 418.        ,    0.3215245...],
                [ 419.        ,    0.3183593...],
                [ 420.        ,    0.3152332...],
                [ 421.        ,    0.3121457...],
                [ 422.        ,    0.3090962...],
                [ 423.        ,    0.3060841...],
                [ 424.        ,    0.3031088...],
                [ 425.        ,    0.3001699...],
                [ 426.        ,    0.2972668...],
                [ 427.        ,    0.2943989...],
                [ 428.        ,    0.2915657...],
                [ 429.        ,    0.2887668...],
                [ 430.        ,    0.2860017...],
                [ 431.        ,    0.2832697...],
                [ 432.        ,    0.2805706...],
                [ 433.        ,    0.2779037...],
                [ 434.        ,    0.2752687...],
                [ 435.        ,    0.2726650...],
                [ 436.        ,    0.2700922...],
                [ 437.        ,    0.2675500...],
                [ 438.        ,    0.2650377...],
                [ 439.        ,    0.2625551...],
                [ 440.        ,    0.2601016...],
                [ 441.        ,    0.2576770...],
                [ 442.        ,    0.2552807...],
                [ 443.        ,    0.2529124...],
                [ 444.        ,    0.2505716...],
                [ 445.        ,    0.2482581...],
                [ 446.        ,    0.2459713...],
                [ 447.        ,    0.2437110...],
                [ 448.        ,    0.2414768...],
                [ 449.        ,    0.2392683...],
                [ 450.        ,    0.2370851...],
                [ 451.        ,    0.2349269...],
```

```
                   [ 452.        ,    0.2327933...],
                   [ 453.        ,    0.2306841...],
                   [ 454.        ,    0.2285989...],
                   [ 455.        ,    0.2265373...],
                   [ 456.        ,    0.2244990...],
                   [ 457.        ,    0.2224838...],
                   [ 458.        ,    0.2204912...],
                   [ 459.        ,    0.2185211...],
                   [ 460.        ,    0.2165730...],
                   [ 461.        ,    0.2146467...],
                   [ 462.        ,    0.2127419...],
                   [ 463.        ,    0.2108583...],
                   [ 464.        ,    0.2089957...],
                   [ 465.        ,    0.2071536...],
                   [ 466.        ,    0.2053320...],
                   [ 467.        ,    0.2035304...],
                   [ 468.        ,    0.2017487...],
                   [ 469.        ,    0.1999865...],
                   [ 470.        ,    0.1982436...],
                   [ 471.        ,    0.1965198...],
                   [ 472.        ,    0.1948148...],
                   [ 473.        ,    0.1931284...],
                   [ 474.        ,    0.1914602...],
                   [ 475.        ,    0.1898101...],
                   [ 476.        ,    0.1881779...],
                   [ 477.        ,    0.1865633...],
                   [ 478.        ,    0.1849660...],
                   [ 479.        ,    0.1833859...],
                   [ 480.        ,    0.1818227...],
                   [ 481.        ,    0.1802762...],
                   [ 482.        ,    0.1787463...],
                   [ 483.        ,    0.1772326...],
                   [ 484.        ,    0.1757349...],
                   [ 485.        ,    0.1742532...],
                   [ 486.        ,    0.1727871...],
                   [ 487.        ,    0.1713365...],
                   [ 488.        ,    0.1699011...],
                   [ 489.        ,    0.1684809...],
                   [ 490.        ,    0.1670755...],
                   [ 491.        ,    0.1656848...],
                   [ 492.        ,    0.1643086...],
                   [ 493.        ,    0.1629468...],
                   [ 494.        ,    0.1615991...],
                   [ 495.        ,    0.1602654...],
                   [ 496.        ,    0.1589455...],
                   [ 497.        ,    0.1576392...],
                   [ 498.        ,    0.1563464...],
                   [ 499.        ,    0.1550668...],
                   [ 500.        ,    0.1538004...],
                   [ 501.        ,    0.1525470...],
                   [ 502.        ,    0.1513063...],
                   [ 503.        ,    0.1500783...],
                   [ 504.        ,    0.1488628...],
                   [ 505.        ,    0.1476597...],
                   [ 506.        ,    0.1464687...],
                   [ 507.        ,    0.1452898...],
                   [ 508.        ,    0.1441228...],
                   [ 509.        ,    0.1429675...],
```

```
[ 510.        ,    0.1418238...],
[ 511.        ,    0.1406916...],
[ 512.        ,    0.1395707...],
[ 513.        ,    0.1384610...],
[ 514.        ,    0.1373624...],
[ 515.        ,    0.1362747...],
[ 516.        ,    0.1351978...],
[ 517.        ,    0.1341316...],
[ 518.        ,    0.1330759...],
[ 519.        ,    0.1320306...],
[ 520.        ,    0.1309956...],
[ 521.        ,    0.1299707...],
[ 522.        ,    0.1289559...],
[ 523.        ,    0.1279511...],
[ 524.        ,    0.1269560...],
[ 525.        ,    0.1259707...],
[ 526.        ,    0.1249949...],
[ 527.        ,    0.1240286...],
[ 528.        ,    0.1230717...],
[ 529.        ,    0.1221240...],
[ 530.        ,    0.1211855...],
[ 531.        ,    0.1202560...],
[ 532.        ,    0.1193354...],
[ 533.        ,    0.1184237...],
[ 534.        ,    0.1175207...],
[ 535.        ,    0.1166263...],
[ 536.        ,    0.1157404...],
[ 537.        ,    0.1148630...],
[ 538.        ,    0.1139939...],
[ 539.        ,    0.1131331...],
[ 540.        ,    0.1122804...],
[ 541.        ,    0.1114357...],
[ 542.        ,    0.1105990...],
[ 543.        ,    0.1097702...],
[ 544.        ,    0.1089492...],
[ 545.        ,    0.1081358...],
[ 546.        ,    0.1073301...],
[ 547.        ,    0.1065319...],
[ 548.        ,    0.1057411...],
[ 549.        ,    0.1049577...],
[ 550.        ,    0.1041815...],
[ 551.        ,    0.1034125...],
[ 552.        ,    0.1026507...],
[ 553.        ,    0.1018958...],
[ 554.        ,    0.1011480...],
[ 555.        ,    0.1004070...],
[ 556.        ,    0.0996728...],
[ 557.        ,    0.0989453...],
[ 558.        ,    0.0982245...],
[ 559.        ,    0.0975102...],
[ 560.        ,    0.0968025...],
[ 561.        ,    0.0961012...],
[ 562.        ,    0.0954062...],
[ 563.        ,    0.0947176...],
[ 564.        ,    0.0940352...],
[ 565.        ,    0.0933589...],
[ 566.        ,    0.0926887...],
[ 567.        ,    0.0920246...],
```

```
                    [ 568.        ,    0.0913664...],
                    [ 569.        ,    0.0907141...],
                    [ 570.        ,    0.0900677...],
                    [ 571.        ,    0.0894270...],
                    [ 572.        ,    0.0887920...],
                    [ 573.        ,    0.0881627...],
                    [ 574.        ,    0.0875389...],
                    [ 575.        ,    0.0869207...],
                    [ 576.        ,    0.0863079...],
                    [ 577.        ,    0.0857006...],
                    [ 578.        ,    0.0850986...],
                    [ 579.        ,    0.0845019...],
                    [ 580.        ,    0.0839104...],
                    [ 581.        ,    0.0833242...],
                    [ 582.        ,    0.0827430...],
                    [ 583.        ,    0.082167 ...],
                    [ 584.        ,    0.0815959...],
                    [ 585.        ,    0.0810298...],
                    [ 586.        ,    0.0804687...],
                    [ 587.        ,    0.0799124...],
                    [ 588.        ,    0.0793609...],
                    [ 589.        ,    0.0788142...],
                    [ 590.        ,    0.0782722...],
                    [ 591.        ,    0.0777349...],
                    [ 592.        ,    0.0772022...],
                    [ 593.        ,    0.0766740...],
                    [ 594.        ,    0.0761504...],
                    [ 595.        ,    0.0756313...],
                    [ 596.        ,    0.0751166...],
                    [ 597.        ,    0.0746063...],
                    [ 598.        ,    0.0741003...],
                    [ 599.        ,    0.0735986...],
                    [ 600.        ,    0.0731012...],
                    [ 601.        ,    0.072608 ...],
                    [ 602.        ,    0.0721189...],
                    [ 603.        ,    0.0716340...],
                    [ 604.        ,    0.0711531...],
                    [ 605.        ,    0.0706763...],
                    [ 606.        ,    0.0702035...],
                    [ 607.        ,    0.0697347...],
                    [ 608.        ,    0.0692697...],
                    [ 609.        ,    0.0688087...],
                    [ 610.        ,    0.0683515...],
                    [ 611.        ,    0.0678981...],
                    [ 612.        ,    0.0674485...],
                    [ 613.        ,    0.0670026...],
                    [ 614.        ,    0.0665603...],
                    [ 615.        ,    0.0661218...],
                    [ 616.        ,    0.0656868...],
                    [ 617.        ,    0.0652555...],
                    [ 618.        ,    0.0648277...],
                    [ 619.        ,    0.0644033...],
                    [ 620.        ,    0.0639825...],
                    [ 621.        ,    0.0635651...],
                    [ 622.        ,    0.0631512...],
                    [ 623.        ,    0.0627406...],
                    [ 624.        ,    0.0623333...],
                    [ 625.        ,    0.0619293...],
```

```
[ 626.        ,    0.0615287...],
[ 627.        ,    0.0611312...],
[ 628.        ,    0.0607370...],
[ 629.        ,    0.0603460...],
[ 630.        ,    0.0599581...],
[ 631.        ,    0.0595733...],
[ 632.        ,    0.0591917...],
[ 633.        ,    0.0588131...],
[ 634.        ,    0.0584375...],
[ 635.        ,    0.0580649...],
[ 636.        ,    0.0576953...],
[ 637.        ,    0.0573286...],
[ 638.        ,    0.0569649...],
[ 639.        ,    0.0566040...],
[ 640.        ,    0.0562460...],
[ 641.        ,    0.0558909...],
[ 642.        ,    0.0555385...],
[ 643.        ,    0.0551890...],
[ 644.        ,    0.0548421...],
[ 645.        ,    0.0544981...],
[ 646.        ,    0.0541567...],
[ 647.        ,    0.053818 ...],
[ 648.        ,    0.0534819...],
[ 649.        ,    0.0531485...],
[ 650.        ,    0.0528176...],
[ 651.        ,    0.0524894...],
[ 652.        ,    0.0521637...],
[ 653.        ,    0.0518405...],
[ 654.        ,    0.0515198...],
[ 655.        ,    0.0512017...],
[ 656.        ,    0.0508859...],
[ 657.        ,    0.0505726...],
[ 658.        ,    0.0502618...],
[ 659.        ,    0.0499533...],
[ 660.        ,    0.0496472...],
[ 661.        ,    0.0493434...],
[ 662.        ,    0.0490420...],
[ 663.        ,    0.0487428...],
[ 664.        ,    0.0484460...],
[ 665.        ,    0.0481514...],
[ 666.        ,    0.0478591...],
[ 667.        ,    0.0475689...],
[ 668.        ,    0.0472810...],
[ 669.        ,    0.0469953...],
[ 670.        ,    0.0467117...],
[ 671.        ,    0.0464302...],
[ 672.        ,    0.0461509...],
[ 673.        ,    0.0458737...],
[ 674.        ,    0.0455986...],
[ 675.        ,    0.0453255...],
[ 676.        ,    0.0450545...],
[ 677.        ,    0.0447855...],
[ 678.        ,    0.0445185...],
[ 679.        ,    0.0442535...],
[ 680.        ,    0.0439905...],
[ 681.        ,    0.0437294...],
[ 682.        ,    0.0434703...],
[ 683.        ,    0.0432131...],
```

```
                    [ 684.        ,    0.0429578...],
                    [ 685.        ,    0.0427044...],
                    [ 686.        ,    0.0424529...],
                    [ 687.        ,    0.0422032...],
                    [ 688.        ,    0.0419553...],
                    [ 689.        ,    0.0417093...],
                    [ 690.        ,    0.0414651...],
                    [ 691.        ,    0.0412226...],
                    [ 692.        ,    0.0409820...],
                    [ 693.        ,    0.0407431...],
                    [ 694.        ,    0.0405059...],
                    [ 695.        ,    0.0402705...],
                    [ 696.        ,    0.0400368...],
                    [ 697.        ,    0.0398047...],
                    [ 698.        ,    0.0395744...],
                    [ 699.        ,    0.0393457...],
                    [ 700.        ,    0.0391187...],
                    [ 701.        ,    0.0388933...],
                    [ 702.        ,    0.0386696...],
                    [ 703.        ,    0.0384474...],
                    [ 704.        ,    0.0382269...],
                    [ 705.        ,    0.0380079...],
                    [ 706.        ,    0.0377905...],
                    [ 707.        ,    0.0375747...],
                    [ 708.        ,    0.0373604...],
                    [ 709.        ,    0.0371476...],
                    [ 710.        ,    0.0369364...],
                    [ 711.        ,    0.0367266...],
                    [ 712.        ,    0.0365184...],
                    [ 713.        ,    0.0363116...],
                    [ 714.        ,    0.0361063...],
                    [ 715.        ,    0.0359024...],
                    [ 716.        ,    0.0357000...],
                    [ 717.        ,    0.0354990...],
                    [ 718.        ,    0.0352994...],
                    [ 719.        ,    0.0351012...],
                    [ 720.        ,    0.0349044...],
                    [ 721.        ,    0.0347090...],
                    [ 722.        ,    0.0345150...],
                    [ 723.        ,    0.0343223...],
                    [ 724.        ,    0.0341310...],
                    [ 725.        ,    0.0339410...],
                    [ 726.        ,    0.0337523...],
                    [ 727.        ,    0.033565 ...],
                    [ 728.        ,    0.0333789...],
                    [ 729.        ,    0.0331941...],
                    [ 730.        ,    0.0330106...],
                    [ 731.        ,    0.0328284...],
                    [ 732.        ,    0.0326474...],
                    [ 733.        ,    0.0324677...],
                    [ 734.        ,    0.0322893...],
                    [ 735.        ,    0.0321120...],
                    [ 736.        ,    0.0319360...],
                    [ 737.        ,    0.0317611...],
                    [ 738.        ,    0.0315875...],
                    [ 739.        ,    0.0314151...],
                    [ 740.        ,    0.0312438...],
                    [ 741.        ,    0.0310737...],
```

```
            [ 742.        ,    0.0309048...],
            [ 743.        ,    0.0307370...],
            [ 744.        ,    0.0305703...],
            [ 745.        ,    0.0304048...],
            [ 746.        ,    0.0302404...],
            [ 747.        ,    0.0300771...],
            [ 748.        ,    0.0299149...],
            [ 749.        ,    0.0297538...],
            [ 750.        ,    0.0295938...],
            [ 751.        ,    0.0294349...],
            [ 752.        ,    0.0292771...],
            [ 753.        ,    0.0291203...],
            [ 754.        ,    0.0289645...],
            [ 755.        ,    0.0288098...],
            [ 756.        ,    0.0286561...],
            [ 757.        ,    0.0285035...],
            [ 758.        ,    0.0283518...],
            [ 759.        ,    0.0282012...],
            [ 760.        ,    0.0280516...],
            [ 761.        ,    0.0279030...],
            [ 762.        ,    0.0277553...],
            [ 763.        ,    0.0276086...],
            [ 764.        ,    0.027463 ...],
            [ 765.        ,    0.0273182...],
            [ 766.        ,    0.0271744...],
            [ 767.        ,    0.0270316...],
            [ 768.        ,    0.0268897...],
            [ 769.        ,    0.0267487...],
            [ 770.        ,    0.0266087...],
            [ 771.        ,    0.0264696...],
            [ 772.        ,    0.0263314...],
            [ 773.        ,    0.0261941...],
            [ 774.        ,    0.0260576...],
            [ 775.        ,    0.0259221...],
            [ 776.        ,    0.0257875...],
            [ 777.        ,    0.0256537...],
            [ 778.        ,    0.0255208...],
            [ 779.        ,    0.0253888...],
            [ 780.        ,    0.0252576...]],
           interpolator=SpragueInterpolator,
           interpolator_args={},
           extrapolator=Extrapolator,
           extrapolator_args={...})
```

colour.scattering_cross_section

colour.**scattering_cross_section**(*wavelength*, *CO2_concentration=300*, *temperature=288.15*, *avogadro_constant=6.02214179e+23*, *n_s=<function air_refraction_index_Bodhaine1999>*, *F_air=<function F_air_Bodhaine1999>*)

Returns the scattering cross section per molecule σ of dry air as function of wavelength λ in centimeters (cm) using given CO_2 concentration in parts per million (ppm) and temperature $T[K]$ in kelvin degrees following *Van de Hulst (1957)* method.

Parameters

- **wavelength** (numeric or array_like) – Wavelength λ in centimeters (cm).

- **CO2_concentration** (numeric or array_like, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric or array_like, optional) – Air temperature $T[K]$ in kelvin degrees.
- **avogadro_constant** (numeric or array_like, optional) – *Avogadro*'s number (molecules mol^{-1}).
- **n_s** (object) – Air refraction index n_s computation method.
- **F_air** (object) – $(6 + 3_p)/(6 - 7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns Scattering cross section per molecule σ of dry air.

Return type numeric or ndarray

Warning: Unlike most objects of `colour.phenomena.rayleigh` module, `colour.scattering_cross_section()` expects wavelength λ to be expressed in centimeters (cm).

References

- *[BWDS99]*
- *[Wiku]*

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

- *[BWDS99]*
- *[Wiku]*

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8)
0.1004070...
```

Plotting

- *Common*
- *Colorimetry*
- *Colour Characterisation*
- *Corresponding Chromaticities*
- *CIE Chromaticity Diagrams*
- *Colour Models*
- *Colour Notation Systems*
- *Optical Phenomena*
- *Colour Quality*
- *Colour Temperature & Correlated Colour Temperature*

- *Colour Models Volume*
- *Geometry Plotting Utilities*

Common

colour.plotting

colour_plotting_defaults([parameters])	Enables *Colour* default plotting parameters.
colour_cycle(**kwargs)	Returns a colour cycle iterator using given colour map.
canvas(**kwargs)	Sets the figure size.
camera(**kwargs)	Sets the camera settings.
decorate(**kwargs)	Sets the figure decorations.
boundaries(**kwargs)	Sets the plot boundaries.
display(**kwargs)	Sets the figure display.
render([with_boundaries, with_decorate])	Convenient wrapper definition combining colour.plotting.decorate(), colour.plotting.boundaries() and colour.plotting.display() definitions.
label_rectangles(rectangles[, rotation, …])	Add labels above given rectangles.
equal_axes3d(axes)	Sets equal aspect ratio to given 3d axes.
single_colour_swatch_plot(colour_swatch, …)	Plots given colour swatch.
multi_colour_swatches_plot(colour_swatches)	Plots given colours swatches.
image_plot(image[, label, label_size, …])	Plots given image.

colour.plotting.colour_plotting_defaults

colour.plotting.**colour_plotting_defaults**(*parameters=None*)

Enables *Colour* default plotting parameters.

Parameters **parameters** (dict, optional) – Parameters to use for plotting.

Returns Definition success.

Return type bool

colour.plotting.colour_cycle

colour.plotting.**colour_cycle**(***kwargs*)

Returns a colour cycle iterator using given colour map.

Other Parameters

- **colour_cycle_map** (*unicode, 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.canvas

`colour.plotting.canvas(**kwargs)`

Sets the figure size.

Other Parameters **figure_size** (*array_like, optional*) – Array defining figure *width* and *height* such as *figure_size = (width, height)*.

Returns Current figure.

Return type Figure

colour.plotting.camera

`colour.plotting.camera(**kwargs)`

Sets the camera settings.

Other Parameters

- **camera_aspect** (*unicode, optional*) – Matplotlib axes aspect. Default is *equal*.
- **elevation** (*numeric, optional*) – Camera elevation.
- **azimuth** (*numeric, optional*) – Camera azimuth.

Returns Current axes.

Return type Axes

colour.plotting.decorate

`colour.plotting.decorate(**kwargs)`

Sets the figure decorations.

Other Parameters

- **title** (*unicode, optional*) – Figure title.
- **x_label** (*unicode, optional*) – *X* axis label.
- **y_label** (*unicode, optional*) – *Y* axis label.
- **legend** (*bool, optional*) – Whether to display the legend. Default is *False*.
- **legend_columns** (*int, optional*) – Number of columns in the legend. Default is *1*.
- **legend_location** (*unicode, optional*) – Matplotlib legend location. Default is *upper right*.
- **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*.
- **x_ticker_locator** (*Locator, optional*) – Locator type for the *X* axis ticker.
- **y_ticker_locator** (*Locator, optional*) – Locator type for the *Y* axis ticker.
- **grid** (*bool, optional*) – Whether to display the grid. Default is *False*.
- **grid_which** (*unicode, optional*) – Controls whether major tick grids, minor tick grids, or both are affected. Default is *both*.

- **grid_axis** (*unicode, optional*) – Controls which set of grid-lines are drawn. Default is *both*.
- **x_axis_line** (*bool, optional*) – Whether to draw the *X* axis line. Default is *False*.
- **y_axis_line** (*bool, optional*) – Whether to draw the *Y* axis line. Default is *False*.
- **aspect** (*unicode, optional*) – Matplotlib axes aspect.
- **no_axes** (*bool, optional*) – Whether to turn off the axes. Default is *False*.

Returns Current axes.

Return type Axes

colour.plotting.boundaries

`colour.plotting.`**`boundaries`**`(`*`**kwargs`*`)`

Sets the plot boundaries.

Other Parameters

- **bounding_box** (*array_like, optional*) – Array defining current axes limits such *bounding_box = (x min, x max, y min, y max)*.
- **x_tighten** (*bool, optional*) – Whether to tighten the *X* axis limit. Default is *False*.
- **y_tighten** (*bool, optional*) – Whether to tighten the *Y* axis limit. Default is *False*.
- **limits** (*array_like, optional*) – Array defining current axes limits such as *limits = (x limit min, x limit max, y limit min, y limit max)*. `limits` argument values are added to the `margins` argument values to define the final bounding box for the current axes.
- **margins** (*array_like, optional*) – Array defining current axes margins such as *margins = (x margin min, x margin max, y margin min, y margin max)*. `margins` argument values are added to the `limits` argument values to define the final bounding box for the current axes.

Returns Current axes.

Return type Axes

colour.plotting.display

`colour.plotting.`**`display`**`(`*`**kwargs`*`)`

Sets the figure display.

Other Parameters

- **standalone** (*bool, optional*) – Whether to show the figure.
- **filename** (*unicode, optional*) – Figure will be saved using given `filename` argument.

Returns Current figure or None.

Return type Figure

colour.plotting.render

`colour.plotting.render(with_boundaries=True, with_decorate=True, **kwargs)`

Convenient wrapper definition combining `colour.plotting.decorate()`, `colour.plotting.boundaries()` and `colour.plotting.display()` definitions.

Parameters

- **with_boundaries** (`bool`, `optional`) – Whether to call `colour.plotting.boundaries()` definition.
- **with_decorate** (`bool`, `optional`) – Whether to call `colour.plotting.decorate()` definition.

Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.render()`}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

colour.plotting.label_rectangles

`colour.plotting.label_rectangles(rectangles, rotation='vertical', text_size=10, offset=None)`

Add labels above given rectangles.

Parameters

- **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.

Returns Definition success.

Return type bool

colour.plotting.equal_axes3d

`colour.plotting.equal_axes3d(axes)`

Sets equal aspect ratio to given 3d axes.

Parameters **axes** (`object`) – Axis to set the equal aspect ratio.

Returns Definition success.

Return type bool

colour.plotting.single_colour_swatch_plot

`colour.plotting.single_colour_swatch_plot(colour_swatch, **kwargs)`

Plots given colour swatch.

Parameters **colour_swatch** (`ColourSwatch`) – ColourSwatch.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **width** (*numeric, optional*) – {colour.plotting.multi_colour_swatches_plot()}, Colour swatch width.
- **height** (*numeric, optional*) – {colour.plotting.multi_colour_swatches_plot()}, Colour swatch height.
- **spacing** (*numeric, optional*) – {colour.plotting.multi_colour_swatches_plot()}, Colour swatches spacing.
- **columns** (*int, optional*) – {colour.plotting.multi_colour_swatches_plot()}, Colour swatches columns count.
- **text_display** (*bool, optional*) – {colour.plotting.multi_colour_swatches_plot()}, Display colour text.
- **text_size** (*numeric, optional*) – {colour.plotting.multi_colour_swatches_plot()}, Colour text size.
- **text_offset** (*numeric, optional*) – {colour.plotting.multi_colour_swatches_plot()}, Colour text offset.

Returns Current figure or None.

Return type Figure

Examples

```
>>> RGB = (0.32315746, 0.32983556, 0.33640183)
>>> single_colour_swatch_plot(ColourSwatch(RGB))
```

colour.plotting.multi_colour_swatches_plot

colour.plotting.**multi_colour_swatches_plot**(*colour_swatches, width=1, height=1, spacing=0, columns=3, text_display=True, text_size='large', text_offset=0.075, background_colour=(1.0, 1.0, 1.0), **kwargs*)

Plots given colours swatches.

Parameters

- **colour_swatches** (list) – ColourSwatch sequence.
- **width** (numeric, optional) – Colour swatch width.
- **height** (numeric, optional) – Colour swatch height.
- **spacing** (numeric, optional) – Colour swatches spacing.
- **columns** (int, optional) – Colour swatches columns count.
- **text_display** (bool, optional) – Display colour text.
- **text_size** (numeric, optional) – Colour text size.
- **text_offset** (numeric, optional) – Colour text offset.
- **background_colour** (array_like or unicode, optional) – Background colour.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> cp1 = ColourSwatch(RGB=(0.45293517, 0.31732158, 0.26414773))
>>> cp2 = ColourSwatch(RGB=(0.77875824, 0.57726450, 0.50453169))
>>> multi_colour_swatches_plot([cp1, cp2])
```

colour.plotting.image_plot

colour.plotting.**image_plot**(*image, label=None, label_size=15, label_colour=None, label_alpha=0.85, interpolation='nearest', colour_map=<matplotlib.colors.LinearSegmentedColormap object>, **kwargs*)

Plots given image.

Parameters

- **image** (array_like) – Image to plot.
- **label** (unicode, optional) – Image label.
- **label_size** (int, optional) – Image label font size.
- **label_colour** (array_like or unicode, optional) – Image label colour.
- **label_alpha** (numeric, optional) – Image label alpha.
- **interpolation** (unicode, optional) – **{'nearest', None, 'none', 'bilinear', 'bicubic', 'spline16', 'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric', 'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos'}** Image display interpolation.
- **colour_map** (unicode, optional) – Colour map used to display single channel images.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> import os
>>> from colour import read_image
>>> path = os.path.join('resources',
...                     ('CIE_1931_Chromaticity_Diagram'
...                      '_CIE_1931_2_Degree_Standard_Observer.png'))
>>> image = read_image(path)
>>> image_plot(image)
```

Colorimetry

colour.plotting

single_spd_plot(spd[, cmfs, ...])	Plots given spectral power distribution.
multi_spd_plot(spds[, cmfs, ...])	Plots given spectral power distributions.
single_cmfs_plot([cmfs])	Plots given colour matching functions.
multi_cmfs_plot([cmfs])	Plots given colour matching functions.
single_illuminant_relative_spd_plot([...])	Plots given single illuminant relative spectral power distribution.
multi_illuminants_relative_spd_plot([...])	Plots given illuminants relative spectral power distributions.
visible_spectrum_plot([cmfs, ...])	Plots the visible colours spectrum using given standard observer *CIE XYZ* colour matching functions.
single_lightness_function_plot([function])	Plots given *Lightness* function.
multi_lightness_function_plot([functions])	Plots given *Lightness* functions.
blackbody_spectral_radiance_plot([...])	Plots given blackbody spectral radiance.
blackbody_colours_plot([shape, cmfs])	Plots blackbody colours.

colour.plotting.single_spd_plot

colour.plotting.**single_spd_plot**(*spd, cmfs='CIE 1931 2 Degree Standard Observer', out_of_gamut_clipping=True, **kwargs*)

Plots given spectral power distribution.

Parameters

- **spd** (SpectralPowerDistribution) – Spectral power distribution to plot.
- **out_of_gamut_clipping** (bool, optional) – Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.
- **cmfs** (unicode) – Standard observer colour matching functions used for spectrum creation.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

References

- *[Spi15]*

Examples

```
>>> from colour import SpectralPowerDistribution
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
```

```
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> spd = SpectralPowerDistribution(data, name='Custom')
>>> single_spd_plot(spd)
```

colour.plotting.multi_spd_plot

colour.plotting.**multi_spd_plot**(*spds, cmfs='CIE 1931 2 Degree Standard Observer', use_spds_colours=False, normalise_spds_colours=False, **kwargs*)

Plots given spectral power distributions.

Parameters

- **spds** (list) – Spectral power distributions to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectrum creation.
- **use_spds_colours** (bool, optional) – Whether to use spectral power distributions colours.
- **normalise_spds_colours** (bool) – Whether to normalise spectral power distributions colours.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> from colour import SpectralPowerDistribution
>>> data_1 = {
...     500: 0.004900,
...     510: 0.009300,
...     520: 0.063270,
...     530: 0.165500,
...     540: 0.290400,
...     550: 0.433450,
...     560: 0.594500
... }
>>> data_2 = {
...     500: 0.323000,
...     510: 0.503000,
...     520: 0.710000,
...     530: 0.862000,
...     540: 0.954000,
...     550: 0.994950,
...     560: 0.995000
... }
>>> spd1 = SpectralPowerDistribution(data_1, name='Custom 1')
```

```
>>> spd2 = SpectralPowerDistribution(data_2, name='Custom 2')
>>> multi_spd_plot([spd1, spd2])
```

colour.plotting.single_cmfs_plot

colour.plotting.**single_cmfs_plot**(*cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)
Plots given colour matching functions.

Parameters **cmfs** (unicode, optional) – Colour matching functions to plot.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> single_cmfs_plot()
```

colour.plotting.multi_cmfs_plot

colour.plotting.**multi_cmfs_plot**(*cmfs=None*, ***kwargs*)
Plots given colour matching functions.

Parameters **cmfs** (array_like, optional) – Colour matching functions to plot.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> cmfs = [
... 'CIE 1931 2 Degree Standard Observer',
... 'CIE 1964 10 Degree Standard Observer']
>>> multi_cmfs_plot(cmfs)
```

colour.plotting.single_illuminant_relative_spd_plot

colour.plotting.**single_illuminant_relative_spd_plot**(*illuminant='A'*, *cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)
Plots given single illuminant relative spectral power distribution.

Parameters

- **illuminant** (unicode, optional) – Factory illuminant to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions to plot.

Other Parameters

- ****kwargs** (*dict, optional*) – {`colour.plotting.render()`}, Please refer to the documentation of the previously listed definition.
- **out_of_gamut_clipping** (*bool, optional*) – {`colour.plotting.single_spd_plot()`}, 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 or None.

Return type Figure

References

- *[Spi15]*

Examples

```
>>> single_illuminant_relative_spd_plot()
```

colour.plotting.multi_illuminants_relative_spd_plot

`colour.plotting.`**`multi_illuminants_relative_spd_plot`**(*illuminants=None*, ***kwargs*)
Plots given illuminants relative spectral power distributions.

Parameters **`illuminants`** (`array_like, optional`) – Factory illuminants to plot.

Other Parameters

- ****kwargs** (*dict, optional*) – {`colour.plotting.render()`}, Please refer to the documentation of the previously listed definition.
- **use_spds_colours** (*bool, optional*) – {`colour.plotting.multi_spd_plot()`} Whether to use spectral power distributions colours.
- **normalise_spds_colours** (*bool*) – {`colour.plotting.multi_spd_plot()`} Whether to normalise spectral power distributions colours.

Returns Current figure or None.

Return type Figure

Examples

```
>>> multi_illuminants_relative_spd_plot(['A', 'B', 'C'])
```

colour.plotting.visible_spectrum_plot

`colour.plotting.`**`visible_spectrum_plot`**(*cmfs='CIE 1931 2 Degree Standard Observer', out_of_gamut_clipping=True*, ***kwargs*)
Plots the visible colours spectrum using given standard observer *CIE XYZ* colour matching functions.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectrum creation.
- **out_of_gamut_clipping** (bool, optional) – Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

References

- *[Spi15]*

Examples

```
>>> visible_spectrum_plot()
```

colour.plotting.single_lightness_function_plot

colour.plotting.**single_lightness_function_plot**(*function='CIE 1976', **kwargs*)

Plots given *Lightness* function.

Parameters **function** (unicode, optional) – *Lightness* function to plot.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> single_lightness_function_plot()
```

colour.plotting.multi_lightness_function_plot

colour.plotting.**multi_lightness_function_plot**(*functions=None, **kwargs*)

Plots given *Lightness* functions.

Parameters **functions** (array_like, optional) – *Lightness* functions to plot.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Raises KeyError – If one of the given *Lightness* function is not found in the factory *Lightness* functions.

Examples

```
>>> fs = ('CIE 1976', 'Wyszecki 1963')
>>> multi_lightness_function_plot(fs)
```

colour.plotting.blackbody_spectral_radiance_plot

colour.plotting.**blackbody_spectral_radiance_plot**(*temperature=3500, cmfs='CIE 1931 2 Degree Standard Observer', blackbody='VY Canis Major', **kwargs*)

Plots given blackbody spectral radiance.

Parameters

- **temperature** (numeric, optional) – Blackbody temperature.
- **cmfs** (unicode, optional) – Standard observer colour matching functions.
- **blackbody** (unicode, optional) – Blackbody name.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> blackbody_spectral_radiance_plot()
```

colour.plotting.blackbody_colours_plot

colour.plotting.**blackbody_colours_plot**(*shape=SpectralShape(150, 12500, 50), cmfs='CIE 1931 2 Degree Standard Observer', **kwargs*)

Plots blackbody colours.

Parameters

- **shape** (SpectralShape, optional) – Spectral shape to use as plot boundaries.
- **cmfs** (unicode, optional) – Standard observer colour matching functions.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> blackbody_colours_plot()
```

Colour Characterisation

`colour.plotting`

`colour_checker_plot([colour_checker])`	Plots given colour checker.

colour.plotting.colour_checker_plot

`colour.plotting.colour_checker_plot(colour_checker='ColorChecker 2005', **kwargs)`

Plots given colour checker.

Parameters **colour_checker** (`unicode, optional`) – Color checker name.

Other Parameters

- ****kwargs** (*dict, optional*) – {`colour.plotting.render()`}, Please refer to the documentation of the previously listed definition.
- **width** (*numeric, optional*) – {`colour.plotting.multi_colour_swatches_plot()`}, Colour swatch width.
- **height** (*numeric, optional*) – {`colour.plotting.multi_colour_swatches_plot()`}, Colour swatch height.
- **spacing** (*numeric, optional*) – {`colour.plotting.multi_colour_swatches_plot()`}, Colour swatches spacing.
- **columns** (*int, optional*) – {`colour.plotting.multi_colour_swatches_plot()`}, Colour swatches columns count.
- **text_display** (*bool, optional*) – {`colour.plotting.multi_colour_swatches_plot()`}, Display colour text.
- **text_size** (*numeric, optional*) – {`colour.plotting.multi_colour_swatches_plot()`}, Colour text size.
- **text_offset** (*numeric, optional*) – {`colour.plotting.multi_colour_swatches_plot()`}, Colour text offset.

Returns Current figure or None.

Return type Figure

Raises `KeyError` – If the given colour rendition chart is not found in the factory colour rendition charts.

Examples

```
>>> colour_checker_plot()
```

Corresponding Chromaticities

colour.plotting

corresponding_chromaticities_prediction_plot([...])	Plots given chromatic adaptation model corresponding chromaticities prediction.

colour.plotting.corresponding_chromaticities_prediction_plot

colour.plotting.**corresponding_chromaticities_prediction_plot**(*experiment=1*, *model='Von Kries'*, *transform='CAT02'*, ***kwargs*)

Plots given chromatic adaptation model corresponding chromaticities prediction.

Parameters

- **experiment** (int, optional) – Corresponding chromaticities prediction experiment number.
- **model** (unicode, optional) – Corresponding chromaticities prediction model name.
- **transform** (unicode, optional) – Transformation to use with *Von Kries* chromatic adaptation model.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1976UCS()} Whether to display the chromaticity diagram background colours.

Returns Current figure or None.

Return type Figure

Examples

```
>>> corresponding_chromaticities_prediction_plot()
```

CIE Chromaticity Diagrams

colour.plotting

chromaticity_diagram_plot_CIE1931([cmfs, ...])	Plots the *CIE 1931 Chromaticity Diagram*.
chromaticity_diagram_plot_CIE1960UCS([cmfs, ...])	Plots the *CIE 1960 UCS Chromaticity Diagram*.
chromaticity_diagram_plot_CIE1976UCS([cmfs, ...])	Plots the *CIE 1976 UCS Chromaticity Diagram*.
spds_chromaticity_diagram_plot_CIE1931(spds)	Plots given spectral power distribution chromaticity coordinates into the *CIE 1931 Chromaticity Diagram*.

Continued on next page

Table 3.177 – continued from previous page

spds_chromaticity_diagram_plot_CIE1960UCS(spds)	Plots given spectral power distribution chromaticity coordinates into the *CIE 1960 UCS Chromaticity Diagram*.
spds_chromaticity_diagram_plot_CIE1976UCS(spds)	Plots given spectral power distribution chromaticity coordinates into the *CIE 1976 UCS Chromaticity Diagram*.

colour.plotting.chromaticity_diagram_plot_CIE1931

colour.plotting.**chromaticity_diagram_plot_CIE1931**(*cmfs='CIE 1931 2 Degree Standard Observer', show_diagram_colours=True, use_cached_diagram_colours=True, **kwargs*)

Plots the *CIE 1931 Chromaticity Diagram*.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for diagram bounds.
- **show_diagram_colours** (bool, optional) – Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (bool, optional) – Whether to used the cached chromaticity diagram background colours image.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.diagrams.chromaticity_diagram_colours_CIE1931(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure or None.

Return type Figure

Examples

```
>>> chromaticity_diagram_plot_CIE1931()
```

colour.plotting.chromaticity_diagram_plot_CIE1960UCS

colour.plotting.**chromaticity_diagram_plot_CIE1960UCS**(*cmfs='CIE 1931 2 Degree Standard Observer', show_diagram_colours=True, use_cached_diagram_colours=True, **kwargs*)

Plots the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for diagram bounds.
- **show_diagram_colours** (bool, optional) – Whether to display the chromaticity diagram background colours.

- **use_cached_diagram_colours** (`bool`, `optional`) – Whether to used the cached chromaticity diagram background colours image.

Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.diagrams.chromaticity_diagram_colours_CIE1960UCS()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure or None.

Return type Figure

Examples

```
>>> chromaticity_diagram_plot_CIE1960UCS()
```

colour.plotting.chromaticity_diagram_plot_CIE1976UCS

`colour.plotting.`**`chromaticity_diagram_plot_CIE1976UCS`**(*cmfs='CIE 1931 2 Degree Standard Observer', show_diagram_colours=True, use_cached_diagram_colours=True, **kwargs*)

Plots the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **cmfs** (`unicode`, `optional`) – Standard observer colour matching functions used for diagram bounds.
- **show_diagram_colours** (`bool`, `optional`) – Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (`bool`, `optional`) – Whether to used the cached chromaticity diagram background colours image.

Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.diagrams.chromaticity_diagram_colours_CIE1976UCS()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure or None.

Return type Figure

Examples

```
>>> chromaticity_diagram_plot_CIE1976UCS()
```

colour.plotting.spds_chromaticity_diagram_plot_CIE1931

colour.plotting.**spds_chromaticity_diagram_plot_CIE1931**(*spds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *annotate=True*, *chromaticity_diagram_callable_CIE1931=<function chromaticity_diagram_plot_CIE1931>*, ***kwargs*)

Plots given spectral power distribution chromaticity coordinates into the *CIE 1931 Chromaticity Diagram*.

Parameters

- **spds** (array_like, optional) – Spectral power distributions to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for diagram bounds.
- **annotate** (bool) – Should resulting chromaticity coordinates annotated with their respective spectral power distribution names.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1931()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1931()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Examples

```
>>> from colour import ILLUMINANTS_RELATIVE_SPDS
>>> A = ILLUMINANTS_RELATIVE_SPDS['A']
>>> D65 = ILLUMINANTS_RELATIVE_SPDS['D65']
>>> spds_chromaticity_diagram_plot_CIE1931([A, D65])
```

colour.plotting.spds_chromaticity_diagram_plot_CIE1960UCS

colour.plotting.**spds_chromaticity_diagram_plot_CIE1960UCS**(*spds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *annotate=True*, *chromaticity_diagram_callable_CIE1960UCS=<function chromaticity_diagram_plot_CIE1960UCS>*, ***kwargs*)

Plots given spectral power distribution chromaticity coordinates into the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **spds** (array_like, optional) – Spectral power distributions to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for diagram bounds.
- **annotate** (bool) – Should resulting chromaticity coordinates annotated with their respective spectral power distribution names.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1960UCS()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1960UCS()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Examples

```
>>> from colour import ILLUMINANTS_RELATIVE_SPDS
>>> A = ILLUMINANTS_RELATIVE_SPDS['A']
>>> D65 = ILLUMINANTS_RELATIVE_SPDS['D65']
>>> spds_chromaticity_diagram_plot_CIE1960UCS([A, D65])
```

colour.plotting.spds_chromaticity_diagram_plot_CIE1976UCS

colour.plotting.**spds_chromaticity_diagram_plot_CIE1976UCS**(*spds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *annotate=True*, *chromaticity_diagram_callable_CIE1976UCS=<function chromaticity_diagram_plot_CIE1976UCS>*, ***kwargs*)

Plots given spectral power distribution chromaticity coordinates into the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **spds** (array_like, optional) – Spectral power distributions to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for diagram bounds.
- **annotate** (bool) – Should resulting chromaticity coordinates annotated with their respective spectral power distribution names.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1976UCS()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1976UCS()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Examples

```
>>> from colour import ILLUMINANTS_RELATIVE_SPDS
>>> A = ILLUMINANTS_RELATIVE_SPDS['A']
>>> D65 = ILLUMINANTS_RELATIVE_SPDS['D65']
>>> spds_chromaticity_diagram_plot_CIE1976UCS([A, D65])
```

Colour Models

colour.plotting

RGB_colourspaces_chromaticity_diagram_plot_CIE1931([...])	Plots given *RGB* colourspaces in *CIE 1931 Chromaticity Diagram*.

Continued on next page

Table 3.178 – continued from previous page

RGB_colourspaces_chromaticity_diagram_plot_CIE1960UCS([...])	Plots given *RGB* colourspaces in *CIE 1960 UCS Chromaticity Diagram*.
RGB_colourspaces_chromaticity_diagram_plot_CIE1976UCS([...])	Plots given *RGB* colourspaces in *CIE 1976 UCS Chromaticity Diagram*.
RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1931(RGB)	Plots given *RGB* colourspace array in *CIE 1931 Chromaticity Diagram*.
RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1960UCS(RGB)	Plots given *RGB* colourspace array in *CIE 1960 UCS Chromaticity Diagram*.
RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1976UCS(RGB)	Plots given *RGB* colourspace array in *CIE 1976 UCS Chromaticity Diagram*.
single_cctf_plot([colourspace, decoding_cctf])	Plots given colourspace colour component transfer function.
multi_cctf_plot([colourspaces, decoding_cctf])	Plots given colourspaces colour component transfer functions.

colour.plotting.RGB_colourspaces_chromaticity_diagram_plot_CIE1931

colour.plotting.**RGB_colourspaces_chromaticity_diagram_plot_CIE1931**(*colourspaces=None, cmfs='CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable_CIE1931=<function chromaticity_diagram_plot_CIE1931>, **kwargs*)

Plots given *RGB* colourspaces in *CIE 1931 Chromaticity Diagram*.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for diagram bounds.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1931()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1931()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Examples

```
>>> c = ['ITU-R Rec. 709', 'ACEScg', 'S-Gamut']
>>> RGB_colourspaces_chromaticity_diagram_plot_CIE1931(c)
...
```

colour.plotting.RGB_colourspaces_chromaticity_diagram_plot_CIE1960UCS

colour.plotting.**RGB_colourspaces_chromaticity_diagram_plot_CIE1960UCS**(*colourspaces=None, cmfs='CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable_CIE1960UCS=<function chromaticity_diagram_plot_CIE1960UCS>, **kwargs*)

Plots given *RGB* colourspaces in *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for diagram bounds.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1960UCS()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1960UCS()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Examples

```
>>> c = ['ITU-R Rec. 709', 'ACEScg', 'S-Gamut']
>>> RGB_colourspaces_chromaticity_diagram_plot_CIE1960UCS(c)
...
```

colour.plotting.RGB_colourspaces_chromaticity_diagram_plot_CIE1976UCS

colour.plotting.**RGB_colourspaces_chromaticity_diagram_plot_CIE1976UCS**(*colourspaces=None, cmfs='CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable_CIE1976UCS=<function chromaticity_diagram_plot_CIE1976UCS>, **kwargs*)

Plots given *RGB* colourspaces in *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for diagram bounds.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1976UCS()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1976UCS()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Examples

```
>>> c = ['ITU-R Rec. 709', 'ACEScg', 'S-Gamut']
>>> RGB_colourspaces_chromaticity_diagram_plot_CIE1976UCS(c)
...
```

colour.plotting.RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1931

colour.plotting.**RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1931**(*RGB, colourspace='sRGB', chromaticity_diagram_callable_CIE1931=<function RGB_colourspaces_chromaticity_di **kwargs*)

Plots given *RGB* colourspace array in *CIE 1931 Chromaticity Diagram*.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1931()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1931()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Examples

```
>>> RGB = np.random.random((10, 10, 3))
>>> c = 'ITU-R Rec. 709'
>>> RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1931(RGB, c)
...
```

colour.plotting.RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1960UCS

colour.plotting.**RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1960UCS**(*RGB, colourspace='sRGB', chromaticity_diagram_callable_CIE1960U RGB_colourspaces_chromaticity **kwargs*)

Plots given *RGB* colourspace array in *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1960UCS()}, Whether to display the chromaticity diagram background colours.

- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1960UCS()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Examples

```
>>> RGB = np.random.random((10, 10, 3))
>>> c = 'ITU-R BT.709'
>>> RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1960UCS(
...     RGB, c)
```

colour.plotting.RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1976UCS

colour.plotting.**RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1976UCS**(*RGB, colourspace='sRGB', chromaticity_diagram_callable_CIE1976U RGB_colourspaces_chromaticity **kwargs*)

Plots given *RGB* colourspace array in *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1976UCS()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1976UCS()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Examples

```
>>> RGB = np.random.random((10, 10, 3))
>>> c = 'ITU-R BT.709'
>>> RGB_chromaticity_coordinates_chromaticity_diagram_plot_CIE1976UCS(
...     RGB, c)
```

colour.plotting.single_cctf_plot

colour.plotting.**single_cctf_plot**(*colourspace='ITU-R BT.709', decoding_cctf=False, **kwargs*)
Plots given colourspace colour component transfer function.

Parameters

- **colourspace** (unicode, optional) – *RGB* Colourspace colour component transfer function to plot.
- **decoding_cctf** (bool) – Plot decoding colour component transfer function instead.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> single_cctf_plot()
```

colour.plotting.multi_cctf_plot

colour.plotting.**multi_cctf_plot**(*colourspaces=None, decoding_cctf=False, **kwargs*)
Plots given colourspaces colour component transfer functions.

Parameters

- **colourspaces** (array_like, optional) – Colourspaces colour component transfer function to plot.
- **decoding_cctf** (bool) – Plot decoding colour component transfer function instead.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> multi_cctf_plot(['ITU-R BT.709', 'sRGB'])
```

Colour Notation Systems

`colour.plotting`

`single_munsell_value_function_plot`([function])	Plots given *Lightness* function.
`multi_munsell_value_function_plot`([functions])	Plots given *Munsell* value functions.

colour.plotting.single_munsell_value_function_plot

colour.plotting.**single_munsell_value_function_plot**(*function='ASTM D1535-08'*, ***kwargs*)
Plots given *Lightness* function.

Parameters **function** (unicode, optional) – *Munsell* value function to plot.

Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.render()`}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> f = 'ASTM D1535-08'
>>> single_munsell_value_function_plot(f)
```

colour.plotting.multi_munsell_value_function_plot

colour.plotting.**multi_munsell_value_function_plot**(*functions=None*, ***kwargs*)
Plots given *Munsell* value functions.

Parameters **functions** (array_like, optional) – *Munsell* value functions to plot.

Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.render()`}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Raises `KeyError` – If one of the given *Munsell* value function is not found in the factory *Munsell* value functions.

Examples

```
>>> fs = ('ASTM D1535-08', 'McCamy 1987')
>>> multi_munsell_value_function_plot(fs)
```

Optical Phenomena

colour.plotting

single_rayleigh_scattering_spd_plot([...])	Plots a single *Rayleigh* scattering spectral power distribution.
the_blue_sky_plot([cmfs])	Plots the blue sky.

colour.plotting.single_rayleigh_scattering_spd_plot

colour.plotting.**single_rayleigh_scattering_spd_plot**(*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 power distribution.

Parameters

- **CO2_concentration** (numeric, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric, optional) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (numeric) – Surface pressure P of the measurement site.
- **latitude** (numeric, optional) – Latitude of the site in degrees.
- **altitude** (numeric, optional) – Altitude of the site in meters.
- **cmfs** (unicode, optional) – Standard observer colour matching functions.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **out_of_gamut_clipping** (*bool, optional*) – {colour.plotting.single_spd_plot()}, 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 or None.

Return type Figure

Examples

```
>>> single_rayleigh_scattering_spd_plot()
```

colour.plotting.the_blue_sky_plot

colour.plotting.**the_blue_sky_plot**(*cmfs='CIE 1931 2 Degree Standard Observer', **kwargs*)

Plots the blue sky.

Parameters **cmfs** (unicode, optional) – Standard observer colour matching functions.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> the_blue_sky_plot()
```

Colour Quality

colour.plotting

single_spd_colour_rendering_index_bars_plot(...)	Plots the *Colour Rendering Index* (CRI) of given illuminant or light source spectral power distribution.
multi_spd_colour_rendering_index_bars_plot(...)	Plots the *Colour Rendering Index* (CRI) of given illuminants or light sources spectral power distributions.
single_spd_colour_quality_scale_bars_plot(...)	Plots the *Colour Quality Scale* (CQS) of given illuminant or light source spectral power distribution.
multi_spd_colour_quality_scale_bars_plot(...)	Plots the *Colour Quality Scale* (CQS) of given illuminants or light sources spectral power distributions.

colour.plotting.single_spd_colour_rendering_index_bars_plot

colour.plotting.**single_spd_colour_rendering_index_bars_plot**(*spd*, ***kwargs*)

Plots the *Colour Rendering Index* (CRI) of given illuminant or light source spectral power distribution.

Parameters **spd** (SpectralPowerDistribution) – Illuminant or light source spectral power distribution to plot the *Colour Rendering Index* (CRI).

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **labels** (*bool, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Hatching pattern repeat.

Returns Current figure or None.

Return type Figure

Examples

```
>>> from colour import ILLUMINANTS_RELATIVE_SPDS
>>> illuminant = ILLUMINANTS_RELATIVE_SPDS['F2']
>>> single_spd_colour_rendering_index_bars_plot(illuminant)
...
```

colour.plotting.multi_spd_colour_rendering_index_bars_plot

colour.plotting.**multi_spd_colour_rendering_index_bars_plot**(*spds*, ***kwargs*)

Plots the *Colour Rendering Index* (CRI) of given illuminants or light sources spectral power distributions.

Parameters **spds** (array_like) – Array of illuminants or light sources spectral power distributions to plot the *Colour Rendering Index* (CRI).

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **labels** (*bool, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Hatching pattern repeat.

Returns Current figure or None.

Return type Figure

Examples

```
>>> from colour import (ILLUMINANTS_RELATIVE_SPDS,
...                     LIGHT_SOURCES_RELATIVE_SPDS)
>>> illuminant = ILLUMINANTS_RELATIVE_SPDS['F2']
>>> light_source = LIGHT_SOURCES_RELATIVE_SPDS['Kinoton 75P']
>>> multi_spd_colour_rendering_index_bars_plot([illuminant, light_source])
...
```

colour.plotting.single_spd_colour_quality_scale_bars_plot

colour.plotting.**single_spd_colour_quality_scale_bars_plot**(*spd*, ***kwargs*)

Plots the *Colour Quality Scale* (CQS) of given illuminant or light source spectral power distribution.

Parameters **spd** (SpectralPowerDistribution) – Illuminant or light source spectral power distribution to plot the *Colour Quality Scale* (CQS).

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

- **labels** (*bool, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Hatching pattern repeat.

Returns Current figure or None.

Return type Figure

Examples

```
>>> from colour import ILLUMINANTS_RELATIVE_SPDS
>>> illuminant = ILLUMINANTS_RELATIVE_SPDS['F2']
>>> single_spd_colour_quality_scale_bars_plot(illuminant)
...
```

colour.plotting.multi_spd_colour_quality_scale_bars_plot

colour.plotting.**multi_spd_colour_quality_scale_bars_plot**(*spds*, ***kwargs*)

Plots the *Colour Quality Scale* (CQS) of given illuminants or light sources spectral power distributions.

Parameters **spds** (array_like) – Array of illuminants or light sources spectral power distributions to plot the *Colour Quality Scale* (CQS).

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **labels** (*bool, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {colour.plotting.quality.colour_quality_bars_plot()}, Hatching pattern repeat.

Returns Current figure or None.

Return type Figure

Examples

```
>>> from colour import (ILLUMINANTS_RELATIVE_SPDS,
...                     LIGHT_SOURCES_RELATIVE_SPDS)
>>> illuminant = ILLUMINANTS_RELATIVE_SPDS['F2']
>>> light_source = LIGHT_SOURCES_RELATIVE_SPDS['Kinoton 75P']
>>> multi_spd_colour_quality_scale_bars_plot([illuminant, light_source])
...
```

Colour Temperature & Correlated Colour Temperature

colour.plotting

planckian_locus_chromaticity_diagram_plot_CIE1931([…])	Plots the planckian locus and given illuminants in *CIE 1931 Chromaticity Diagram*.
planckian_locus_chromaticity_diagram_plot_CIE1960UCS([…])	Plots the planckian locus and given illuminants in *CIE 1960 UCS Chromaticity Diagram*.

colour.plotting.planckian_locus_chromaticity_diagram_plot_CIE1931

colour.plotting.**planckian_locus_chromaticity_diagram_plot_CIE1931**(*illuminants=None, chromaticity_diagram_callable_CIE1931=<function chromaticity_diagram_plot_CIE1931>, **kwargs*)

Plots the planckian locus and given illuminants in *CIE 1931 Chromaticity Diagram*.

Parameters

- **illuminants** (array_like, optional) – Factory illuminants to plot.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1931()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1931()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Raises KeyError – If one of the given illuminant is not found in the factory illuminants.

Examples

```
>>> planckian_locus_chromaticity_diagram_plot_CIE1931(['A', 'B', 'C'])
...
```

colour.plotting.planckian_locus_chromaticity_diagram_plot_CIE1960UCS

colour.plotting.**planckian_locus_chromaticity_diagram_plot_CIE1960UCS**(*illuminants=None, chromaticity_diagram_callable_CIE1960UCS=<function chromaticity_diagram_plot_CIE1960UCS>, **kwargs*)

Plots the planckian locus and given illuminants in *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **illuminants** (array_like, optional) – Factory illuminants to plot.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.
- **show_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1960UCS()}, Whether to display the chromaticity diagram background colours.
- **use_cached_diagram_colours** (*bool, optional*) – {colour.plotting.chromaticity_diagram_plot_CIE1960UCS()}, Whether to used the cached chromaticity diagram background colours image.

Returns Current figure or None.

Return type Figure

Raises KeyError – If one of the given illuminant is not found in the factory illuminants.

Examples

```
>>> planckian_locus_chromaticity_diagram_plot_CIE1960UCS(['A', 'C', 'E'])
...
```

Colour Models Volume

colour.plotting

RGB_colourspaces_gamuts_plot([colourspaces, …])	Plots given *RGB* colourspaces gamuts in given reference colourspace.
RGB_scatter_plot(RGB, colourspace[, …])	Plots given *RGB* colourspace array in a scatter plot.

colour.plotting.RGB_colourspaces_gamuts_plot

colour.plotting.**RGB_colourspaces_gamuts_plot**(*colourspaces=None, reference_colourspace='CIE xyY', segments=8, display_grid=True, grid_segments=10, spectral_locus=False, spectral_locus_colour=None, cmfs='CIE 1931 2 Degree Standard Observer', **kwargs*)

Plots given *RGB* colourspaces gamuts in given reference colourspace.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot the gamuts.
- **reference_colourspace** (unicode, optional) – **{'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'IPT', 'Hunter Lab', 'Hunter Rdab'}**, Reference colourspace to plot the gamuts into.
- **segments** (int, optional) – Edge segments count for each *RGB* colourspace cubes.
- **display_grid** (bool, optional) – Display a grid at the bottom of the *RGB* colourspace cubes.
- **grid_segments** (bool, optional) – Edge segments count for the grid.
- **spectral_locus** (bool, optional) – Is spectral locus line plotted.
- **spectral_locus_colour** (array_like, optional) – Spectral locus line colour.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectral locus.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.volume.nadir_grid()}, Please refer to the documentation of the previously listed definition.
- **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 or None.

Return type Figure

Examples

```
>>> c = ['ITU-R BT.709', 'ACEScg', 'S-Gamut']
>>> RGB_colourspaces_gamuts_plot(c)
```

colour.plotting.RGB_scatter_plot

colour.plotting.**RGB_scatter_plot**(*RGB, colourspace, reference_colourspace='CIE xyY', colourspaces=None, segments=8, display_grid=True, grid_segments=10, spectral_locus=False, spectral_locus_colour=None, points_size=12, cmfs='CIE 1931 2 Degree Standard Observer', **kwargs*)

Plots given *RGB* colourspace array in a scatter plot.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (RGB_Colourspace) – *RGB* colourspace of the *RGB* array.
- **reference_colourspace** (unicode, optional) – **{'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'IPT', 'Hunter Lab', 'Hunter Rdab'}**, Reference colourspace for colour conversion.
- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot the gamuts.
- **segments** (int, optional) – Edge segments count for each *RGB* colourspace cubes.
- **display_grid** (bool, optional) – Display a grid at the bottom of the *RGB* colourspace cubes.
- **grid_segments** (bool, optional) – Edge segments count for the grid.
- **spectral_locus** (bool, optional) – Is spectral locus line plotted.
- **spectral_locus_colour** (array_like, optional) – Spectral locus line colour.
- **points_size** (numeric, optional) – Scatter points size.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectral locus.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.RGB_colourspaces_gamuts_plot()}, Please refer to the documentation of the previously listed definition.

Returns Current figure or None.

Return type Figure

Examples

```
>>> c = 'ITU-R BT.709'
>>> RGB_scatter_plot(c)
```

Geometry Plotting Utilities

colour.plotting

quad([plane, origin, width, height, depth])	Returns the vertices of a quad geometric element in counter-clockwise order.
grid([plane, origin, width, height, depth, . . .])	Returns the vertices of a grid made of quads.

Continued on next page

Table 3.184 – continued from previous page

cube([plane, origin, width, height, depth, . . .])	Returns the vertices of a cube made of grids.

colour.plotting.quad

colour.plotting.**quad**(*plane='xy', origin=None, width=1, height=1, depth=0*)

Returns the vertices of a quad geometric element in counter-clockwise order.

Parameters

- **plane** (array_like, optional) – **{'xy', 'xz', 'yz'}**, Construction plane of the quad.
- **origin** (array_like, optional) – Quad origin on the construction plane.
- **width** (numeric, optional) – Quad width.
- **height** (numeric, optional) – Quad height.
- **depth** (numeric, optional) – Quad depth.

Returns Quad vertices.

Return type ndarray

Examples

```
>>> quad()
array([[0, 0, 0],
       [1, 0, 0],
       [1, 1, 0],
       [0, 1, 0]])
```

colour.plotting.grid

colour.plotting.**grid**(*plane='xy', origin=None, width=1, height=1, depth=0, width_segments=1, height_segments=1*)

Returns the vertices of a grid made of quads.

Parameters

- **plane** (array_like, optional) – **{'xy', 'xz', 'yz'}**, Construction plane of the grid.
- **origin** (array_like, optional) – Grid origin on the construction plane.
- **width** (numeric, optional) – Grid width.
- **height** (numeric, optional) – Grid height.
- **depth** (numeric, optional) – Grid depth.
- **width_segments** (int, optional) – Grid segments, quad counts along the width.
- **height_segments** (int, optional) – Grid segments, quad counts along the height.

Returns Grid vertices.

Return type ndarray

Examples

```
>>> grid(width_segments=2, height_segments=2)
array([[[ 0. ,  0. ,  0. ],
        [ 0.5,  0. ,  0. ],
        [ 0.5,  0.5,  0. ],
        [ 0. ,  0.5,  0. ]],

       [[ 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.plotting.cube

colour.plotting.**cube**(*plane=None*, *origin=None*, *width=1*, *height=1*, *depth=1*, *width_segments=1*, *height_segments=1*, *depth_segments=1*)

Returns the vertices of a cube made of grids.

Parameters

- **plane** (array_like, optional) – Any combination of **{'+x', '-x', '+y', '-y', '+z', '-z'}**, Included grids in the cube construction.
- **origin** (array_like, optional) – Cube origin.
- **width** (numeric, optional) – Cube width.
- **height** (numeric, optional) – Cube height.
- **depth** (numeric, optional) – Cube depth.
- **width_segments** (int, optional) – Cube segments, quad counts along the width.
- **height_segments** (int, optional) – Cube segments, quad counts along the height.
- **depth_segments** (int, optional) – Cube segments, quad counts along the depth.

Returns Cube vertices.

Return type ndarray

Examples

```
>>> cube()
array([[[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
```

```
        [ 0.,  1.,  0.]],

       [[ 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.]]])
```

Colour Quality

Colour Rendering Index

colour

colour_rendering_index(spd_test[, ...])	Returns the *Colour Rendering Index* (CRI) Q_a of given spectral power distribution.

colour.colour_rendering_index

colour.**colour_rendering_index**(*spd_test*, *additional_data=False*)

Returns the *Colour Rendering Index* (CRI) Q_a of given spectral power distribution.

Parameters

- **spd_test** (SpectralPowerDistribution) – Test spectral power distribution.
- **additional_data** (bool, optional) – Output additional data.

Returns *Colour Rendering Index* (CRI).

Return type numeric or *CRI_Specification*

References

- *[OD08]*

Examples

```
>>> from colour import ILLUMINANTS_RELATIVE_SPDS
>>> spd = ILLUMINANTS_RELATIVE_SPDS['F2']
>>> colour_rendering_index(spd)
64.1515202...
```

colour.quality

CRI_Specification	Defines the *Colour Rendering Index* (CRI) colour quality specification.

colour.quality.CRI_Specification

class colour.quality.**CRI_Specification**

Defines the *Colour Rendering Index* (CRI) colour quality specification.

Parameters

- **name** (unicode) – Name of the test spectral power distribution.
- **Q_a** (numeric) – *Colour Rendering Index* (CRI) Q_a.
- **Q_as** (dict) – Individual *colour rendering indexes* data for each sample.
- **colorimetry_data** (tuple) – Colorimetry data for the test and reference computations.

References

- *[OD08]*

Create new instance of CRI_Specification(name, Q_a, Q_as, colorimetry_data)

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

Colour Quality Scale

colour

colour_quality_scale(spd_test[, additional_data])	Returns the *Colour Quality Scale* (CQS) of given spectral power distribution.

colour.colour_quality_scale

colour.**colour_quality_scale**(*spd_test*, *additional_data=False*)

Returns the *Colour Quality Scale* (CQS) of given spectral power distribution.

Parameters

- **spd_test** (SpectralPowerDistribution) – Test spectral power distribution.
- **additional_data** (bool, optional) – Output additional data.

Returns Color quality scale.

Return type numeric or *CQS_Specification*

References

- *[DO10]*
- *[OD08]*

Examples

```
>>> from colour import ILLUMINANTS_RELATIVE_SPDS
>>> spd = ILLUMINANTS_RELATIVE_SPDS['F2']
>>> colour_quality_scale(spd)
64.6863391...
```

colour.quality

CQS_Specification	Defines the *Colour Quality Scale* (CQS) colour quality specification.

colour.quality.CQS_Specification

class colour.quality.**CQS_Specification**

Defines the *Colour Quality Scale* (CQS) colour quality specification.

Parameters

- **name** (unicode) – Name of the test spectral power 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. 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.
- **Q_as** (dict) – Individual *Colour Quality Scale* (CQS) data for each sample.
- **colorimetry_data** (tuple) – Colorimetry data for the test and reference computations.

References

- *[DO10]*
- *[OD08]*

Create new instance of CQS_Specification(name, Q_a, Q_f, Q_p, Q_g, Q_d, Q_as, colorimetry_data)

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

Reflectance Recovery

- *CIE XYZ Colourspace to Spectral*
- *Smits (1999)*
- *Meng, Simon and Hanika (2015)*

CIE XYZ Colourspace to Spectral

colour

XYZ_to_spectral(XYZ[, method])	Recovers the spectral power distribution of given *CIE XYZ* tristimulus values using given method.
REFLECTANCE_RECOVERY_METHODS	Supported reflectance recovery methods.

colour.XYZ_to_spectral

`colour.XYZ_to_spectral`(*XYZ*, *method='Meng 2015'*, ***kwargs*)

Recovers the spectral power distribution of given *CIE XYZ* tristimulus values using given method.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values to recover the spectral power distribution from.
- **method** (unicode, optional) – **{'Meng 2015', 'Smits 1999'}**, Computation method.

Other Parameters

- **cmfs** (*XYZ_ColourMatchingFunctions*) – {colour.recovery.XYZ_to_spectral_Meng2015()}, Standard observer colour matching functions.
- **interval** (*numeric, optional*) – {colour.recovery.XYZ_to_spectral_Meng2015()}, Wavelength λ_i range interval in nm. The smaller interval is, the longer the computations will be.
- **tolerance** (*numeric, optional*) – {colour.recovery.XYZ_to_spectral_Meng2015()}, Tolerance for termination. The lower tolerance is, the smoother the recovered spectral power distribution will be.
- **maximum_iterations** (*int, optional*) – {colour.recovery.XYZ_to_spectral_Meng2015()}, Maximum number of iterations to perform.

Returns Recovered spectral power distribution.

Return type *SpectralPowerDistribution*

Notes

- *Smits (1999)* method will internally convert given *CIE XYZ* tristimulus values to *RGB* colourspace array assuming equal energy illuminant *E*.

References

- *[MSHD15]*
- *[Smi99]*

Examples

Meng (2015) reflectance recovery:

```
>>> from colour.utilities import numpy_print_options
>>> from colour.colorimetry import spectral_to_XYZ_integration
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> spd = XYZ_to_spectral(XYZ, interval=10)
>>> with numpy_print_options(suppress=True):
...     spd
SpectralPowerDistribution([[ 360.        ,    0.0788075...],
                           [ 370.        ,    0.0788543...],
                           [ 380.        ,    0.0788825...],
                           [ 390.        ,    0.0788714...],
```

```
                            [ 400.        ,    0.0788911...],
                            [ 410.        ,    0.07893  ...],
                            [ 420.        ,    0.0797471...],
                            [ 430.        ,    0.0813339...],
                            [ 440.        ,    0.0840145...],
                            [ 450.        ,    0.0892826...],
                            [ 460.        ,    0.0965359...],
                            [ 470.        ,    0.1053176...],
                            [ 480.        ,    0.1150921...],
                            [ 490.        ,    0.1244252...],
                            [ 500.        ,    0.1326083...],
                            [ 510.        ,    0.1390282...],
                            [ 520.        ,    0.1423548...],
                            [ 530.        ,    0.1414636...],
                            [ 540.        ,    0.1365195...],
                            [ 550.        ,    0.1277319...],
                            [ 560.        ,    0.1152622...],
                            [ 570.        ,    0.1004513...],
                            [ 580.        ,    0.0844187...],
                            [ 590.        ,    0.0686863...],
                            [ 600.        ,    0.0543013...],
                            [ 610.        ,    0.0423486...],
                            [ 620.        ,    0.0333861...],
                            [ 630.        ,    0.0273558...],
                            [ 640.        ,    0.0233407...],
                            [ 650.        ,    0.0211208...],
                            [ 660.        ,    0.0197248...],
                            [ 670.        ,    0.0187157...],
                            [ 680.        ,    0.0181510...],
                            [ 690.        ,    0.0179691...],
                            [ 700.        ,    0.0179247...],
                            [ 710.        ,    0.0178665...],
                            [ 720.        ,    0.0178005...],
                            [ 730.        ,    0.0177570...],
                            [ 740.        ,    0.0177090...],
                            [ 750.        ,    0.0175743...],
                            [ 760.        ,    0.0175058...],
                            [ 770.        ,    0.0174492...],
                            [ 780.        ,    0.0174984...],
                            [ 790.        ,    0.0175667...],
                            [ 800.        ,    0.0175657...],
                            [ 810.        ,    0.0175319...],
                            [ 820.        ,    0.0175184...],
                            [ 830.        ,    0.0175390...]],
                           interpolator=SpragueInterpolator,
                           interpolator_args={},
                           extrapolator=Extrapolator,
                           extrapolator_args={...})
>>> spectral_to_XYZ_integration(spd) / 100
array([ 0.0705100...,  0.1007987...,  0.0956738...])
```

Smits (1999) reflectance recovery:

```
>>> spd = XYZ_to_spectral(XYZ, method='Smits 1999')
>>> with numpy_print_options(suppress=True):
...     spd
SpectralPowerDistribution([[ 380.        ,    0.0908046...],
                            [ 417.7778    ,    0.0887761...],
```

```
                              [ 455.5556    ,    0.0939795...],
                              [ 493.3333    ,    0.1236033...],
                              [ 531.1111    ,    0.1315788...],
                              [ 568.8889    ,    0.1293411...],
                              [ 606.6667    ,    0.0392680...],
                              [ 644.4444    ,    0.0214496...],
                              [ 682.2222    ,    0.0214496...],
                              [ 720.        ,    0.0215462...]],
                             interpolator=CubicSplineInterpolator,
                             interpolator_args={},
                             extrapolator=Extrapolator,
                             extrapolator_args={...})
>>> spectral_to_XYZ_integration(spd) / 100
array([ 0.0753341...,  0.1054586...,  0.0977855...])
```

colour.REFLECTANCE_RECOVERY_METHODS

colour.**REFLECTANCE_RECOVERY_METHODS = CaseInsensitiveMapping({'Meng 2015': ..., 'Smits 1999': ...})**

Supported reflectance recovery methods.

References

- *[MSHD15]*
- *[Smi99]*

REFLECTANCE_RECOVERY_METHODS [CaseInsensitiveMapping] **{'Meng 2015', 'Smits 1999'}**

Smits (1999)

colour.recovery

RGB_to_spectral_Smits1999(RGB)	Recovers the spectral power distribution of given *RGB* colourspace array using *Smits (1999)* method.
SMITS_1999_SPDS	*Smits (1999)* spectral power distributions.

colour.recovery.RGB_to_spectral_Smits1999

colour.recovery.**RGB_to_spectral_Smits1999**(*RGB*)

Recovers the spectral power distribution of given *RGB* colourspace array using *Smits (1999)* method.

Parameters **RGB** (array_like, (3,)) – *RGB* colourspace array to recover the spectral power distribution from.

Returns Recovered spectral power distribution.

Return type *SpectralPowerDistribution*

References

- *[Smi99]*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> RGB = np.array([0.02144962, 0.13154603, 0.09287601])
>>> with numpy_print_options(suppress=True):
...     RGB_to_spectral_Smits1999(RGB)
SpectralPowerDistribution([[ 380.        ,    0.0908046...],
                           [ 417.7778    ,    0.0887761...],
                           [ 455.5556    ,    0.0939795...],
                           [ 493.3333    ,    0.1236033...],
                           [ 531.1111    ,    0.1315788...],
                           [ 568.8889    ,    0.1293411...],
                           [ 606.6667    ,    0.0392680...],
                           [ 644.4444    ,    0.0214496...],
                           [ 682.2222    ,    0.0214496...],
                           [ 720.        ,    0.0215463...]],
                          interpolator=CubicSplineInterpolator,
                          interpolator_args={},
                          extrapolator=Extrapolator,
                          extrapolator_args={...})
```

colour.recovery.SMITS_1999_SPDS

`colour.recovery.`**`SMITS_1999_SPDS = CaseInsensitiveMapping({u'blue': ..., u'yellow': ..., u'green': ..., u'cy`**

Smits (1999) spectral power distributions.

References

- *[Smi99]*

SMITS_1999_SPDS : CaseInsensitiveMapping

Meng, Simon and Hanika (2015)

`colour.recovery`

`XYZ_to_spectral_Meng2015`(XYZ[, cmfs, . . .])	Recovers the spectral power distribution of given *CIE XYZ* tristimulus values using *Meng et alii (2015)* method.

colour.recovery.XYZ_to_spectral_Meng2015

colour.recovery.**XYZ_to_spectral_Meng2015**(*XYZ, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), interval=5, tolerance=1e-10, maximum_iterations=2000*)

Recovers the spectral power distribution of given *CIE XYZ* tristimulus values using *Meng et alii (2015)* method.

Parameters

- **XYZ** (array_like, (3,)) – *CIE XYZ* tristimulus values to recover the spectral power distribution from.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **interval** (numeric, optional) – Wavelength λ_i range interval in nm. The smaller interval is, the longer the computations will be.
- **tolerance** (numeric, optional) – Tolerance for termination. The lower tolerance is, the smoother the recovered spectral power distribution will be.
- **maximum_iterations** (int, optional) – Maximum number of iterations to perform.

Returns Recovered spectral power distribution.

Return type *SpectralPowerDistribution*

Notes

- 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.spectral_to_XYZ_ASTME30815() definition that handles only measurement interval of 1, 5, 10 or 20nm.

References

- *[MSHD15]*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.07049534, 0.10080000, 0.09558313])
>>> spd = XYZ_to_spectral_Meng2015(XYZ, interval=10)
>>> with numpy_print_options(suppress=True):
...     spd
SpectralPowerDistribution([[ 360.        ,    0.0788075...],
                           [ 370.        ,    0.0788543...],
                           [ 380.        ,    0.0788825...],
                           [ 390.        ,    0.0788714...],
                           [ 400.        ,    0.0788911...],
                           [ 410.        ,    0.07893  ...],
                           [ 420.        ,    0.0797471...],
                           [ 430.        ,    0.0813339...],
                           [ 440.        ,    0.0840145...],
```

```
                           [ 450.          ,    0.0892826...],
                           [ 460.          ,    0.0965359...],
                           [ 470.          ,    0.1053176...],
                           [ 480.          ,    0.1150921...],
                           [ 490.          ,    0.1244252...],
                           [ 500.          ,    0.1326083...],
                           [ 510.          ,    0.1390282...],
                           [ 520.          ,    0.1423548...],
                           [ 530.          ,    0.1414636...],
                           [ 540.          ,    0.1365195...],
                           [ 550.          ,    0.1277319...],
                           [ 560.          ,    0.1152622...],
                           [ 570.          ,    0.1004513...],
                           [ 580.          ,    0.0844187...],
                           [ 590.          ,    0.0686863...],
                           [ 600.          ,    0.0543013...],
                           [ 610.          ,    0.0423486...],
                           [ 620.          ,    0.0333861...],
                           [ 630.          ,    0.0273558...],
                           [ 640.          ,    0.0233407...],
                           [ 650.          ,    0.0211208...],
                           [ 660.          ,    0.0197248...],
                           [ 670.          ,    0.0187157...],
                           [ 680.          ,    0.0181510...],
                           [ 690.          ,    0.0179691...],
                           [ 700.          ,    0.0179247...],
                           [ 710.          ,    0.0178665...],
                           [ 720.          ,    0.0178005...],
                           [ 730.          ,    0.0177570...],
                           [ 740.          ,    0.0177090...],
                           [ 750.          ,    0.0175743...],
                           [ 760.          ,    0.0175058...],
                           [ 770.          ,    0.0174492...],
                           [ 780.          ,    0.0174984...],
                           [ 790.          ,    0.0175667...],
                           [ 800.          ,    0.0175657...],
                           [ 810.          ,    0.0175319...],
                           [ 820.          ,    0.0175184...],
                           [ 830.          ,    0.0175390...]],
                          interpolator=SpragueInterpolator,
                          interpolator_args={},
                          extrapolator=Extrapolator,
                          extrapolator_args={...})
>>> spectral_to_XYZ_integration(spd) / 100
array([ 0.0705100...,  0.1007987...,  0.0956738...])
```

Colour Temperature

- *Hernandez-Andres, Lee and Romero (1999)*
- *Kang, Moon, Hong, Lee, Cho and Kim (2002)*
- *CIE Illuminant D Series*

Correlated Colour Temperature

colour

CCT_to_uv(CCT[, 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.
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_xy(CCT[, method])	Returns the *CIE XYZ* tristimulus values *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 XYZ* tristimulus values *xy* chromaticity coordinates computation methods.
xy_to_CCT(xy[, method])	Returns the correlated colour temperature T_{cp} from given *CIE XYZ* tristimulus values *xy* chromaticity coordinates using given method.
XY_TO_CCT_METHODS	Supported *CIE XYZ* tristimulus values *xy* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

colour.CCT_to_uv

colour.**CCT_to_uv**(*CCT*, *method=u'Ohno 2013'*, ***kwargs*)

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.

Parameters

- **CCT** (numeric) – Correlated colour temperature T_{cp}.
- **method** (unicode, optional) – **{'Ohno 2013', 'Robertson 1968', 'Krystek 1985}**, Computation method.

Other Parameters

- **D_uv** (*numeric*) – {CCT_to_uv_Ohno2013, CCT_to_uv_Robertson1968()}, Δ_{uv}.
- **cmfs** (*XYZ_ColourMatchingFunctions, optional*) – {colour.temperature.CCT_to_uv_Ohno2013()}, Standard observer colour matching functions.

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

Return type ndarray

References

- *[AdobeSystems13a]*
- *[AdobeSystems13b]*
- *[Kry85]*
- *[Ohn14]*
- *[WS00c]*

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> CCT = 6507.47380460
>>> D_uv = 0.00322335
>>> CCT_to_uv(CCT, D_uv=D_uv, cmfs=cmfs)
array([ 0.1977999...,  0.3121999...])
```

colour.CCT_TO_UV_METHODS

colour.**CCT_TO_UV_METHODS = CaseInsensitiveMapping({u'Ohno 2013': ..., u'Krystek 1985': ..., u'robertson1968**

Supported correlated colour temperature T_{cp} to *CIE UCS* colourspace *uv* chromaticity coordinates computation methods.

References

- *[AdobeSystems13a]*
- *[AdobeSystems13b]*
- *[Kry85]*
- *[Ohn14]*
- *[WS00c]*

CCT_TO_UV_METHODS [CaseInsensitiveMapping] **{'Ohno 2013', 'Robertson 1968', 'Krystek 1985}**

Aliases:

- 'ohno2013': 'Ohno 2013'
- 'robertson1968': 'Robertson 1968'

colour.uv_to_CCT

`colour.uv_to_CCT(uv, method=u'Ohno 2013', **kwargs)`

Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using given method.

Parameters

- **uv** (array_like) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **method** (unicode, optional) – **{'Ohno 2013', 'Robertson 1968'}**, Computation method.

Other Parameters

- **cmfs** (*XYZ_ColourMatchingFunctions, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Standard observer colour matching functions.
- **start** (*numeric, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Temperature range start in kelvins.
- **end** (*numeric, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Temperature range end in kelvins.
- **count** (*int, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Temperatures count in the planckian tables.
- **iterations** (*int, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Number of planckian tables to generate.

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type ndarray

References

- *[AdobeSystems13a]*
- *[AdobeSystems13b]*
- *[Ohn14]*
- *[WS00c]*

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> uv = np.array([0.1978, 0.3122])
>>> uv_to_CCT(uv, cmfs=cmfs)
array([  6.5074738...e+03,   3.2233461...e-03])
```

colour.UV_TO_CCT_METHODS

`colour.UV_TO_CCT_METHODS = CaseInsensitiveMapping({u'Ohno 2013': ..., u'robertson1968': ..., u'ohno2013': .`

Supported *CIE UCS* colourspace *uv* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

- *[AdobeSystems13a]*
- *[AdobeSystems13b]*
- *[Ohn14]*
- *[WS00c]*

UV_TO_CCT_METHODS [CaseInsensitiveMapping] **{'Ohno 2013', 'Robertson 1968'}**

Aliases:

- 'ohno2013': 'Ohno 2013'
- 'robertson1968': 'Robertson 1968'

colour.CCT_to_xy

`colour.CCT_to_xy`(*CCT*, *method=u'Kang 2002'*)

Returns the *CIE XYZ* tristimulus values *xy* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.

Parameters

- **CCT** (`numeric or array_like`) – Correlated colour temperature T_{cp}.
- **method** (`unicode, optional`) – **{'Kang 2002', 'CIE Illuminant D Series'}**, Computation method.

Returns *xy* chromaticity coordinates.

Return type ndarray

References

- *[KMH+02]*
- *[Wiki]*
- *[WS00b]*

colour.CCT_TO_XY_METHODS

`colour.CCT_TO_XY_METHODS = CaseInsensitiveMapping({u'cie_d': ..., u'CIE Illuminant D Series': ..., u'kang20`

Supported correlated colour temperature T_{cp} to *CIE XYZ* tristimulus values *xy* chromaticity coordinates computation methods.

References

- *[KMH+02]*
- *[Wiki]*
- *[WS00b]*

CCT_TO_XY_METHODS [CaseInsensitiveMapping] **{'Kang 2002', 'CIE Illuminant D Series'}**

Aliases:

- 'kang2002': 'Kang 2002'
- 'cie_d': 'Hernandez 1999'

colour.xy_to_CCT

colour.**xy_to_CCT**(*xy*, *method=u'McCamy 1992'*)

Returns the correlated colour temperature T_{cp} from given *CIE XYZ* tristimulus values *xy* chromaticity coordinates using given method.

Parameters

- **xy** (array_like) – *xy* chromaticity coordinates.
- **method** (unicode, optional) – **{'McCamy 1992', 'Hernandez 1999'}**, Computation method.

Returns Correlated colour temperature T_{cp}.

Return type numeric or ndarray

References

- *[HernandezAndresLR99]*
- *[Wika]*
- *[Wiki]*

colour.XY_TO_CCT_METHODS

colour.**XY_TO_CCT_METHODS = CaseInsensitiveMapping({u'hernandez1999': ..., u'Hernandez 1999': ..., u'McCamy**

Supported *CIE XYZ* tristimulus values *xy* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

- *[HernandezAndresLR99]*
- *[Wika]*
- *[Wiki]*

XY_TO_CCT_METHODS [CaseInsensitiveMapping] **{'McCamy 1992', 'Hernandez 1999'}**

Aliases:

- 'mccamy1992': 'McCamy 1992'
- 'hernandez1999': 'Hernandez 1999'

Robertson (1968)

`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.
`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.

colour.temperature.CCT_to_uv_Robertson1968

`colour.temperature.`**`CCT_to_uv_Robertson1968`**(*CCT*, *D_uv=0*)

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} and Δ_{uv} using *Roberston (1968)* method.

Parameters

- **CCT** (`numeric`) – Correlated colour temperature T_{cp}.
- **D_uv** (`numeric`) – Δ_{uv}.

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

Return type ndarray

References

- *[AdobeSystems13b]*
- *[WS00c]*

Examples

```
>>> CCT = 6500.0081378199056
>>> D_uv = 0.008333331244225
>>> CCT_to_uv_Robertson1968(CCT, D_uv)
array([ 0.1937413...,  0.3152210...])
```

colour.temperature.uv_to_CCT_Robertson1968

`colour.temperature.`**`uv_to_CCT_Robertson1968`**(*uv*)

Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Roberston (1968)* method.

Parameters **uv** (`array_like`) – *CIE UCS* colourspace *uv* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type ndarray

References

- *[AdobeSystems13a]*
- *[WS00c]*

Examples

```
>>> uv = np.array([0.193741375998230, 0.315221043940594])
>>> uv_to_CCT_Robertson1968(uv)
array([  6.5000162...e+03,   8.3333289...e-03])
```

Krystek (1985)

colour.temperature

CCT_to_uv_Krystek1985(CCT)	Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using *Krystek (1985)* method.

colour.temperature.CCT_to_uv_Krystek1985

colour.temperature.**CCT_to_uv_Krystek1985**(*CCT*)

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using *Krystek (1985)* method.

Parameters **CCT** (numeric) – Correlated colour temperature T_{cp}.

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

Return type ndarray

Notes

- *Krystek (1985)* method computations are valid for correlated colour temperature T_{cp} in domain [1000, 15000].

References

- *[Kry85]*

Examples

```
>>> CCT_to_uv_Krystek1985(6504.38938305)
array([ 0.1837669...,  0.3093443...])
```

Ohno (2013)

`colour.temperature`

`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.
`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.

colour.temperature.CCT_to_uv_Ohno2013

`colour.temperature.`**`CCT_to_uv_Ohno2013`**(*CCT, D_uv=0, 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** (`numeric`) – Correlated colour temperature T_{cp}.
- **D_uv** (`numeric, optional`) – Δ_{uv}.
- **cmfs** (`XYZ_ColourMatchingFunctions, optional`) – Standard observer colour matching functions.

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

Return type ndarray

References

- *[Ohn14]*

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> CCT = 6507.4342201047066
>>> D_uv = 0.003223690901513
>>> CCT_to_uv_Ohno2013(CCT, D_uv, cmfs)
array([ 0.1977999...,  0.3122004...])
```

colour.temperature.uv_to_CCT_Ohno2013

`colour.temperature.`**`uv_to_CCT_Ohno2013`**(*uv, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), start=1000, end=100000, count=10, iterations=6*)

Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* 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

- *[Ohn14]*

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> uv = np.array([0.1978, 0.3122])
>>> uv_to_CCT_Ohno2013(uv, cmfs)
array([  6.5074738...e+03,   3.2233461...e-03])
```

Hernandez-Andres, Lee and Romero (1999)

`colour.temperature`

xy_to_CCT_Hernandez1999(xy)	Returns the correlated colour temperature T_{cp} from given *CIE XYZ* tristimulus values *xy* chromaticity coordinates using *Hernandez-Andres et alii (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 XYZ* tristimulus values *xy* chromaticity coordinates using *Hernandez-Andres et alii (1999)* method.

Parameters **xy** (array_like) – *xy* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}.

Return type numeric

References

- [HernandezAndresLR99]

Examples

```
>>> xy = np.array([0.31270, 0.32900])
>>> xy_to_CCT_Hernandez1999(xy)
6500.7420431...
```

Kang, Moon, Hong, Lee, Cho and Kim (2002)

`colour.temperature`

`CCT_to_xy_Kang2002`(CCT)	Returns the *CIE XYZ* tristimulus values *xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Kang et alii (2002)* method.

colour.temperature.CCT_to_xy_Kang2002

`colour.temperature.`**`CCT_to_xy_Kang2002`**`(CCT)`

Returns the *CIE XYZ* tristimulus values *xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Kang et alii (2002)* method.

Parameters **`CCT`** (`numeric or array_like`) – Correlated colour temperature T_{cp}.

Returns *xy* chromaticity coordinates.

Return type ndarray

Raises `ValueError` – If the correlated colour temperature is not in appropriate domain.

References

- [KMH+02]

Examples

```
>>> CCT_to_xy_Kang2002(6504.38938305)
array([ 0.313426...,  0.3235959...])
```

CIE Illuminant D Series

`colour.temperature`

CCT_to_xy_CIE_D(CCT)	Converts from the correlated colour temperature T_{cp} of a *CIE Illuminant D Series* to the chromaticity of that *CIE Illuminant D Series* illuminant.

colour.temperature.CCT_to_xy_CIE_D

colour.temperature.**CCT_to_xy_CIE_D**(*CCT*)

Converts from the correlated colour temperature T_{cp} of a *CIE Illuminant D Series* to the chromaticity of that *CIE Illuminant D Series* illuminant.

Parameters **CCT** (numeric or array_like) – Correlated colour temperature T_{cp}.

Returns *xy* chromaticity coordinates.

Return type ndarray

Raises ValueError – If the correlated colour temperature is not in appropriate domain.

References

- *[WS00b]*

Examples

```
>>> CCT_to_xy_CIE_D(6504.38938305)
array([ 0.3127077...,  0.3291128...])
```

Utilities

Common

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.

Continued on next page

Table 3.201 – continued from previous page

batch(iterable[, k])	Returns a batch generator from given iterable.
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_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.
filter_kwargs(function, **kwargs)	Filters keyword arguments incompatible with the given function signature.
first_item(a)	Return the first item of an iterable.

colour.utilities.handle_numpy_errors

colour.utilities.**handle_numpy_errors**(***kwargs*)

Decorator for handling *Numpy* errors.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments.

Returns

Return type object

References

- *[KPK11]*

Examples

```
>>> import numpy
>>> @handle_numpy_errors(all='ignore')
... def f():
...     1 / numpy.zeros(3)
>>> f()
```

colour.utilities.ignore_numpy_errors

colour.utilities.**ignore_numpy_errors**(*function*)

Wrapper for given function.

colour.utilities.raise_numpy_errors

colour.utilities.**raise_numpy_errors**(*function*)

Wrapper for given function.

colour.utilities.print_numpy_errors

colour.utilities.**print_numpy_errors**(*function*)
Wrapper for given function.

colour.utilities.warn_numpy_errors

colour.utilities.**warn_numpy_errors**(*function*)
Wrapper for given function.

colour.utilities.ignore_python_warnings

colour.utilities.**ignore_python_warnings**(*function*)
Decorator for ignoring *Python* warnings.

Parameters **function** (object) – Function to decorate.

Returns

Return type object

Examples

```
>>> @ignore_python_warnings
... def f():
...     warnings.warn('This is an ignored warning!')
>>> f()
```

colour.utilities.batch

colour.utilities.**batch**(*iterable*, *k=3*)
Returns a batch generator from given iterable.

Parameters

- **iterable** (iterable) – Iterable to create batches from.
- **k** (integer) – Batches size.

Returns Is *string_like* variable.

Return type bool

Examples

```
>>> batch(tuple(range(10)))
<generator object batch at 0x...>
```

colour.utilities.is_openimageio_installed

`colour.utilities.is_openimageio_installed(raise_exception=False)`

Returns if *OpenImageIO* is installed and available.

Parameters **raise_exception** (bool) – Raise exception if *OpenImageIO* is unavailable.

Returns Is *OpenImageIO* installed.

Return type bool

Raises ImportError – If *OpenImageIO* is not installed.

colour.utilities.is_pandas_installed

`colour.utilities.is_pandas_installed(raise_exception=False)`

Returns if *Pandas* is installed and available.

Parameters **raise_exception** (bool) – Raise exception if *Pandas* is unavailable.

Returns Is *Pandas* installed.

Return type bool

Raises ImportError – If *Pandas* is not installed.

colour.utilities.is_iterable

`colour.utilities.is_iterable(a)`

Returns if given a variable is iterable.

Parameters **a** (object) – Variable to check the iterability.

Returns a variable iterability.

Return type bool

Examples

```
>>> is_iterable([1, 2, 3])
True
>>> is_iterable(1)
False
```

colour.utilities.is_string

`colour.utilities.is_string(a)`

Returns if given a variable is a *string* like variable.

Parameters **a** (object) – Data to test.

Returns Is a variable a *string* like variable.

Return type bool

Examples

```
>>> is_string("I'm a string!")
True
>>> is_string(["I'm a string!"])
False
```

colour.utilities.is_numeric

`colour.utilities.is_numeric(a)`

Returns if given a variable is a number.

Parameters **a** (`object`) – Variable to check.

Returns Is a variable a number.

Return type bool

Examples

```
>>> is_numeric(1)
True
>>> is_numeric((1,))
False
```

colour.utilities.is_integer

`colour.utilities.is_integer(a)`

Returns if given a variable is an integer under given threshold.

Parameters **a** (`object`) – Variable to check.

Returns Is a variable an integer.

Return type bool

Notes

- The determination threshold is defined by the `colour.algebra.common.INTEGER_THRESHOLD` attribute.

Examples

```
>>> is_integer(1)
True
>>> is_integer(1.01)
False
```

colour.utilities.filter_kwargs

colour.utilities.**filter_kwargs**(*function*, ***kwargs*)

Filters keyword arguments incompatible with the given function signature.

Parameters **function** (callable) – Callable to filter the incompatible keyword arguments.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments.

Returns Filtered keyword arguments.

Return type dict

Examples

```
>>> def fn_a(a):
...     return a
>>> def fn_b(a, b=0):
...     return a, b
>>> def fn_c(a, b=0, c=0):
...     return a, b, c
>>> fn_a(1, **filter_kwargs(fn_a, b=2, c=3))
1
>>> fn_b(1, **filter_kwargs(fn_b, b=2, c=3))
(1, 2)
>>> fn_c(1, **filter_kwargs(fn_c, b=2, c=3))
(1, 2, 3)
```

colour.utilities.first_item

colour.utilities.**first_item**(*a*)

Return the first item of an iterable.

Parameters **a** (object) – Iterable to get the first item from.

Returns

Return type object

Raises StopIteration – If the iterable is empty.

Examples

```
>>> a = range(10)
>>> first_item(a)
0
```

Array

colour.utilities

as_numeric(a[, type_])	Converts given a variable to *numeric*.

Continued on next page

Table 3.202 – continued from previous page

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)	Stacks arrays in sequence along the last axis (tail).
tsplit(a)	Splits arrays in sequence along the last axis (tail).
row_as_diagonal(a)	Returns the per row diagonal matrices of the given array.
dot_vector(m, v)	Convenient wrapper around `np.einsum()` with the following subscripts: '*...ij,...j->...i*'.
dot_matrix(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.
fill_nan(a[, method, default])	Fills given array NaNs according to given method.
ndarray_write(*args, **kwds)	A context manager setting given array writeable to perform an operation and then read-only.

colour.utilities.as_numeric

colour.utilities.**as_numeric**(*a*, *type_=<type 'numpy.float64'>*)

Converts given a variable to *numeric*. In the event where a cannot be converted, it is passed as is.

Parameters

- **a** (object) – Variable to convert.
- **type** (object) – Type to use for conversion.

Returns a variable converted to *numeric*.

Return type ndarray

Examples

```
>>> as_numeric(np.array([1]))
1.0
>>> as_numeric(np.arange(10))
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

colour.utilities.as_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])
>>> closest_indexes(a, 63)
array([3])
>>> closest_indexes(a, [63, 25])
array([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 between in 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

- *[Yor14]*

Examples

```
>>> a = np.array([0.50, 0.60])
>>> b = np.linspace(0, 10, 101)
>>> np.in1d(a, b)
array([ True, False], dtype=bool)
>>> in_array(a, b)
array([ True,  True], dtype=bool)
```

colour.utilities.tstack

colour.utilities.**tstack**(*a*)

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.

Returns

Return type ndarray

Examples

```
>>> a = 0
>>> tstack((a, a, a))
array([0, 0, 0])
>>> a = np.arange(0, 6)
>>> tstack((a, a, a))
array([[0, 0, 0],
       [1, 1, 1],
       [2, 2, 2],
       [3, 3, 3],
```

```
      [4, 4, 4],
      [5, 5, 5]])
>>> a = np.reshape(a, (1, 6))
>>> tstack((a, a, a))
array([[[0, 0, 0],
        [1, 1, 1],
        [2, 2, 2],
        [3, 3, 3],
        [4, 4, 4],
        [5, 5, 5]]])
>>> a = np.reshape(a, (1, 1, 6))
>>> tstack((a, a, a))
array([[[[0, 0, 0],
         [1, 1, 1],
         [2, 2, 2],
         [3, 3, 3],
         [4, 4, 4],
         [5, 5, 5]]]])
```

colour.utilities.tsplit

colour.utilities.**tsplit**(*a*)

Splits arrays in sequence along the last axis (tail).

Parameters **a** (array_like) – Array to perform the splitting.

Returns

Return type ndarray

Examples

```
>>> a = np.array([0, 0, 0])
>>> tsplit(a)
array([0, 0, 0])
>>> a = np.array(
...     [[0, 0, 0],
...      [1, 1, 1],
...      [2, 2, 2],
...      [3, 3, 3],
...      [4, 4, 4],
...      [5, 5, 5]]
... )
>>> tsplit(a)
array([[0, 1, 2, 3, 4, 5],
       [0, 1, 2, 3, 4, 5],
       [0, 1, 2, 3, 4, 5]])
>>> a = np.array(
...     [[[0, 0, 0],
...       [1, 1, 1],
...       [2, 2, 2],
...       [3, 3, 3],
...       [4, 4, 4],
...       [5, 5, 5]]]
... )
```

```
>>> tsplit(a)
array([[[0, 1, 2, 3, 4, 5]],

       [[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.

Returns

Return type ndarray

References

- *[Cas14]*

Examples

```
>>> a = np.array(
...     [[0.25891593, 0.07299478, 0.36586996],
...       [0.30851087, 0.37131459, 0.16274825],
...       [0.71061831, 0.67718718, 0.09562581],
...       [0.71588836, 0.76772047, 0.15476079],
...       [0.92985142, 0.22263399, 0.88027331]]
... )
>>> row_as_diagonal(a)
array([[[ 0.25891593,  0.        ,  0.        ],
        [ 0.        ,  0.07299478,  0.        ],
        [ 0.        ,  0.        ,  0.36586996]],

       [[ 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.dot_vector

colour.utilities.**dot_vector**(*m*, *v*)

Convenient wrapper around `np.einsum()` with the following subscripts: *'…ij,…j->…i'*.

It performs the dot product of two arrays where *m* parameter is expected to be an array of 3x3 matrices and parameter *v* an array of vectors.

Parameters

- **m** (array_like) – Array of 3x3 matrices.
- **v** (array_like) – Array of vectors.

Returns

Return type ndarray

Examples

```
>>> m = np.array(
...     [[0.7328, 0.4296, -0.1624],
...      [-0.7036, 1.6975, 0.0061],
...      [0.0030, 0.0136, 0.9834]]
... )
>>> m = np.reshape(np.tile(m, (6, 1)), (6, 3, 3))
>>> v = np.array([0.07049534, 0.10080000, 0.09558313])
>>> v = np.tile(v, (6, 1))
>>> dot_vector(m, v)
array([[ 0.0794399...,  0.1220905...,  0.0955788...],
       [ 0.0794399...,  0.1220905...,  0.0955788...],
       [ 0.0794399...,  0.1220905...,  0.0955788...],
       [ 0.0794399...,  0.1220905...,  0.0955788...],
       [ 0.0794399...,  0.1220905...,  0.0955788...],
       [ 0.0794399...,  0.1220905...,  0.0955788...]])
```

colour.utilities.dot_matrix

colour.utilities.**dot_matrix**(*a*, *b*)

Convenient wrapper around `np.einsum()` with the following subscripts: *'…ij,…jk->…ik'*.

It performs the dot product of two arrays where *a* parameter is expected to be an array of 3x3 matrices and parameter *b* another array of of 3x3 matrices.

Parameters

- **a** (array_like) – Array of 3x3 matrices.
- **b** (array_like) – Array of 3x3 matrices.

Returns

Return type ndarray

Examples

```
>>> a = np.array(
...     [[0.7328, 0.4296, -0.1624],
...      [-0.7036, 1.6975, 0.0061],
...      [0.0030, 0.0136, 0.9834]]
... )
>>> a = np.reshape(np.tile(a, (6, 1)), (6, 3, 3))
>>> b = a
>>> dot_matrix(a, b)
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

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.fill_nan

colour.utilities.**fill_nan**(*a*, *method=u'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**(**args*, ***kwds*)

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
```

Data Structures

colour.utilities

CaseInsensitiveMapping([data])	Implements a case-insensitive mutable mapping / *dict* object.
Lookup	Extends *dict* type to provide a lookup by value(s).
Structure(*args, **kwargs)	Defines an object similar to C/C++ structured type.

colour.utilities.CaseInsensitiveMapping

class colour.utilities.**CaseInsensitiveMapping**(*data=None*, ***kwargs*)

Implements a case-insensitive mutable mapping / *dict* object.

Allows values retrieving from keys while ignoring the key case. The keys are expected to be unicode or string-like objects supporting the `str.lower()` method.

Parameters **data** (dict) – *dict* of data to store into the mapping at initialisation.

Other Parameters ****kwargs** (*dict, optional*) – Key / Value pairs to store into the mapping at initialisation.

__setitem__()

__getitem__()

__delitem__()

__contains__()

__iter__()

__len__()

__eq__()

__ne__()

__repr__()

copy()

lower_items()

> **Warning:** The keys are expected to be unicode or string-like objects.

References

- *[Rei]*

Examples

```
>>> methods = CaseInsensitiveMapping({'McCamy': 1, 'Hernandez': 2})
>>> methods['mccamy']
1
```

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

Methods

`__init__`([data])	
`clear`(() -> None. Remove all items from D.)	
`copy`()	Returns a copy of the mapping.
`get`((k[,d]) -> D[k] if k in D, …)	
`items`(() -> list of D's (key, value) pairs, …)	
`iteritems`(() -> an iterator over the (key, …)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(…)	
`keys`(() -> list of D's keys)	
`lower_items`()	Iterates over the lower items names.
`pop`((k[,d]) -> v, …)	If key is not found, d is returned if given, otherwise KeyError is raised.
`popitem`(() -> (k, v), …)	as a 2-tuple; but raise KeyError if D is empty.
`setdefault`((k[,d]) -> D.get(k,d), …)	
`update`(([E, …)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	

colour.utilities.Lookup

class colour.utilities.**Lookup**

Extends *dict* type to provide a lookup by value(s).

first_key_from_value()

keys_from_value()

References

- *[Mana]*

Examples

```
>>> person = Lookup(first_name='Doe', last_name='John', gender='male')
>>> person.first_key_from_value('Doe')
'first_name'
>>> persons = Lookup(John='Doe', Jane='Doe', Luke='Skywalker')
>>> sorted(persons.keys_from_value('Doe'))
['Jane', 'John']
```

__init__()

x.__init__(…) initializes x; see help(type(x)) for signature

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
first_key_from_value(value)	Gets the first key with given value.
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
keys_from_value(value)	Gets the keys with given value.
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	

colour.utilities.Structure

class colour.utilities.**Structure**(**args*, ***kwargs*)

Defines an object similar to C/C++ structured type.

Other Parameters

- ***args** (*list, optional*) – Arguments.
- ****kwargs** (*dict, optional*) – Key / Value pairs.

__getattr__()

__setattr__()

__delattr__()

update()

References

- *[Manb]*

Examples

```
>>> person = Structure(first_name='Doe', last_name='John', gender='male')
>>> # Doctests skip for Python 2.x compatibility.
>>> person.first_name
'Doe'
>>> sorted(person.keys())
```

```
['first_name', 'gender', 'last_name']
>>> # Doctests skip for Python 2.x compatibility.
>>> person['gender']
'male'
```

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

Methods

__init__(*args, **kwargs)	
clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
setdefault((k[,d]) -> D.get(k,d), ...)	
update(*args, **kwargs)	Updates both keys and sibling attributes.
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	

Verbose

colour.utilities

message_box(message[, width, padding])	Prints a message inside a box.
warning(*args, **kwargs)	Issues a warning.
filter_warnings([state, colour_warnings_only])	Filters *Colour* and also optionally overall Python warnings.
suppress_warnings(*args, **kwds)	A context manager filtering *Colour* and also optionally overall Python warnings.
numpy_print_options(*args, **kwds)	A context manager implementing context changes to *Numpy* print behaviour.

colour.utilities.message_box

colour.utilities.**message_box**(*message*, *width=79*, *padding=3*)

Prints a message inside a box.

Parameters

- **message** (unicode) – Message to print.
- **width** (int, optional) – Message box width.
- **padding** (unicode) – Padding on each sides of the message.

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.

Other Parameters

- ***args** (*list, optional*) – Arguments.
- ****kwargs** (*dict, optional*) – Keywords arguments.

Returns Definition success.

Return type bool

Examples

```
>>> warning('This is a warning!')
/Users/.../colour/utilities/verbose.py:132: UserWarning: This is a warning!
```

colour.utilities.filter_warnings

`colour.utilities.`**`filter_warnings`**(*state=True*, *colour_warnings_only=True*)

Filters *Colour* and also optionally overall Python warnings.

Parameters

- **state** (`bool`, `optional`) – Warnings filter state.
- **colour_warnings_only** (`bool`, `optional`) – Whether to only filter *Colour* warnings or also overall Python warnings.

Returns Definition success.

Return type bool

Examples

Filtering *Colour* only warnings: >>> filter_warnings() True

Filtering *Colour* and also Python warnings: >>> filter_warnings(colour_warnings_only=False) True

colour.utilities.suppress_warnings

`colour.utilities.`**`suppress_warnings`**(**args*, ***kwds*)

A context manager filtering *Colour* and also optionally overall Python warnings.

Parameters **colour_warnings_only** (`bool`, `optional`) – Whether to only filter *Colour* warnings or also overall Python warnings.

colour.utilities.numpy_print_options

`colour.utilities.`**`numpy_print_options`**(**args*, ***kwds*)

A context manager implementing context changes to *Numpy* print behaviour.

Other Parameters

- ***args** (*list, optional*) – Arguments.
- ****kwargs** (*dict, optional*) – Keywords arguments.

Examples

```
>>> np.array([np.pi])
array([ 3.1415926...])
>>> with numpy_print_options(formatter={'float': '{:0.1f}'.format}):
...     np.array([np.pi])
array([3.1])
```

Ancillary Objects

`colour.utilities`

`ColourWarning`	This is the base class of *Colour* warnings.

colour.utilities.ColourWarning

exception colour.utilities.**ColourWarning**

This is the base class of *Colour* warnings. It is a subclass of `Warning`.

Colour Volume

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.
`ILLUMINANTS_OPTIMAL_COLOUR_STIMULI`	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

- Input *CIE xyY* colourspace array is in domain [0, 1].

Examples

```
>>> is_within_macadam_limits(np.array([0.3205, 0.4131, 0.51]), 'A')
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.51],
...               [0.0005, 0.0031, 0.001]])
>>> is_within_macadam_limits(a, 'A')
array([ True, False], dtype=bool)
```

colour.ILLUMINANTS_OPTIMAL_COLOUR_STIMULI

colour.**ILLUMINANTS_OPTIMAL_COLOUR_STIMULI = CaseInsensitiveMapping({u'A': ..., u'D65': ..., u'C': ...})**
Illuminants *Optimal Colour Stimuli*.

References

- *[Wikw]*

ILLUMINANTS_OPTIMAL_COLOUR_STIMULI [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

- Input *CIE XYZ* tristimulus values are in domain [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 *Lab* 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([0.3457, 0.3585])*)

Computes given *RGB* colourspace volume limits in *Lab* colourspace.

Parameters

- **colourspace** (RGB_Colourspace) – *RGB* colourspace to compute the volume of.
- **illuminant** (array_like, optional) – *Lab* colourspace *illuminant* chromaticity coordinates.

Returns *RGB* colourspace volume limits.

Return type ndarray

Examples

```
>>> from colour import sRGB_COLOURSPACE as sRGB
>>> RGB_colourspace_limits(sRGB)
array([[   0...        ,  100.0000848...],
       [ -79.2197012...,   94.6760011...],
       [-114.7814393...,   96.7261797...]])
```

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 import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_pointer_gamut_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng)
83...
```

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 import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_visible_spectrum_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng)
36...
```

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([0.3457, 0.3585])*, *chromatic_adaptation_method=u'CAT02'*, *random_generator=<function random_triplet_generator>*, *random_state=None*, *processes=None*)

Performs given *RGB* colourspace volume computation using *Monte Carlo* method and multiprocessing.

Parameters

- **colourspace** (`RGB_Colourspace`) – *RGB* colourspace to compute the volume of.
- **samples** (`numeric, optional`) – Samples count.
- **limits** (`array_like, optional`) – *Lab* colourspace volume.
- **illuminant_Lab** (`array_like, optional`) – *Lab* colourspace *illuminant* chromaticity coordinates.
- **chromatic_adaptation_method** (`unicode, optional`) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* method.
- **random_generator** (`generator, optional`) – Random triplet generator providing the random samples within the *Lab* colourspace volume.
- **random_state** (`RandomState, optional`) – Mersenne Twister pseudo-random number generator to use in the random number generator.
- **processes** (`integer, optional`) – Processes count, default to `multiprocessing.cpu_count()` definition.

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 import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> processes = 1
>>> RGB_colourspace_volume_MonteCarlo(sRGB, 10e3, random_state=prng,
...                                   processes=processes)
...
858...
```

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 import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_volume_coverage_MonteCarlo(
...     sRGB, is_within_pointer_gamut, 10e3, random_state=prng)
...
83...
```

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', ...), tolerance=None)`

Returns if given *CIE XYZ* tristimulus values are within visible spectrum volume / given colour matching functions volume.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **tolerance** (numeric, optional) – Tolerance allowed in the inside-triangle check.

Returns Is within visible spectrum.

Return type bool

Notes

- Input *CIE XYZ* tristimulus values are in domain [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)
```

3.1.2.2 Indices and tables

- genindex
- search

3.1.3 Bibliography

3.1.3.1 Indirect References

Some extra references used in the codebase but not directly part of the public api:

- *[Cen14e]*
- *[Cen14k]*
- *[Cen14h]*
- *[Cen14c]*
- *[Cen14j]*
- *[Cen14i]*
- *[Cen14g]*
- *[Cen14d]*
- *[Cen14f]*
- *[Cen14b]*
- *[Cen14a]*

- *[CIET13805d]*
- *[Hou15]*
- *[Lau12]*
- *[Mac35]*
- *[MunsellCScienceb]*
- *[Poi80]*
- *[RenewableRDCenter03]*
- *[SWD05]*
- *[WS00h]*
- *[WS00j]*
- *[WS00f]*

3.2 Examples

Chromatic Adaptation

```
>>> import colour
>>> XYZ = [0.07049534, 0.10080000, 0.09558313]
>>> A = colour.ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['A']
>>> D65 = colour.ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']
>>> colour.chromatic_adaptation(
...     XYZ, colour.xy_to_XYZ(A), colour.xy_to_XYZ(D65))
array([ 0.08398225,  0.11413379,  0.28629643])
>>> sorted(colour.CHROMATIC_ADAPTATION_METHODS.keys())
['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Von Kries']
```

Algebra

```
>>> import colour
>>> 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])
>>> colour.SpragueInterpolator(x, y)([0.25, 0.75, 5.50])
array([  6.72951612,   7.81406251,  43.77379185])
```

Spectral Computations

```
>>> import colour
>>> colour.spectral_to_XYZ(colour.LIGHT_SOURCES_RELATIVE_SPDS['Neodimium Incandescent'])
array([ 36.94726204,  32.62076174,  13.0143849 ])
>>> sorted(colour.SPECTRAL_TO_XYZ_METHODS.keys())
[u'ASTM E308-15', u'Integration', u'astm2015']
```

Blackbody Spectral Radiance Computation

```
>>> import colour
>>> colour.blackbody_spd(5000)
SpectralPowerDistribution([[  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={u'right': None, u'method': u'Constant', u'left': None})
```

Dominant, Complementary Wavelength & Colour Purity Computation

```
>>> import colour
>>> xy = [0.26415, 0.37770]
>>> xy_n = [0.31270, 0.32900]
>>> colour.dominant_wavelength(xy, xy_n)
(array(504.0),
 array([ 0.00369694,  0.63895775]),
 array([ 0.00369694,  0.63895775]))
```

Lightness Computation

```
>>> import colour
>>> colour.lightness(10.08)
24.902290269546651
>>> sorted(colour.LIGHTNESS_METHODS.keys())
[u'CIE 1976',
 u'Fairchild 2010',
 u'Glasser 1958',
 u'Lstar1976',
 u'Wyszecki 1963']
```

Luminance Computation

```
>>> import colour
>>> colour.luminance(37.98562910)
10.080000001314646
>>> sorted(colour.LUMINANCE_METHODS.keys())
[u'ASTM D1535-08',
 u'CIE 1976',
 u'Fairchild 2010',
 u'Newhall 1943',
 u'astm2008',
 u'cie1976']
```

Whiteness Computation

```
>>> import colour
>>> colour.whiteness(xy=[0.3167, 0.3334], Y=100, xy_n=[0.3139, 0.3311])
array([ 93.85 ,  -1.305])
>>> sorted(colour.WHITENESS_METHODS.keys())
[u'ASTM E313',
 u'Berger 1959',
 u'CIE 2004',
 u'Ganz 1979',
 u'Stensby 1968',
 u'Taube 1960',
 u'cie2004']
```

Yellowness Computation

```
>>> import colour
>>> XYZ = [95.00000000, 100.00000000, 105.00000000]
>>> colour.yellowness(XYZ)
11.065000000000003
>>> sorted(colour.YELLOWNESS_METHODS.keys())
[u'ASTM D1925', u'ASTM E313']
```

Luminous Flux, Efficiency & Efficacy Computation

```
>>> import colour
>>> spd = colour.LIGHT_SOURCES_RELATIVE_SPDS['Neodimium Incandescent']
>>> colour.luminous_flux(spd)
3807.655527367202
>>> colour.luminous_efficiency(spd)
0.19943935624521045
>>> colour.luminous_efficiency(spd)
136.21708031547874
```

Colour Models

```
>>> import colour
>>> XYZ = [0.07049534, 0.10080000, 0.09558313]
>>> colour.XYZ_to_Lab(XYZ)
array([ 37.9856291 , -23.62907688,  -4.41746615])
>>> colour.XYZ_to_Luv(XYZ)
array([ 37.9856291 , -28.80219593,  -1.35800706])
>>> colour.XYZ_to_UCS(XYZ)
array([ 0.04699689,  0.1008    ,  0.1637439 ])
>>> colour.XYZ_to_UVW(XYZ)
array([ 4.0680797 ,  0.12787175, -5.36516614])
>>> colour.XYZ_to_xyY(XYZ)
array([ 0.26414772,  0.37770001,  0.1008    ])
>>> colour.XYZ_to_hdr_CIELab(XYZ)
array([ 24.90206646, -46.83127607, -10.14274843])
>>> colour.XYZ_to_hdr_IPT(XYZ)
array([ 25.18261761, -22.62111297,   3.18511729])
>>> colour.XYZ_to_Hunter_Lab([7.049534, 10.080000, 9.558313])
array([ 31.74901573, -15.11462629,  -2.78660758])
>>> colour.XYZ_to_Hunter_Rdab([7.049534, 10.080000, 9.558313])
array([ 10.08      , -18.67653764,  -3.44329925])
>>> colour.XYZ_to_IPT(XYZ)
array([ 0.36571124, -0.11114798,  0.01594746])

>>> 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 = colour.CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = colour.XYZ_to_CIECAM02(
        XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> colour.JMh_CIECAM02_to_CAM02UCS(JMh)
array([ 54.90433134,  -0.08442362,  -0.06848314])
>>> 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([ 54.89102616,  -9.42910274,  -5.52845976])

>>> XYZ = [0.07049534, 0.10080000, 0.09558313]
>>> illuminant_XYZ = [0.34570, 0.35850]
>>> illuminant_RGB = [0.31270, 0.32900]
>>> chromatic_adaptation_transform = 'Bradford'
>>> XYZ_to_RGB_matrix = [
        [3.24062548, -1.53720797, -0.49862860],
        [-0.96893071, 1.87575606, 0.04151752],
        [0.05571012, -0.20402105, 1.05699594]]
>>> colour.XYZ_to_RGB(
        XYZ,
        illuminant_XYZ,
        illuminant_RGB,
        XYZ_to_RGB_matrix,
        chromatic_adaptation_transform)
array([ 0.01100154,  0.12735048,  0.11632713])

>>> colour.RGB_to_ICTCP([0.35181454, 0.26934757, 0.21288023])
array([ 0.09554079, -0.00890639,  0.01389286])

>>> colour.RGB_to_HSV([0.49019608, 0.98039216, 0.25098039])
array([ 0.27867383,  0.744     ,  0.98039216])

>>> 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]])

>>> colour.RGB_to_Prismatic([0.25, 0.50, 0.75])
array([ 0.75      ,  0.16666667,  0.33333333,  0.5       ])

>>> colour.RGB_to_YCbCr([1.0, 1.0, 1.0])
array([ 0.92156863,  0.50196078,  0.50196078])
```

RGB Colourspaces

```
>>> import colour
>>> sorted(colour.RGB_COLOURSPACES.keys())
[u'ACES2065-1',
 u'ACEScc',
 u'ACEScct',
 u'ACEScg',
 u'ACESproxy',
 u'ALEXA Wide Gamut',
 u'Adobe RGB (1998)',
 u'Adobe Wide Gamut RGB',
 u'Apple RGB',
 u'Best RGB',
 u'Beta RGB',
 u'CIE RGB',
 u'Cinema Gamut',
 u'ColorMatch RGB',
 u'DCI-P3',
 u'DCI-P3+',
 u'DRAGONcolor',
```

```
 u'DRAGONcolor2',
 u'Don RGB 4',
 u'ECI RGB v2',
 u'ERIMM RGB',
 u'Ekta Space PS 5',
 u'ITU-R BT.2020',
 u'ITU-R BT.470 - 525',
 u'ITU-R BT.470 - 625',
 u'ITU-R BT.709',
 u'Max RGB',
 u'NTSC',
 u'Pal/Secam',
 u'ProPhoto RGB',
 u'Protune Native',
 u'REDWideGamutRGB',
 u'REDcolor',
 u'REDcolor2',
 u'REDcolor3',
 u'REDcolor4',
 u'RIMM RGB',
 u'ROMM RGB',
 u'Russell RGB',
 u'S-Gamut',
 u'S-Gamut3',
 u'S-Gamut3.Cine',
 u'SMPTE 240M',
 u'V-Gamut',
 u'Xtreme RGB',
 'aces',
 'adobe1998',
 'prophoto',
 u'sRGB']
```

OETFs

```
>>> import colour
>>> sorted(colour.OETFS.keys())
['ARIB STD-B67',
 'DCI-P3',
 'DICOM GSDF',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ITU-R BT.601',
 'ITU-R BT.709',
 'ProPhoto RGB',
 'RIMM RGB',
 'ROMM RGB',
 'SMPTE 240M',
 'ST 2084',
 'sRGB']
```

EOTFs

```
>>> import colour
>>> sorted(colour.EOTFS.keys())
['DCI-P3',
 'DICOM GSDF',
```

```
 'ITU-R BT.1886',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ProPhoto RGB',
 'RIMM RGB',
 'ROMM RGB',
 'SMPTE 240M',
 'ST 2084']
```

OOTFs

```
>>> import colour
>>> sorted(colour.OOTFS.keys())
['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ']
```

Log Encoding / Decoding Curves

```
>>> import colour
>>> sorted(colour.LOG_ENCODING_CURVES.keys())
['ACEScc',
 'ACEScct',
 'ACESproxy',
 'ALEXA Log C',
 'Canon Log',
 'Canon Log 2',
 'Canon Log 3',
 'Cineon',
 'ERIMM RGB',
 'Log3G10',
 'Log3G12',
 'PLog',
 'Panalog',
 'Protune',
 'REDLog',
 'REDLogFilm',
 'S-Log',
 'S-Log2',
 'S-Log3',
 'V-Log',
 'ViperLog']
```

Chromatic Adaptation Models

```
>>> import colour
>>> XYZ = [0.07049534, 0.10080000, 0.09558313]
>>> XYZ_w = [1.09846607, 1.00000000, 0.35582280]
>>> XYZ_wr = [0.95042855, 1.00000000, 1.08890037]
>>> colour.chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr)
array([ 0.08397461,  0.11413219,  0.28625545])
```

Colour Appearance Models

```
>>> import colour
>>> XYZ = [19.01, 20.00, 21.78]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
```

```
>>> colour.XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b)
CIECAM02_Specification(J=41.731091132513917, C=0.10470775717103062, h=219.04843265831178, s=2.
↪3603053739196032, Q=195.37132596607671, M=0.10884217566914849, H=278.06073585667758, HC=None)
```

Colour Difference

```
>>> import colour
>>> Lab_1 = [100.00000000, 21.57210357, 272.22819350]
>>> Lab_2 = [100.00000000, 426.67945353, 72.39590835]
>>> colour.delta_E(Lab_1, Lab_2)
94.035649026659485
>>> sorted(colour.DELTA_E_METHODS.keys())
['CAM02-LCD',
 'CAM02-SCD',
 'CAM02-UCS',
 'CAM16-LCD',
 'CAM16-SCD',
 'CAM16-UCS',
 'CIE 1976',
 'CIE 1994',
 'CIE 2000',
 'CMC',
 'cie1976',
 'cie1994',
 'cie2000']
```

Colour Notation Systems

```
>>> import colour
>>> colour.munsell_value(10.1488096782)
3.7462971142584354
>>> sorted(colour.MUNSELL_VALUE_METHODS.keys())
[u'ASTM D1535-08',
 u'Ladd 1955',
 u'McCamy 1987',
 u'Moon 1943',
 u'Munsell 1933',
 u'Priest 1920',
 u'Saunderson 1944',
 u'astm2008']
>>> colour.xyY_to_munsell_colour([0.38736945, 0.35751656, 0.59362000])
u'4.2YR 8.1/5.3'
>>> colour.munsell_colour_to_xyY('4.2YR 8.1/5.3')
array([ 0.38736945,  0.35751656,  0.59362   ])
```

Optical Phenomena

```
>>> import colour
>>> colour.rayleigh_scattering_spd()
SpectralPowerDistribution([[  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={u'right': None, u'method': u'Constant', u'left': None})
```

Light Quality

```
>>> import colour
>>> colour.colour_quality_scale(colour.ILLUMINANTS_RELATIVE_SPDS['F2'])
64.686416902221609
>>> colour.colour_rendering_index(colour.ILLUMINANTS_RELATIVE_SPDS['F2'])
64.151520202968015
```

Reflectance Recovery

```
>>> import colour
>>> colour.XYZ_to_spectral([0.07049534, 0.10080000, 0.09558313])
SpectralPowerDistribution([[  3.60000000e+02,   7.96361498e-04],
                           [  3.65000000e+02,   7.96489667e-04],
                           [  3.70000000e+02,   7.96543669e-04],
                           ...
                           [  8.20000000e+02,   1.71014294e-04],
                           [  8.25000000e+02,   1.71621924e-04],
                           [  8.30000000e+02,   1.72026883e-04]],
                          interpolator=SpragueInterpolator,
                          interpolator_args={},
                          extrapolator=Extrapolator,
                          extrapolator_args={u'right': None, u'method': u'Constant', u'left': None})
>>> sorted(colour.REFLECTANCE_RECOVERY_METHODS.keys())
['Meng 2015', 'Smits 1999']
```

Correlated Colour Temperature Computation Methods

```
>>> import colour
>>> colour.uv_to_CCT([0.1978, 0.3122])
array([  6.50751282e+03,   3.22335875e-03])
>>> sorted(colour.UV_TO_CCT_METHODS.keys())
[u'Ohno 2013', u'Robertson 1968', u'ohno2013', u'robertson1968']
>>> sorted(colour.UV_TO_CCT_METHODS.keys())
[u'Krystek 1985',
 u'Ohno 2013',
 u'Robertson 1968',
 u'ohno2013',
 u'robertson1968']
 >>> sorted(colour.XY_TO_CCT_METHODS.keys())
 [u'Hernandez 1999', u'McCamy 1992', u'hernandez1999', u'mccamy1992']
 >>> sorted(colour.CCT_TO_XY_METHODS.keys())
 [u'CIE Illuminant D Series', u'Kang 2002', su'cie_d', u'kang2002']
```

Volume

```
>>> import colour
>>> colour.RGB_colourspace_volume_MonteCarlo(colour.sRGB_COLOURSPACE)
857011.5
```

CHAPTER

FOUR

CONTRIBUTING

If you would like to contribute to Colour, please refer to the following Contributing guide.

CHAPTER

FIVE

CHANGES

The changes are viewable on the Releases page.

CHAPTER

SIX

BIBLIOGRAPHY

The bibliography is available on the Bibliography page.

It is also viewable directly from the repository in BibTeX format.

CHAPTER SEVEN

SEE ALSO

Here is a list of notable colour science packages sorted by languages:

Python

- ColorPy by Kness, M.
- Colorspacious by Smith, N. J., et al.
- python-colormath by Taylor, G., et al.

.NET

- Colourful by Pažourek, T., et al.

Julia

- Colors.jl by Holy, T., et al.

Matlab & Octave

- COLORLAB by Malo, J., et al.
- Psychtoolbox by Brainard, D., et al.
- The Munsell and Kubelka-Munk Toolbox by Centore, P.

CHAPTER

EIGHT

ABOUT

Colour by Colour Developers - 2013-2018
Copyright © 2013-2018 – Colour Developers – colour-science@googlegroups.com
This software is released under terms of New BSD License: http://opensource.org/licenses/BSD-3-Clause
http://github.com/colour-science/colour

BIBLIOGRAPHY

[ANS03] ANSI. Specification of ROMM RGB. 2003. URL: http://www.color.org/ROMMRGB.pdf.

[ARR12] ARRI. ALEXA - Log C Curve - Usage in VFX. 2012. URL: http://www.arri.com/?eID=registration&file_uid=8026.

[Bab12a] BabelColor. ColorChecker RGB and spectra. 2012. URL: http://www.babelcolor.com/download/ColorChecker_RGB_and_spectra.xls.

[Bab12b] BabelColor. The ColorChecker (since 1976!). 2012. URL: http://www.babelcolor.com/main_level/ColorChecker.htm.

[BS10] S. Bianco and R. Schettini. Two new von Kries based chromatic adaptation transforms found by numerical optimization. *Color Research & Application*, 35(3):184–192, jun 2010. URL: http://doi.wiley.com/10.1002/col.20573, doi:10.1002/col.20573.

[BWDS99] Barry A. Bodhaine, Norman B. Wood, Ellsworth G. Dutton, and James R. Slusser. On Rayleigh Optical Depth Calculations. *Journal of Atmospheric and Oceanic Technology*, 16(11):1854–1861, nov 1999. URL: http://journals.ametsoc.org/doi/abs/10.1175/1520-0426%281999%29016%3C1854%3AORODC%3E2.0.CO%3B2, doi:10.1175/1520-0426(1999)016<1854:ORODC>2.0.CO;2.

[Bor17] Tim Borer. Private Discussion with Mansencal, T. and Shaw, N. 2017.

[Bou] Paul Bourke. Intersection point of two line segments in 2 dimensions. URL: http://paulbourke.net/geometry/pointlineplane/.

[Bre87] Edwin J Breneman. Corresponding chromaticities for different states of adaptation to complex visual fields. *Journal of the Optical Society of America A*, 4(6):1115, jun 1987. URL: https://www.osapublishing.org/abstract.cfm?URI=josaa-4-6-1115, doi:10.1364/JOSAA.4.001115.

[BS08] Michael H. Brill and Sabine Susstrunk. Repairing gamut problems in CIECAM02: A progress report. *Color Research & Application*, 33(5):424–426, oct 2008. URL: http://doi.wiley.com/10.1002/col.20432, doi:10.1002/col.20432.

[Bro09] A. D. Broadbent. Calculation from the original experimental data of the CIE 1931 RGB standard observer spectral chromaticity co-ordinates and color matching functions. 2009. URL: http://www.cis.rit.edu/mcsl/research/1931.php.

[BB09] Wilhelm Burger and Mark James Burge. *Principles of Digital Image Processing*. Undergraduate Topics in Computer Science. Springer London, London, 2009. ISBN 978-1-84800-194-7. URL: http://link.springer.com/10.1007/978-1-84800-195-4, doi:10.1007/978-1-84800-195-4.

[Can] Canon. EOS C300 Mark II - EOS C300 Mark II Input Transform Version 2.0 (for Cinema Gamut / BT.2020). URL: https://www.usa.canon.com/internet/portal/us/home/support/details/cameras/cinema-eos/eos-c300-mark-ii.

[Can14] Canon. EOS C500 Firmware Update. 2014. URL: https://www.usa.canon.com/internet/portal/us/home/explore/product-showcases/cameras-and-lenses/cinema-eos-firmware/c500.

[Cas14] Saullo Castro. Numpy: Fastest way of computing diagonal for each row of a 2d array. 2014. URL: http://stackoverflow.com/questions/26511401/numpy-fastest-way-of-computing-diagonal-for-each-row-of-a-2d-array/26517247#26517247.

[Cen] Paul Centore. The Munsell and Kubelka-Munk Toolbox. URL: http://www.munsellcolourscienceforpainters.com/MunsellAndKubelkaMunkToolbox/MunsellAndKubelkaMunkToolbox.html.

[Cen12] Paul Centore. An open-source inversion algorithm for the Munsell renotation. *Color Research & Application*, 37(6):455–464, dec 2012. URL: http://doi.wiley.com/10.1002/col.20715, doi:10.1002/col.20715.

[Cen14a] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - GeneralRoutines/CIELABtoApproxMunsellSpec.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14b] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/ChromDiagHueAngleToMunsellHue.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14c] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/FindHueOnRenotationOvoid.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14d] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MaxChromaForExtrapolatedRenotation.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14e] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellHueToASTMHue.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14f] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellHueToChromDiagHueAngle.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14g] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellToxyForIntegerMunsellValue.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14h] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellToxyY.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14i] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/xyYtoMunsell.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14j] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellSystemRoutines/BoundingRenotationHues.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14k] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellSystemRoutines/LinearVsRadialInterpOnRenotationOvoid.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[CIE] CIE. CIE Spectral Data. URL: http://files.cie.co.at/204.xls.

[CIE04] CIE. CIE 15:2004 Tables Data. 2004. URL: https://law.resource.org/pub/us/cfr/ibr/003/cie.15.2004.tables.xls.

[Cot] Russell Cottrell. The Russell RGB working color space. URL: http://www.russellcottrell.com/photo/downloads/RussellRGB.icc.

[CVRa] CVRL. CIE (2012) 10-deg XYZ “physiologically-relevant” colour matching functions. URL: http://www.cvrl.org/database/text/cienewxyz/cie2012xyz10.htm.

[CVRb] CVRL. CIE (2012) 2-deg XYZ “physiologically-relevant” colour matching functions. URL: http://www.cvrl.org/database/text/cienewxyz/cie2012xyz2.htm.

[CVRc] CVRL. Cone Fundamentals. URL: http://www.cvrl.org/cones.htm.

[CVRd] CVRL. Luminous efficiency. URL: http://www.cvrl.org/lumindex.htm.

[CVRe] CVRL. New CIE XYZ functions transformed from the CIE (2006) LMS functions. URL: http://cvrl.ioo.ucl.ac.uk/ciexyzpr.htm.

[CVRf] CVRL. Older CIE Standards. URL: http://cvrl.ioo.ucl.ac.uk/cie.htm.

[CVRg] CVRL. Stiles & Burch individual 10-deg colour matching data. URL: http://www.cvrl.org/stilesburch10_ind.htm.

[CVRh] CVRL. Stiles & Burch individual 2-deg colour matching data. URL: http://www.cvrl.org/stilesburch2_ind.htm.

[DFGM15] Maryam Mohammadzadeh Darrodi, Graham Finlayson, Teresa Goodman, and Michal Mackiewicz. Reference data set for camera spectral sensitivity estimation. *Journal of the Optical Society of America A*, 32(3):381, mar 2015. URL: https://www.osapublishing.org/abstract.cfm?URI=josaa-32-3-381, doi:10.1364/JOSAA.32.000381.

[DO10] Wendy Davis and Yoshiro Ohno. Color quality scale. *Optical Engineering*, 49(3):033602, mar 2010. URL: http://opticalengineering.spiedigitallibrary.org/article.aspx?doi=10.1117/1.3360335, doi:10.1117/1.3360335.

[Dol16] Dolby. WHAT IS ICTCP? - INTRODUCTION. 2016. URL: https://www.dolby.com/us/en/technologies/dolby-vision/ICtCp-white-paper.pdf.

[Easa] EasyRGB. CMY —> CMYK. URL: http://www.easyrgb.com/index.php?X=MATH&H=13#text13.

[Easb] EasyRGB. CMY —> RGB. URL: http://www.easyrgb.com/index.php?X=MATH&H=12#text12.

[Easc] EasyRGB. CMYK —> CMY. URL: http://www.easyrgb.com/index.php?X=MATH&H=14#text14.

[Easd] EasyRGB. HSL —> RGB. URL: http://www.easyrgb.com/index.php?X=MATH&H=19#text19.

[Ease] EasyRGB. HSV —> RGB. URL: http://www.easyrgb.com/index.php?X=MATH&H=21#text21.

[Easf] EasyRGB. RGB —> CMY. URL: http://www.easyrgb.com/index.php?X=MATH&H=11#text11.

[Easg] EasyRGB. RGB —> HSL. URL: http://www.easyrgb.com/index.php?X=MATH&H=18#text18.

[Eash] EasyRGB. RGB —> HSV. URL: http://www.easyrgb.com/index.php?X=MATH&H=20#text20.

[Erda] U. Murat Erdem. Fast Line Segment Intersection. URL: http://www.mathworks.com/matlabcentral/fileexchange/27205-fast-line-segment-intersection.

[Erdb] Turan Erdogan. How to Calculate Luminosity, Dominant Wavelength, and Excitation Purity. URL: http://www.semrock.com/Data/Sites/1/semrockpdfs/whitepaper_howtocalculateluminositywavelengthandpurity.pdf.

[FW98] M. Fairchild and D. Wyble. Colorimetric Characterization of The Apple Studio Display (flat panel LCD). 1998. URL: http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1922&context=article.

[FC11] Mark D Fairchild and Ping-hsu Chen. Brightness, lightness, and specifying color in high-dynamic-range scenes and images. In Susan P. Farnand and Frans Gaykema, editors, *Proc. SPIE 7867, Image Quality and System Performance VIII*, 78670O. jan 2011. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.872075, doi:10.1117/12.872075.

[Fai] Mark D. Fairchild. Fairchild YSh. URL: http://rit-mcsl.org/fairchild//files/FairchildYSh.zip.

[Fai91] Mark D. Fairchild. Formulation and testing of an incomplete-chromatic-adaptation model. *Color Research & Application*, 16(4):243–250, aug 1991. URL: http://doi.wiley.com/10.1002/col.5080160406, doi:10.1002/col.5080160406.

[Fai96] Mark D. Fairchild. Refinement of the RLAB color space. *Color Research & Application*, 21(5):338–346, oct 1996. URL: http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199610%2921%3A5%3C338%3A%3AAID-COL3%3E3.0.CO%3B2-Z, doi:10.1002/(SICI)1520-6378(199610)21:5<338::AID-COL3>3.0.CO;2-Z.

[Fai04] Mark D. Fairchild. CIECAM02. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter CIECAM02, pages 289–301. Wiley, 2 edition, 2004.

[Fai13a] Mark D. Fairchild. ATD Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 14.2, pages 5852–5991. Wiley, 3 edition, 2013.

[Fai13b] Mark D. Fairchild. Chromatic Adaptation Models. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 11, pages 4179–4252. Wiley, 3 edition, 2013.

[Fai13c] Mark D. Fairchild. FAIRCHILD'S 1990 MODEL. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 11, pages 4418–4495. Wiley, 3 edition, 2013.

[Fai13d] Mark D. Fairchild. IPT Colourspace. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 20.3, pages 6197–6223. Wiley, 3 edition, 2013.

[Fai13e] Mark D. Fairchild. LLAB Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 14.3, pages 6025–6178. Wiley, 3 edition, 2013.

[Fai13f] Mark D. Fairchild. The Hunt Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 12, pages 5094–5556. Wiley, 3 edition, 2013.

[Fai13g] Mark D. Fairchild. The Nayatani et al. Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 11, pages 4810–5085. Wiley, 3 edition, 2013.

[Fai13h] Mark D. Fairchild. The RLAB Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 13, pages 5563–5824. Wiley, 3 edition, 2013.

[FW10] Mark D. Fairchild and David R. Wyble. hdr-CIELAB and hdr-IPT: Simple Models for Describing the Color of High-Dynamic-Range and Wide-Color-Gamut Images. In *Proc. of Color and Imaging Conference*, 322–326. 2010. URL: http://www.ingentaconnect.com/content/ist/cic/2010/00002010/00000001/art00057.

[Fai85] Hugh S. Fairman. The calculation of weight factors for tristimulus integration. *Color Research & Application*, 10(4):199–203, 1985. URL: http://doi.wiley.com/10.1002/col.5080100407, doi:10.1002/col.5080100407.

[FBH97] Hugh S. Fairman, Michael H. Brill, and Henry Hemmendinger. How the CIE 1931 color-matching functions were derived from Wright-Guild data. *Color Research & Application*, 22(1):11–23, feb 1997. URL: http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199702%2922%3A1%3C11%3A%3AAID-COL4%3E3.0.CO%3B2-7, doi:10.1002/(SICI)1520-6378(199702)22:1<11::AID-COL4>3.0.CO;2-7.

[GDY+] Hugo Gaggioni, Patel Dhanendra, Jin Yamashita, N. Kawada, K. Endo, and Curtis Clark. S-Log: A new LUT for digital production mastering and interchange applications. URL: http://pro.sony.com/bbsccms/assets/files/mkt/cinema/solutions/slog_manual.pdf.

[GMRS58] L. G. Glasser, A. H. McKinney, C. D. Reilly, and P. D. Schnelle. Cube-Root Color Coordinate System. *Journal of the Optical Society of America*, 48(10):736, oct 1958. URL: https://www.osapublishing.org/abstract.cfm?URI=josa-48-10-736, doi:10.1364/JOSA.48.000736.

[GDM16] GoPro, Haarm-Pieter Duiker, and Thomas Mansencal. Gopro.py. 2016. URL: https://github.com/hpd/OpenColorIO-Configs/blob/master/aces_1.0.3/python/aces_ocio/colorspaces/gopro.py.

[Gut95] S. Lee Guth. Further applications of the ATD model for color vision. In Eric Walowit, editor, *Proc. SPIE 2414, Device-Independent Color Imaging II*, volume 2414, 12–26. apr 1995. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=991324, doi:10.1117/12.206546.

[HernandezAndresLR99] Javier Hernández-Andrés, Raymond L. Lee, and Javier Romero. Calculating correlated color temperatures across the entire gamut of daylight and skylight chromaticities. *Applied Optics*, 38(27):5703, sep 1999. URL: https://www.osapublishing.org/abstract.cfm?URI=ao-38-27-5703, doi:10.1364/AO.38.005703.

[Hol] Joseph Holmes. Ekta Space PS 5. URL: https://www.josephholmes.com/userfiles/Ekta_Space_PS5_JHolmes.zip.

[Hou15] Jim Houston. Private Discussion with Mansencal, T. 2015.

[Hun04] R.W.G. Hunt. *The Reproduction of Colour*. The Wiley-IS&T Series in Imaging Science and Technology. John Wiley & Sons, Ltd, Chichester, UK, 6 edition, sep 2004. ISBN 9780470024270. URL: http://doi.wiley.com/10.1002/0470024275, doi:10.1002/0470024275.

[Hun08a] HunterLab. Hunter L,a,b Color Scale. 2008. URL: http://www.hunterlab.se/wp-content/uploads/2012/11/Hunter-L-a-b.pdf.

[Hun08b] HunterLab. Illuminant Factors in Universal Software and EasyMatch Coatings. 2008. URL: https://support.hunterlab.com/hc/en-us/article_attachments/201437785/an02_02.pdf.

[Hun12] HunterLab. Hunter Rd,a,b Color Scale – History and Application. 2012. URL: https://hunterlabdotcom.files.wordpress.com/2012/07/an-1016-hunter-rd-a-b-color-scale-update-12-07-03.pdf.

[Huta] HutchColor. BestRGB (4 K). URL: http://www.hutchcolor.com/profiles/BestRGB.zip.

[Hutb] HutchColor. DonRGB4 (4 K). URL: http://www.hutchcolor.com/profiles/DonRGB4.zip.

[Hutc] HutchColor. MaxRGB (4 K). URL: http://www.hutchcolor.com/profiles/MaxRGB.zip.

[Hutd] HutchColor. XtremeRGB (4 K). URL: http://www.hutchcolor.com/profiles/XtremeRGB.zip.

[KMH+02] Bongsoon Kang, Ohak Moon, Changhee Hong, Honam Lee, Bonghwan Cho, and Youngsun Kim. Design of advanced color: Temperature control system for HDTV applications. *Journal of the Korean Physical Society*, 41(6):865–871, 2002. URL: http://cat.inist.fr/?aModele=afficheN&cpsidt=14448733.

[KPK11] Paul Kienzle, Nikunj Patel, and James Krycka. refl1d.numpyerrors - Refl1D v0.6.19 documentation. 2011. URL: http://www.reflectometry.org/danse/docs/refl1d/_modules/refl1d/numpyerrors.html.

[Kry85] M Krystek. An algorithm to calculate correlated colour temperature. *Color Research & Application*, 10(1):38–40, 1985. URL: http://doi.wiley.com/10.1002/col.5080100109, doi:10.1002/col.5080100109.

[Lau12] Laurent. Reproducibility of python pseudo-random numbers across systems and versions? 2012. URL: http://stackoverflow.com/questions/8786084/reproducibility-of-python-pseudo-random-numbers-across-systems-and-versions.

[LLW+17] Changjun Li, Zhiqiang Li, Zhifeng Wang, Yang Xu, Ming Ronnier Luo, Guihua Cui, Manuel Melgosa, Michael H Brill, and Michael Pointer. Comprehensive color solutions: CAM16, CAT16, and CAM16-UCS. *Color Research & Application*, 42(6):703–718, dec 2017. URL: http://doi.wiley.com/10.1002/col.22131, doi:10.1002/col.22131.

[LLRH02] Changjun Li, Ming Ronnier Luo, Bryan Rigg, and Robert W. G. Hunt. CMC 2000 chromatic adaptation transform: CMCCAT2000. *Color Research & Application*, 27(1):49–58, feb 2002. URL: http://doi.wiley.com/10.1002/col.10005, doi:10.1002/col.10005.

[LPLMartinezverdu07] Changjun Li, Esther Perales, Ming Ronnier Luo, and Francisco Martínez-verdú. The Problem with CAT02 and Its Correction. 2007. URL: https://pdfs.semanticscholar.org/b5a9/0215ad9a1fb6b01f310b3d64305f7c9feb3a.pdf.

[Lin03a] Bruce Lindbloom. A Continuity Study of the CIE L* Function. 2003. URL: http://brucelindbloom.com/LContinuity.html.

[Lin03b] Bruce Lindbloom. Delta E (CIE 1976). 2003. URL: http://brucelindbloom.com/Eqn_DeltaE_CIE76.html.

[Lin03c] Bruce Lindbloom. XYZ to xyY. 2003. URL: http://www.brucelindbloom.com/Eqn_XYZ_to_xyY.html.

[Lin07] Bruce Lindbloom. Spectral Power Distribution of a CIE D-Illuminant. 2007. URL: http://www.brucelindbloom.com/Eqn_DIlluminant.html.

[Lin09a] Bruce Lindbloom. Chromatic Adaptation. 2009. URL: http://brucelindbloom.com/Eqn_ChromAdapt.html.

[Lin09b] Bruce Lindbloom. Delta E (CIE 2000). 2009. URL: http://brucelindbloom.com/Eqn_DeltaE_CIE2000.html.

[Lin09c] Bruce Lindbloom. Delta E (CMC). 2009. URL: http://brucelindbloom.com/Eqn_DeltaE_CMC.html.

[Lin09d] Bruce Lindbloom. xyY to XYZ. 2009. URL: http://www.brucelindbloom.com/Eqn_xyY_to_XYZ.html.

[Lin11] Bruce Lindbloom. Delta E (CIE 1994). 2011. URL: http://brucelindbloom.com/Eqn_DeltaE_CIE94.html.

[Lin14] Bruce Lindbloom. RGB Working Space Information. 2014. URL: http://www.brucelindbloom.com/WorkingSpaceInfo.html.

[LPY+16] Taoran Lu, Fangjun Pu, Peng Yin, Tao Chen, Walt Husak, Jaclyn Pytlarz, Robin Atkins, Jan Froehlich, and Guan-Ming Su. ITP Colour Space and Its Compression Performance for High Dynamic Range and Wide Colour Gamut Video Distribution. *ZTE Communications*, 14(1):32–38, 2016. URL: http://www.cnki.net/kcms/detail/34.1294.TN.20160205.1903.006.html.

[LCL06] M. Ronnier Luo, Guihua Cui, and Changjun Li. Uniform colour spaces based on CIECAM02 colour appearance model. *Color Research & Application*, 31(4):320–330, aug 2006. URL: http://doi.wiley.com/10.1002/col.20227, doi:10.1002/col.20227.

[LL13] Ming Ronnier Luo and Changjun Li. CIECAM02 and Its Recent Developments. In Christine Fernandez-Maloigne, editor, *Advanced Color Image Processing and Analysis*, pages 19–58. Springer New York, New York, NY, 2013. URL: http://link.springer.com/10.1007/978-1-4419-6190-7, doi:10.1007/978-1-4419-6190-7.

[LLK96] Ming Ronnier Luo, Mei-Chun Lo, and Wen-Guey Kuo. The LLAB (l:c) colour model. *Color Research & Application*, 21(6):412–429, dec 1996. URL: http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199612%2921%3A6%3C412%3A%3AAID-COL4%3E3.0.CO%3B2-Z, doi:10.1002/(SICI)1520-6378(199612)21:6<412::AID-COL4>3.0.CO;2-Z.

[LM96] Ming Ronnier Luo and Ján Morovic. Two Unsolved Issues in Colour Management – Colour Appearance and Gamut Mapping. In *Conference: 5th International Conference on High Technology: Imaging Science and Technology – Evolution & Promise*, 136–147. 1996. URL: http://www.researchgate.net/publication/236348295_Two_Unsolved_Issues_in_Colour_Management__Colour_Appearance_and_Gamut_Mapping.

[Mac35] David L. MacAdam. Maximum Visual Efficiency of Colored Materials. *Journal of the Optical Society of America*, 25(11):361–367, nov 1935. URL: http://www.opticsinfobase.org/abstract.cfm?URI=josa-25-11-361, doi:10.1364/JOSA.25.000361.

[Mac10] Gustavo Mello Machado. A model for simulation of color vision deficiency and a color contrast enhancement technique for dichromats. 2010. URL: http://www.lume.ufrgs.br/handle/10183/26950.

[Mana] Thomas Mansencal. Lookup. URL: https://github.com/KelSolaar/Foundations/blob/develop/foundations/data_structures.py.

[Manb] Thomas Mansencal. Structure. URL: https://github.com/KelSolaar/Foundations/blob/develop/foundations/data_structures.py.

[Man15] Thomas Mansencal. RED Colourspaces Derivation. 2015. URL: http://colour-science.org/posts/red-colourspaces-derivation.

[Mel13] Manuel Melgosa. CIE / ISO new standard: CIEDE2000. 2013. URL: http://www.color.org/events/colorimetry/Melgosa_CIEDE2000_Workshop-July4.pdf.

[MSHD15] Johannes Meng, Florian Simon, Johannes Hanika, and Carsten Dachsbacher. Physically Meaningful Rendering using Tristimulus Colours. *Computer Graphics Forum*, 34(4):31–40, jul 2015. URL: http://doi.wiley.com/10.1111/cgf.12676, doi:10.1111/cgf.12676.

[MDolbyLaboratories14] Scott Miller and Dolby Laboratories. A Perceptual EOTF for Extended Dynamic Range Imagery. 2014. URL: https://www.smpte.org/sites/default/files/2014-05-06-EOTF-Miller-1-2-handout.pdf.

[MFH+02] Nathan Moroney, Mark D. Fairchild, Robert W. G. Hunt, Changjun Li, Ming Ronnier Luo, and Todd Newman. The CIECAM02 Color Appearance Model. *Color and Imaging Conference*, pages 23–27, 2002. URL: http://www.ingentaconnect.com/content/ist/cic/2002/00002002/00000001/art00006.

[Nat16] Graeme Nattress. Private Discussion with Shaw, N. 2016.

[NSY95] Yoshinobu Nayatani, Hiroaki Sobagaki, and Kenjiro Hashimoto Tadashi Yano. Lightness dependency of chroma scales of a nonlinear color-appearance model and its latest formulation. *Color Research & Application*, 20(3):156–167, jun 1995. URL: http://doi.wiley.com/10.1002/col.5080200305, doi:10.1002/col.5080200305.

[NNJ43] Sidney M. Newhall, Dorothy Nickerson, and Deane B. Judd. Final Report of the OSA Subcommittee on the Spacing of the Munsell Colors. *Journal of the Optical Society of America*, 33(7):385, jul 1943. URL: https://www.osapublishing.org/abstract.cfm?URI=josa-33-7-385, doi:10.1364/JOSA.33.000385.

[Ohn14] Yoshiro Ohno. Practical Use and Calculation of CCT and Duv. *LEUKOS*, 10(1):47–55, jan 2014. URL: http://www.tandfonline.com/doi/abs/10.1080/15502724.2014.839020, doi:10.1080/15502724.2014.839020.

[OD08] Yoshiro Ohno and Wendy Davis. NIST CQS simulation 7.4. 2008. URL: https://drive.google.com/file/d/1PsuU6QjUJjCX6tQyCud6ul2Tbs8rYWW9/view?usp=sharing.

[Oht97] N. Ohta. The basis of color reproduction engineering. 1997.

[Pan14] Panasonic. VARICAM V-Log/V-Gamut. 2014. URL: http://pro-av.panasonic.net/en/varicam/common/pdf/VARICAM_V-Log_V-Gamut.pdf.

[Poi80] Michael R. Pointer. Pointer's Gamut Data. 1980. URL: http://www.cis.rit.edu/research/mcsl2/online/PointerData.xls.

[Rei] Kenneth Reitz. CaseInsensitiveDict. URL: https://github.com/kennethreitz/requests/blob/v1.2.3/requests/structures.py#L37.

[Sae] Saeedn. Extend a line segment a specific distance. URL: http://stackoverflow.com/questions/7740507/extend-a-line-segment-a-specific-distance.

[Sas] Sastanin. How to make scipy.interpolate give an extrapolated result beyond the input range? URL: http://stackoverflow.com/a/2745496/931625.

[SWD05] Gaurav Sharma, Wencheng Wu, and Edul N. Dalal. The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1):21–30, feb 2005. URL: http://doi.wiley.com/10.1002/col.20070, doi:10.1002/col.20070.

[SH15] Peter Shirley and David Hart. The prismatic color space for rgb computations. 2015.

[Smi78] Alvy Ray Smith. Color gamut transform pairs. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques - SIGGRAPH '78*, SIGGRAPH '78, 12–19. New York, New York, USA, 1978. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=800248.807361, doi:10.1145/800248.807361.

[Smi99] Brian Smits. An RGB-to-Spectrum Conversion for Reflectances. *Journal of Graphics Tools*, 4(4):11–22, jan 1999. URL: http://www.tandfonline.com/doi/abs/10.1080/10867651.1999.10487511, doi:10.1080/10867651.1999.10487511.

[SWG00] K E Spaulding, G J Woolfe, and E J Giorgianni. Reference Input/Output Medium Metric RGB Color Encodings (RIMM/ROMM RGB). 2000. URL: http://www.photo-lovers.org/pdf/color/romm.pdf.

[Spi15] Nick Spiker. Private Discussion with Mansencal, T. 2015. URL: http://www.invisiblelightimages.com/.

[SS88] E. I. Stearns and R. E. Stearns. An example of a method for correcting radiance data for Bandpass error. *Color Research & Application*, 13(4):257–259, aug 1988. URL: http://doi.wiley.com/10.1002/col.5080130410, doi:10.1002/col.5080130410.

[SBS99] Sabine Susstrunk, Robert Buckley, and Steve Swen. Standard RGB Color Spaces. 1999.

[Tho12] Larry Thorpe. CANON-LOG TRANSFER CHARACTERISTIC. 2012. URL: http://downloads.canon.com/CDLC/Canon-Log_Transfer_Characteristic_6-20-2012.pdf.

[Tri15] Tashi Trieu. Private Discussion with Mansencal, T. 2015.

[WRC12a] Stephen Westland, Caterina Ripamonti, and Vien Cheung. CMCCAT2000. In *Computational Colour Science Using MATLAB*, chapter 6.2.3, pages 83–86. 2 edition, 2012.

[WRC12b] Stephen Westland, Caterina Ripamonti, and Vien Cheung. CMCCAT97. In *Computational Colour Science Using MATLAB*, chapter 6.2.2, pages 80. 2 edition, 2012.

[WRC12c] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Correction for Spectral Bandpass. In *Computational Colour Science Using MATLAB*, chapter 4.4, pages 38. 2 edition, 2012.

[WRC12d] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Extrapolation Methods. In *Computational Colour Science Using MATLAB*, chapter 4.4, pages 38. 2 edition, 2012.

[WRC12e] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Interpolation Methods. In *Computational Colour Science Using MATLAB*, chapter 4.3, pages 29–37. 2 edition, 2012.

[Wika] Wikipedia. Approximation. URL: http://en.wikipedia.org/wiki/Color_temperature#Approximation.

[Wikb] Wikipedia. CAT02. URL: http://en.wikipedia.org/wiki/CIECAM02#CAT02.

[Wikc] Wikipedia. CIE 1931 color space. URL: http://en.wikipedia.org/wiki/CIE_1931_color_space.

[Wikd] Wikipedia. CIE 1960 color space. URL: http://en.wikipedia.org/wiki/CIE_1960_color_space.

[Wike] Wikipedia. CIE 1964 color space. URL: http://en.wikipedia.org/wiki/CIE_1964_color_space.

[Wikf] Wikipedia. CIECAM02. URL: http://en.wikipedia.org/wiki/CIECAM02.

[Wikg] Wikipedia. CIELUV. URL: http://en.wikipedia.org/wiki/CIELUV.

[Wikh] Wikipedia. Color difference. URL: http://en.wikipedia.org/wiki/Color_difference.

[Wiki] Wikipedia. Color temperature. URL: http://en.wikipedia.org/wiki/Color_temperature.

[Wikj] Wikipedia. HSL and HSV. URL: http://en.wikipedia.org/wiki/HSL_and_HSV.

[Wikk] Wikipedia. ISO 31-11. URL: https://en.wikipedia.org/wiki/ISO_31-11.

[Wikl] Wikipedia. Lagrange polynomial - Definition. URL: https://en.wikipedia.org/wiki/Lagrange_polynomial#Definition.

[Wikm] Wikipedia. Lanczos resampling. URL: https://en.wikipedia.org/wiki/Lanczos_resampling.

[Wikn] Wikipedia. Lightness. URL: http://en.wikipedia.org/wiki/Lightness.

[Wiko] Wikipedia. List of common coordinate transformations. URL: http://en.wikipedia.org/wiki/List_of_common_coordinate_transformations.

[Wikp] Wikipedia. Luminance. URL: https://en.wikipedia.org/wiki/Luminance.

[Wikq] Wikipedia. Luminosity function. URL: https://en.wikipedia.org/wiki/Luminosity_function#Details.

[Wikr] Wikipedia. Luminous Efficacy. URL: https://en.wikipedia.org/wiki/Luminous_efficacy.

[Wiks] Wikipedia. Mesopic weighting function. URL: http://en.wikipedia.org/wiki/Mesopic_vision#Mesopic_weighting_function.

[Wikt] Wikipedia. Michaelis–Menten kinetics. URL: https://en.wikipedia.org/wiki/Michaelis\T1\textendash{}Menten_kinetics.

[Wiku] Wikipedia. Rayleigh scattering. URL: http://en.wikipedia.org/wiki/Rayleigh_scattering.

[Wikv] Wikipedia. Relation to CIE XYZ. URL: http://en.wikipedia.org/wiki/CIE_1960_color_space#Relation_to_CIE_XYZ.

[Wikw] Wikipedia. Surfaces. URL: http://en.wikipedia.org/wiki/Gamut#Surfaces.

[Wikx] Wikipedia. The reverse transformation. URL: http://en.wikipedia.org/wiki/CIELUV#The_reverse_transformation.

[Wiky] Wikipedia. White points of standard illuminants. URL: http://en.wikipedia.org/wiki/Standard_illuminant#White_points_of_standard_illuminants.

[Wikz] Wikipedia. Whiteness. URL: http://en.wikipedia.org/wiki/Whiteness.

[Wik{] Wikipedia. Wide-gamut RGB color space. URL: http://en.wikipedia.org/wiki/Wide-gamut_RGB_color_space.

[Wik|] Wikipedia. YCbCr. URL: https://en.wikipedia.org/wiki/YCbCr.

[Wys63] Günter Wyszecki. Proposal for a New Color-Difference Formula. *Journal of the Optical Society of America*, 53(11):1318, nov 1963. URL: https://www.osapublishing.org/abstract.cfm?URI=josa-53-11-1318, doi:10.1364/JOSA.53.001318.

[WS00a] Günther Wyszecki and W. S. Stiles. CIE 1976 (L*u*v*)-Space and Color-Difference Formula. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 167. Wiley, 2000.

[WS00b] Günther Wyszecki and W. S. Stiles. CIE Method of Calculating D-Illuminants. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 145–146. Wiley, 2000.

[WS00c] Günther Wyszecki and W. S. Stiles. DISTRIBUTION TEMPERATURE, COLOR TEMPERATURE, AND CORRELATED COLOR TEMPERATURE. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 224–229. Wiley, 2000.

[WS00d] Günther Wyszecki and W. S. Stiles. Integration Replaced by Summation. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 158–163. Wiley, 2000.

[WS00e] Günther Wyszecki and W. S. Stiles. Standard Photometric Observers. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 256–259,395. Wiley, 2000.

[WS00f] Günther Wyszecki and W. S. Stiles. Table 1(3.11) Isotemperature Lines. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 228. Wiley, 2000.

[WS00g] Günther Wyszecki and W. S. Stiles. Table 1(3.3.3). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 138–139. Wiley, 2000.

[WS00h] Günther Wyszecki and W. S. Stiles. Table I(3.7). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 776–777. Wiley, 2000.

[WS00i] Günther Wyszecki and W. S. Stiles. Table I(6.5.3) Whiteness Formulae (Whiteness Measure Denoted by W). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 837–839. Wiley, 2000.

[WS00j] Günther Wyszecki and W. S. Stiles. Table II(3.7). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 778–779. Wiley, 2000.

[WS00k] Günther Wyszecki and W. S. Stiles. The CIE 1964 Standard Observer. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 141. Wiley, 2000.

[XRP12] X-Rite and Pantone. Color iQC and Color iMatch Color Calculations Guide. 2012. URL: https://www.xrite.com/-/media/xrite/files/apps_engineering_techdocuments/c/09_color_calculations_en.pdf.

[Yor14] Rory Yorke. Python: Change format of np.array or allow tolerance in in1d function. 2014. URL: http://stackoverflow.com/a/23521245/931625.

[AdobeSystems05] Adobe Systems. Adobe RGB (1998) Color Image Encoding. 2005. URL: http://www.adobe.com/digitalimag/pdfs/AdobeRGB1998.pdf.

[AdobeSystems13a] Adobe Systems. Adobe DNG Software Development Kit (SDK) - 1.3.0.0 - dng_sdk_1_3/dng_sdk/source/dng_temperature.cpp::dng_temperature::Set_xy_coord. 2013. URL: https://www.adobe.com/support/downloads/dng/dng_sdk.html.

[AdobeSystems13b] Adobe Systems. Adobe DNG Software Development Kit (SDK) - 1.3.0.0 - dng_sdk_1_3/dng_sdk/source/dng_temperature.cpp::dng_temperature::xy_coord. 2013. URL: https://www.adobe.com/support/downloads/dng/dng_sdk.html.

[AssociationoRIaBusinesses15] Association of Radio Industries and Businesses. Essential Parameter Values for the Extended Image Dynamic Range Television (EIDRTV) System for Programme Production. 2015. URL: https://www.arib.or.jp/english/std_tr/broadcasting/desc/std-b67.html.

[ASTMInternational89] ASTM International. ASTM D1535-89 - Standard Practice for Specifying Color by the Munsell System. 1989. URL: http://www.astm.org/DATABASE.CART/HISTORICAL/D1535-89.htm.

[ASTMInternational08] ASTM International. ASTM D1535-08e1 - Standard Practice for Specifying Color by the Munsell System. 2008. doi:10.1520/D1535-08E01.

[ASTMInternational11] ASTM International. ASTM E2022-11 - Standard Practice for Calculation of Weighting Factors for Tristimulus Integration. 2011. doi:10.1520/E2022-11.

[ASTMInternational15] ASTM International. ASTM E308-15 - Standard Practice for Computing the Colors of Objects by Using the CIE System. 2015. doi:10.1520/E0308-15.

[CIET13294] CIE TC 1-32. CIE 109-1994 A Method of Predicting Corresponding Colours under Different Chromatic and Illuminance Adaptations. 1994. URL: http://div1.cie.co.at/?i_ca_id=551&pubid=34.

[CIET13606] CIE TC 1-36. CIE 170-1:2006 Fundamental Chromaticity Diagram with Physiological Axes - Part 1. 2006. URL: http://div1.cie.co.at/?i_ca_id=551&pubid=48.

[CIET13805a] CIE TC 1-38. 9. INTERPOLATION. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter 9, pages 14–19. 2005. URL: http://div1.cie.co.at/?i_ca_id=551&pubid=47.

[CIET13805b] CIE TC 1-38. 9.2.4 Method of interpolation for uniformly spaced independent variable. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter 9.2.4, pages 1–27. 2005. URL: http://div1.cie.co.at/?i_ca_id=551&pubid=47.

[CIET13805c] CIE TC 1-38. EXTRAPOLATION. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter 10, pages 19–20. 2005. URL: http://div1.cie.co.at/?i_ca_id=551&pubid=47.

[CIET13805d] CIE TC 1-38. Table V. Values of the c-coefficients of Equ.s 6 and 7. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter Table V, pages 19. 2005. URL: http://div1.cie.co.at/?i_ca_id=551&pubid=47.

[CIET14804a] CIE TC 1-48. 3.1 Recommendations concerning standard physical data of illuminants. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 3.1, pages 12–13. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804b] CIE TC 1-48. 9.1 Dominant wavelength and purity. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 9.1, pages 32–33. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804c] CIE TC 1-48. APPENDIX E. INFORMATION ON THE USE OF PLANCK'S EQUATION FOR STANDARD AIR. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter APPENDIX E, pages 77–82. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804d] CIE TC 1-48. *CIE 015:2004 Colorimetry, 3rd Edition*. Commission internationale de l'éclairage, 2004. ISBN 978-3-901-90633-6. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804e] CIE TC 1-48. CIE 1976 uniform chromaticity scale diagram (UCS diagram). In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 8.1, pages 24. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804f] CIE TC 1-48. CIE 1976 uniform colour spaces. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 8.2, pages 24. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804g] CIE TC 1-48. Extrapolation. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 7.2.2.1, pages 24. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804h] CIE TC 1-48. The evaluation of whiteness. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 9.4, pages 24. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[DigitalCInitiatives07] Digital Cinema Initiatives. Digital Cinema System Specification - Version 1.1. 2007. URL: http://www.dcimovies.com/archives/spec_v1_1/DCI_DCinema_System_Spec_v1_1.pdf.

[EuropeanCInitiative02] European Color Initiative. ECI RGB v2. 2002. URL: http://www.eci.org/_media/downloads/icc_profiles_from_eci/ecirgbv20.zip.

[HewlettPDCompany09] Hewlett-Packard Development Company. Understanding the HP DreamColor LP2480zx DCI-P3 Emulation Color Space. 2009. URL: http://www.hp.com/united-states/campaigns/workstations/pdfs/lp2480zx-dci–p3-emulation.pdf.

[IESCCommitteeTM2714WGroup14] IES Computer Committee and TM-27-14 Working Group. IES Standard Format for the Electronic Transfer of Spectral Data Electronic Transfer of Spectral Data. 2014.

[InternationalECommission99] International Electrotechnical Commission. IEC 61966-2-1:1999 - Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB. 1999. URL: https://webstore.iec.ch/publication/6169.

[InternationalTUnion98] International Telecommunication Union. Recommendation ITU-R BT.470-6 - CONVENTIONAL TELEVISION SYSTEMS. 1998. URL: http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.470-6-199811-S!!PDF-E.pdf.

[InternationalTUnion11a] International Telecommunication Union. Recommendation ITU-R BT.1886 - Reference electro-optical transfer function for flat panel displays used in HDTV studio production BT Series Broadcasting service. 2011. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1886-0-201103-I!!PDF-E.pdf.

[InternationalTUnion11b] International Telecommunication Union. Recommendation ITU-R BT.601-7 - Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. 2011. URL: http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf.

[InternationalTUnion11c] International Telecommunication Union. Recommendation ITU-T T.871 - Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF). 2011. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.871-201105-I!!PDF-E&type=items.

[InternationalTUnion15a] International Telecommunication Union. Recommendation ITU-R BT.2020 - Parameter values for ultra-high definition television systems for production and international programme exchange. 2015. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.2020-2-201510-I!!PDF-E.pdf.

[InternationalTUnion15b] International Telecommunication Union. Recommendation ITU-R BT.709-6 - Parameter values for the HDTV standards for production and international programme exchange BT Series Broadcasting service. 2015. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.709-6-201506-I!!PDF-E.pdf.

[InternationalTUnion16] International Telecommunication Union. Recommendation ITU-R BT.2100-1 - Image parameter values for high dynamic range television for use in production and international programme exchange. 2016. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.2100-1-201706-I!!PDF-E.pdf.

[MunsellCSciencea] Munsell Color Science. Macbeth Colorchecker. URL: http://www.rit-mcsl.org/UsefulData/MacbethColorChecker.xls.

[MunsellCScienceb] Munsell Color Science. Munsell Colours Data. URL: http://www.cis.rit.edu/research/mcsl2/online/munsell.php.

[NationalEMAssociation04] National Electrical Manufacturers Association. Digital Imaging and Communications in Medicine (DICOM) Part 14: Grayscale Standard Display Function. 2004. URL: http://medical.nema.org/dicom/2004/04_14PU.PDF.

[RenewableRDCenter03] Renewable Resource Data Center. Reference Solar Spectral Irradiance: ASTM G-173. 2003. URL: http://rredc.nrel.gov/solar/spectra/am1.5/ASTMG173/ASTMG173.html.

[SocietyoMPaTEngineers93] Society of Motion Picture and Television Engineers. *RP 177:1993 : Derivation of Basic Television Color Equations*. Volume RP 177:199. The Society of Motion Picture and Television Engineers, jan 1993. ISBN 978-1-61482-191-5. URL: http://standards.smpte.org/lookup/doi/10.5594/S9781614821915, doi:10.5594/S9781614821915.

[SocietyoMPaTEngineers99] Society of Motion Picture and Television Engineers. ANSI/SMPTE 240M-1995 - Signal Parameters - 1125-Line High-Definition Production Systems. 1999. URL: http://car.france3.mars.free.fr/HD/INA-26jan06/SMPTEnormesetconfs/s240m.pdf.

[SocietyoMPaTEngineers04] Society of Motion Picture and Television Engineers. *RP 145:2004: SMPTE C Color Monitor Colorimetry*. Volume RP 145:200. The Society of Motion Picture and Television Engineers, jan 2004. ISBN 978-1-61482-164-9. URL: http://standards.smpte.org/lookup/doi/10.5594/S9781614821649, doi:10.5594/S9781614821649.

[SocietyoMPaTEngineers14] Society of Motion Picture and Television Engineers. SMPTE ST 2084:2014 - Dynamic Range Electro-Optical Transfer Function of Mastering Reference Displays. 2014. URL: http://www.techstreet.com/products/1883436, doi:10.5594/SMPTE.ST2084.2014.

[SonyCorporationa] Sony Corporation. S-Gamut3_S-Gamut3Cine_Matrix.xlsx. URL: https://community.sony.com/sony/attachments/sony/large-sensor-camera-F5-F55/12359/3/S-Gamut3_S-Gamut3Cine_Matrix.xlsx.

[SonyCorporationb] Sony Corporation. S-Log Whitepaper. URL: http://www.theodoropoulos.info/attachments/076_onS-Log.pdf.

[SonyCorporationc] Sony Corporation. Technical Summary for S-Gamut3.Cine/S-Log3 and S-Gamut3/S-Log3. URL: http://community.sony.com/sony/attachments/sony/large-sensor-camera-F5-F55/12359/2/TechnicalSummary_for_S-Gamut3Cine_S-Gamut3_S-Log3_V1_00.pdf.

[SonyCorporation12] Sony Corporation. S-Log2 Technical Paper. 2012. URL: https://pro.sony.com/bbsccms/assets/files/micro/dmpc/training/S-Log2_Technical_PaperV1_0.pdf.

[SonyImageworks12] Sony Imageworks. Make.py. 2012. URL: https://github.com/imageworks/OpenColorIO-Configs/blob/master/nuke-default/make.py.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Academy Color Encoding System. URL: http://www.oscars.org/science-technology/council/projects/aces.html.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2013-001 - ACESproxy , an Integer Log Encoding of ACES Image Data. 2014. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2014-003 - ACEScc , A Logarithmic Encoding of ACES Data for use within Color Grading Systems. 2014. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Technical Bulletin TB-2014-004 - Informative Notes on SMPTE ST 2065-1 – Academy Color Encoding Specification (ACES). 2014. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Technical Bulletin TB-2014-012 - Academy Color Encoding System Version 1.0 Component Names. 2014. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee15] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2014-004 - ACEScg – A Working Space for CGI Render and Compositing. 2015. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project. Specification S-2016-001 - ACEScct, A Quasi-Logarithmic Encoding of ACES Data for use within Color Grading Systems. 2016. URL: https://github.com/ampas/aces-dev/tree/v1.0.3/documents.

INDEX

Symbols

A

B

C

D

E

F

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z