Colour Documentation

Release 0.4.3

Colour Developers

Aug 26, 2023

CONTENTS

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

It is freely available under the BSD-3-Clause terms.

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

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

CHAPTER

ONE

2 SPONSORS

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

CHAPTER

TWO

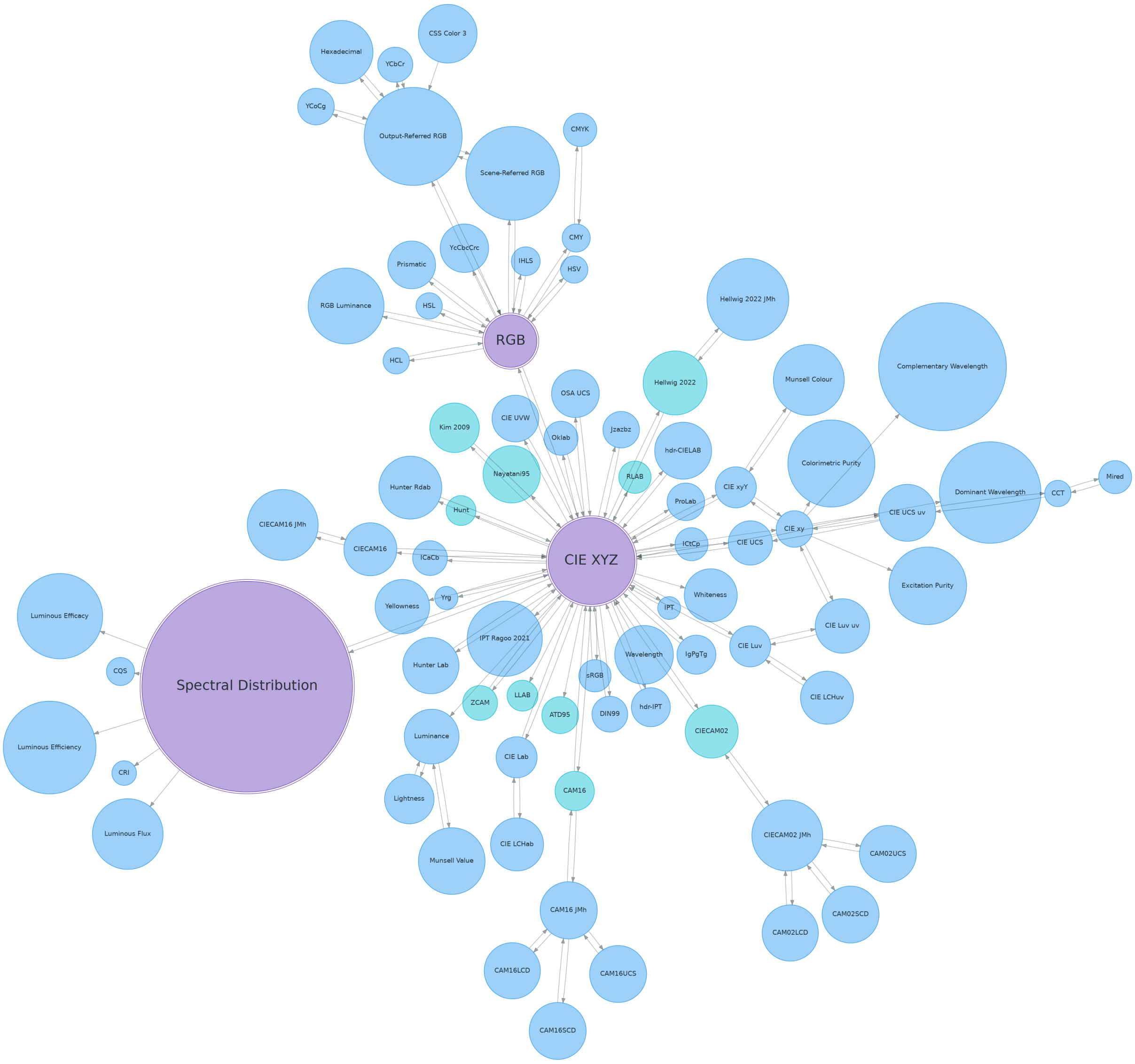
3 FEATURES

Most of the objects are available from the colour namespace:

```
>>> import colour
```

2.1 3.1 Automatic Colour Conversion Graph - colour.graph

Starting with version *0.3.14*, **Colour** implements an automatic colour conversion graph enabling easier colour conversions.




```
>>> sd = colour.SDS_COLOURCHECKERS["ColorChecker N Ohta"]["dark skin"]
>>> colour.convert(
...     sd, "Spectral Distribution", "sRGB", verbose={"mode": "Short"}
... )
```

```
===============================================================================
*                                                                             *
*   [ Conversion Path ]                                                       *
*                                                                             *
*   "sd_to_XYZ" --> "XYZ_to_sRGB"                                             *
*                                                                             *
===============================================================================
array([ 0.45675795,  0.30986982,  0.24861924])
```

```
>>> illuminant = colour.SDS_ILLUMINANTS["FL2"]
>>> colour.convert(
...     sd,
...     "Spectral Distribution",
...     "sRGB",
...     sd_to_XYZ={"illuminant": illuminant},
... )
```

(continues on next page)

(continued from previous page)

```
array([ 0.47924575,  0.31676968,  0.17362725])
```

2.2 3.2 Chromatic Adaptation - colour.adaptation

```
>>> XYZ = [0.20654008, 0.12197225, 0.05136952]
>>> D65 = colour.CCS_ILLUMINANTS["CIE 1931 2 Degree Standard Observer"][
...     "D65"
... ]
>>> A = colour.CCS_ILLUMINANTS["CIE 1931 2 Degree Standard Observer"]["A"]
>>> colour.chromatic_adaptation(
...     XYZ, colour.xy_to_XYZ(D65), colour.xy_to_XYZ(A)
... )
array([ 0.2533053 ,  0.13765138,  0.01543307])
>>> sorted(colour.CHROMATIC_ADAPTATION_METHODS)
['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Von Kries', 'Zhai 2018']
```

2.3 3.3 Algebra - colour.algebra

2.3.1 3.3.1 Kernel Interpolation

```
>>> y = [5.9200, 9.3700, 10.8135, 4.5100, 69.5900, 27.8007, 86.0500]
>>> x = range(len(y))
>>> colour.KernelInterpolator(x, y)([0.25, 0.75, 5.50])
array([  6.18062083,   8.08238488,  57.85783403])
```

2.3.2 3.3.2 Sprague (1880) Interpolation

```
>>> y = [5.9200, 9.3700, 10.8135, 4.5100, 69.5900, 27.8007, 86.0500]
>>> x = range(len(y))
>>> colour.SpragueInterpolator(x, y)([0.25, 0.75, 5.50])
array([  6.72951612,   7.81406251,  43.77379185])
```

2.4 3.4 Colour Appearance Models - colour.appearance

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
>>> colour.XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b)
CAM_Specification_CIECAM02(J=34.434525727858997, C=67.365010921125943, h=22.
↪279164147957065, s=62.81485585332716, Q=177.47124941102123, M=70.024939419291414, H=2.
↪6896085344238898, HC=None)
>>> colour.XYZ_to_CIECAM16(XYZ, XYZ_w, L_A, Y_b)
CAM_Specification_CIECAM16(J=34.434525727858997, C=67.365010921125943, h=22.
↪279164147957065, s=62.81485585332716, Q=177.47124941102123, M=70.024939419291414, H=2.
↪6896085344238898, HC=None)
>>> colour.XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b)
```

(continues on next page)

(continued from previous page)

```
CAM_Specification_CAM16(J=33.880368498111686, C=69.444353357408033, h=19.510887327451748,␣
↪s=64.03612114840314, Q=176.03752758512178, M=72.18638534116765, H=399.52975599115319,␣
↪HC=None)
>>> colour.XYZ_to_Hellwig2022(XYZ, XYZ_w, L_A)
CAM_Specification_Hellwig2022(J=33.880368498111686, C=40.347043294550311, h=19.
↪510887327451748, s=117.38555017188679, Q=45.34489577734751, M=53.228355383108031, H=399.
↪52975599115319, HC=None)
>>> colour.XYZ_to_Kim2009(XYZ, XYZ_w, L_A)
CAM_Specification_Kim2009(J=19.879918542450902, C=55.839055250876946, h=22.
↪013388165090046, s=112.97979354939129, Q=36.309026130161449, M=46.346415858227864, H=2.
↪3543198369639931, HC=None)
>>> colour.XYZ_to_ZCAM(XYZ, XYZ_w, L_A, Y_b)
CAM_Specification_ZCAM(J=38.347186278956357, C=21.12138989208518, h=33.711578931095197,␣
↪s=81.444585609489536, Q=76.986725284523772, M=42.403805833900506, H=0.45779200212219573,
↪ HC=None, V=43.623590687423544, K=43.20894953152817, W=34.829588380192149)
```

2.5 3.5 Colour Blindness - colour.blindness

```
>>> import numpy as np
>>> cmfs = colour.LMS_CMFS["Stockman & Sharpe 2 Degree Cone Fundamentals"]
>>> colour.msds_cmfs_anomalous_trichromacy_Machado2009(
...     cmfs, np.array([15, 0, 0])
... )[450]
array([ 0.08912884,  0.0870524 ,  0.955393  ])
>>> primaries = colour.MSDS_DISPLAY_PRIMARIES["Apple Studio Display"]
>>> d_LMS = (15, 0, 0)
>>> colour.matrix_anomalous_trichromacy_Machado2009(cmfs, primaries, d_LMS)
array([[-0.27774652,  2.65150084, -1.37375432],
       [ 0.27189369,  0.20047862,  0.52762768],
       [ 0.00644047,  0.25921579,  0.73434374]])
```

2.6 3.6 Colour Correction - colour characterisation

```
>>> import numpy as np
>>> RGB = [0.17224810, 0.09170660, 0.06416938]
>>> M_T = np.random.random((24, 3))
>>> M_R = M_T + (np.random.random((24, 3)) - 0.5) * 0.5
>>> colour.colour_correction(RGB, M_T, M_R)
array([ 0.1806237 ,  0.07234791,  0.07848845])
>>> sorted(colour.COLOUR_CORRECTION_METHODS)
['Cheung 2004', 'Finlayson 2015', 'Vandermonde']
```

2.7 3.7 ACES Input Transform - colour characterisation

```
>>> sensitivities = colour.MSDS_CAMERA_SENSITIVITIES["Nikon 5100 (NPL)"]
>>> illuminant = colour.SDS_ILLUMINANTS["D55"]
>>> colour.matrix_idt(sensitivities, illuminant)
(array([[ 0.59368175,  0.30418371,  0.10213454],
       [ 0.00457979,  1.14946003, -0.15403982],
       [ 0.03552213, -0.16312291,  1.12760077]]), array([ 1.58214188,  1.        ,  1.
↪28910346]))
```

2.8 3.8 Colorimetry - colour.colorimetry

2.8.1 3.8.1 Spectral Computations

```
>>> colour.sd_to_XYZ(colour.SDS_LIGHT_SOURCES["Neodimium Incandescent"])
array([ 36.94726204,  32.62076174,  13.0143849 ])
>>> sorted(colour.SPECTRAL_TO_XYZ_METHODS)
['ASTM E308', 'Integration', 'astm2015']
```

2.8.2 3.8.2 Multi-Spectral Computations

```
>>> msds = np.array(
...     [
...         [
...             [
...                 0.01367208,
...                 0.09127947,
...                 0.01524376,
...                 0.02810712,
...                 0.19176012,
...                 0.04299992,
...             ],
...             [
...                 0.00959792,
...                 0.25822842,
...                 0.41388571,
...                 0.22275120,
...                 0.00407416,
...                 0.37439537,
...             ],
...             [
...                 0.01791409,
...                 0.29707789,
...                 0.56295109,
...                 0.23752193,
...                 0.00236515,
...                 0.58190280,
...             ],
...         ],
...         [
...             [
...                 0.01492332,
```

(continues on next page)

(continued from previous page)

```
...                 0.10421912,
...                 0.02240025,
...                 0.03735409,
...                 0.57663846,
...                 0.32416266,
...             ],
...             [
...                 0.04180972,
...                 0.26402685,
...                 0.03572137,
...                 0.00413520,
...                 0.41808194,
...                 0.24696727,
...             ],
...             [
...                 0.00628672,
...                 0.11454948,
...                 0.02198825,
...                 0.39906919,
...                 0.63640803,
...                 0.01139849,
...             ],
...         ],
...         [
...             [
...                 0.04325933,
...                 0.26825359,
...                 0.23732357,
...                 0.05175860,
...                 0.01181048,
...                 0.08233768,
...             ],
...             [
...                 0.02484169,
...                 0.12027161,
...                 0.00541695,
...                 0.00654612,
...                 0.18603799,
...                 0.36247808,
...             ],
...             [
...                 0.03102159,
...                 0.16815442,
...                 0.37186235,
...                 0.08610666,
...                 0.00413520,
...                 0.78492409,
...             ],
...         ],
...         [
...             [
...                 0.11682307,
...                 0.78883040,
...                 0.74468607,
...                 0.83375293,
...                 0.90571451,
```

(continues on next page)

(continued from previous page)

```
...                 0.70054168,
...             ],
...             [
...                 0.06321812,
...                 0.41898224,
...                 0.15190357,
...                 0.24591440,
...                 0.55301750,
...                 0.00657664,
...             ],
...             [
...                 0.00305180,
...                 0.11288624,
...                 0.11357290,
...                 0.12924391,
...                 0.00195315,
...                 0.21771573,
...             ],
...         ],
...     ]
... )
>>> colour.msds_to_XYZ(
...     msds,
...     method="Integration",
...     shape=colour.SpectralShape(400, 700, 60),
... )
array([[[  7.68544647,   4.09414317,   8.49324254],
        [ 17.12567298,  27.77681821,  25.52573685],
        [ 19.10280411,  34.45851476,  29.76319628]],
       [[ 18.03375827,   8.62340812,   9.71702574],
        [ 15.03110867,   6.54001068,  24.53208465],
        [ 37.68269495,  26.4411103 ,  10.66361816]],
       [[  8.09532373,  12.75333339,  25.79613956],
        [  7.09620297,   2.79257389,  11.15039854],
        [  8.933163  ,  19.39985815,  17.14915636]],
       [[ 80.00969553,  80.39810464,  76.08184429],
        [ 33.27611427,  24.38947838,  39.34919287],
        [  8.89425686,  11.05185138,  10.86767594]]])
>>> sorted(colour.MSDS_TO_XYZ_METHODS)
['ASTM E308', 'Integration', 'astm2015']
```

2.8.3 3.8.3 Blackbody Spectral Radiance Computation

```
>>> colour.sd_blackbody(5000)
SpectralDistribution([[  3.60000000e+02,   6.65427827e+12],
                      [  3.61000000e+02,   6.70960528e+12],
                      [  3.62000000e+02,   6.76482512e+12],
                      ...
                      [  7.78000000e+02,   1.06068004e+13],
                      [  7.79000000e+02,   1.05903327e+13],
                      [  7.80000000e+02,   1.05738520e+13]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
```

(continues on next page)

(continued from previous page)

```
                      extrapolator_args={'right': None, 'method': 'Constant', 'left': None}
↪)
```

2.8.4 3.8.4 Dominant, Complementary Wavelength & Colour Purity Computation

```
>>> xy = [0.54369557, 0.32107944]
>>> xy_n = [0.31270000, 0.32900000]
>>> colour.dominant_wavelength(xy, xy_n)
(array(616.0),
 array([ 0.68354746,  0.31628409]),
 array([ 0.68354746,  0.31628409]))
```

2.8.5 3.8.5 Lightness Computation

```
>>> colour.lightness(12.19722535)
41.527875844653451
>>> sorted(colour.LIGHTNESS_METHODS)
['Abebe 2017',
 'CIE 1976',
 'Fairchild 2010',
 'Fairchild 2011',
 'Glasser 1958',
 'Lstar1976',
 'Wyszecki 1963']
```

2.8.6 3.8.6 Luminance Computation

```
>>> colour.luminance(41.52787585)
12.197225353400775
>>> sorted(colour.LUMINANCE_METHODS)
['ASTM D1535',
 'CIE 1976',
 'Fairchild 2010',
 'Fairchild 2011',
 'Newhall 1943',
 'astm2008',
 'cie1976']
```

2.8.7 3.8.7 Whiteness Computation

```
>>> XYZ = [95.00000000, 100.00000000, 105.00000000]
>>> XYZ_0 = [94.80966767, 100.00000000, 107.30513595]
>>> colour.whiteness(XYZ, XYZ_0)
array([ 93.756     ,  -1.33000001])
>>> sorted(colour.WHITENESS_METHODS)
['ASTM E313',
 'Berger 1959',
 'CIE 2004',
 'Ganz 1979',
```

(continues on next page)

(continued from previous page)

```
 'Stensby 1968',
 'Taube 1960',
 'cie2004']
```

2.8.8 3.8.8 Yellowness Computation

```
>>> XYZ = [95.00000000, 100.00000000, 105.00000000]
>>> colour.yellowness(XYZ)
4.3400000000000034
>>> sorted(colour.YELLOWNESS_METHODS)
['ASTM D1925', 'ASTM E313', 'ASTM E313 Alternative']
```

2.8.9 3.8.9 Luminous Flux, Efficiency & Efficacy Computation

```
>>> sd = colour.SDS_LIGHT_SOURCES["Neodimium Incandescent"]
>>> colour.luminous_flux(sd)
23807.655527367202
>>> sd = colour.SDS_LIGHT_SOURCES["Neodimium Incandescent"]
>>> colour.luminous_efficiency(sd)
0.19943935624521045
>>> sd = colour.SDS_LIGHT_SOURCES["Neodimium Incandescent"]
>>> colour.luminous_efficacy(sd)
136.21708031547874
```

2.9 3.9 Contrast Sensitivity Function - colour.contrast

```
>>> colour.contrast_sensitivity_function(u=4, X_0=60, E=65)
358.51180789884984
>>> sorted(colour.CONTRAST_SENSITIVITY_METHODS)
['Barten 1999']
```

2.10 3.10 Colour Difference - colour.difference

```
>>> Lab_1 = [100.00000000, 21.57210357, 272.22819350]
>>> Lab_2 = [100.00000000, 426.67945353, 72.39590835]
>>> colour.delta_E(Lab_1, Lab_2)
94.035649026659485
>>> sorted(colour.DELTA_E_METHODS)
['CAM02-LCD',
 'CAM02-SCD',
 'CAM02-UCS',
 'CAM16-LCD',
 'CAM16-SCD',
 'CAM16-UCS',
 'CIE 1976',
 'CIE 1994',
 'CIE 2000',
 'CMC',
```

(continues on next page)

(continued from previous page)

```
 'DIN99',
 'ITP',
 'cie1976',
 'cie1994',
 'cie2000']
```

2.11 3.11 IO - colour.io

2.11.1 3.11.1 Images

```
>>> RGB = colour.read_image("Ishihara_Colour_Blindness_Test_Plate_3.png")
>>> RGB.shape
(276, 281, 3)
```

2.11.2 3.11.2 Look Up Table (LUT) Data

```
>>> LUT = colour.read_LUT("ACES_Proxy_10_to_ACES.cube")
>>> print(LUT)
```

```
LUT3x1D - ACES Proxy 10 to ACES
-------------------------------
Dimensions : 2
Domain     : [[0 0 0]
              [1 1 1]]
Size       : (32, 3)
```

```
>>> RGB = [0.17224810, 0.09170660, 0.06416938]
>>> LUT.apply(RGB)
array([ 0.00575674,  0.00181493,  0.00121419])
```

2.12 3.12 Colour Models - colour.models

2.12.1 3.12.1 CIE xyY Colourspace

```
>>> colour.XYZ_to_xyY([0.20654008, 0.12197225, 0.05136952])
array([ 0.54369557,  0.32107944,  0.12197225])
```

2.12.2 3.12.2 CIE L*a*b* Colourspace

```
>>> colour.XYZ_to_Lab([0.20654008, 0.12197225, 0.05136952])
array([ 41.52787529,  52.63858304,  26.92317922])
```

2.12.3 3.12.3 CIE L*u*v* Colourspace

```
>>> colour.XYZ_to_Luv([0.20654008, 0.12197225, 0.05136952])
array([ 41.52787529,  96.83626054,  17.75210149])
```

2.12.4 3.12.4 CIE 1960 UCS Colourspace

```
>>> colour.XYZ_to_UCS([0.20654008, 0.12197225, 0.05136952])
array([ 0.13769339,  0.12197225,  0.1053731 ])
```

2.12.5 3.12.5 CIE 1964 U*V*W* Colourspace

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> colour.XYZ_to_UVW(XYZ)
array([ 94.55035725,  11.55536523,  40.54757405])
```

2.12.6 3.12.6 CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006)

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = colour.VIEWING_CONDITIONS_CIECAM02["Average"]
>>> specification = colour.XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> colour.JMh_CIECAM02_to_CAM02UCS(JMh)
array([ 47.16899898,  38.72623785,  15.8663383 ])
>>> XYZ = [0.20654008, 0.12197225, 0.05136952]
>>> XYZ_w = [95.05 / 100, 100.00 / 100, 108.88 / 100]
>>> colour.XYZ_to_CAM02UCS(XYZ, XYZ_w=XYZ_w, L_A=L_A, Y_b=Y_b)
array([ 47.16899898,  38.72623785,  15.8663383 ])
```

2.12.7 3.12.7 CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017)

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = colour.VIEWING_CONDITIONS_CAM16["Average"]
>>> specification = colour.XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> colour.JMh_CAM16_to_CAM16UCS(JMh)
array([ 46.55542238,  40.22460974,  14.25288392]
>>> XYZ = [0.20654008, 0.12197225, 0.05136952]
>>> XYZ_w = [95.05 / 100, 100.00 / 100, 108.88 / 100]
>>> colour.XYZ_to_CAM16UCS(XYZ, XYZ_w=XYZ_w, L_A=L_A, Y_b=Y_b)
array([ 46.55542238,  40.22460974,  14.25288392])
```

2.12.8 3.12.8 DIN99 Colourspace and DIN99b, DIN99c, DIN99d Refined Formulas

```
>>> Lab = [41.52787529, 52.63858304, 26.92317922]
>>> colour.Lab_to_DIN99(Lab)
array([ 53.22821988,  28.41634656,   3.89839552])
```

2.12.9 3.12.9 ICaCb Colourspace

```
>>> XYZ_to_ICaCb(np.array([0.20654008, 0.12197225, 0.05136952]))
array([ 0.06875297,  0.05753352,  0.02081548])
```

2.12.10 3.12.10 IgPgTg Colourspace

```
>>> colour.XYZ_to_IgPgTg([0.20654008, 0.12197225, 0.05136952])
array([ 0.42421258,  0.18632491,  0.10689223])
```

2.12.11 3.12.11 IPT Colourspace

```
>>> colour.XYZ_to_IPT([0.20654008, 0.12197225, 0.05136952])
array([ 0.38426191,  0.38487306,  0.18886838])
```

2.12.12 3.12.12 Jzazbz Colourspace

```
>>> colour.XYZ_to_Jzazbz([0.20654008, 0.12197225, 0.05136952])
array([ 0.00535048,  0.00924302,  0.00526007])
```

2.12.13 3.12.13 hdr-CIELAB Colourspace

```
>>> colour.XYZ_to_hdr_CIELab([0.20654008, 0.12197225, 0.05136952])
array([ 51.87002062,  60.4763385 ,  32.14551912])
```

2.12.14 3.12.14 hdr-IPT Colourspace

```
>>> colour.XYZ_to_hdr_IPT([0.20654008, 0.12197225, 0.05136952])
array([ 25.18261761, -22.62111297,   3.18511729])
```

2.12.15 3.12.15 Hunter L,a,b Colour Scale

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> colour.XYZ_to_Hunter_Lab(XYZ)
array([ 34.92452577,  47.06189858,  14.38615107])
```

2.12.16 3.12.16 Hunter Rd,a,b Colour Scale

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> colour.XYZ_to_Hunter_Rdab(XYZ)
array([ 12.197225  ,  57.12537874,  17.46241341])
```

2.12.17 3.12.17 Oklab Colourspace

```
>>> colour.XYZ_to_Oklab([0.20654008, 0.12197225, 0.05136952])
array([ 0.51634019,  0.154695  ,  0.06289579])
```

2.12.18 3.12.18 OSA UCS Colourspace

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> colour.XYZ_to_OSA_UCS(XYZ)
array([-3.0049979 ,  2.99713697, -9.66784231])
```

2.12.19 3.12.19 ProLab Colourspace

```
>>> colour.XYZ_to_ProLab([0.51634019, 0.15469500, 0.06289579])
array([1.24610688, 2.39525236, 0.41902126])
```

2.12.20 3.12.20 Ragoo and Farup (2021) Optimised IPT Colourspace

```
>>> colour.XYZ_to_IPT_Ragoo2021([0.20654008, 0.12197225, 0.05136952])
array([ 0.42248243,  0.2910514 ,  0.20410663])
```

2.12.21 3.12.21 Yrg Colourspace - Kirk (2019)

```
>>> colour.XYZ_to_Yrg([0.20654008, 0.12197225, 0.05136952])
array([ 0.13137801,  0.49037645,  0.37777388])
```

2.12.22 3.12.22 Y’CbCr Colour Encoding

```
>>> colour.RGB_to_YCbCr([1.0, 1.0, 1.0])
array([ 0.92156863,  0.50196078,  0.50196078])
```

2.12.23 3.12.23 YCoCg Colour Encoding

```
>>> colour.RGB_to_YCoCg([0.75, 0.75, 0.0])
array([ 0.5625,  0.375 ,  0.1875])
```

2.12.24 3.12.24 ICtCp Colour Encoding

```
>>> colour.RGB_to_ICtCp([0.45620519, 0.03081071, 0.04091952])
array([ 0.07351364,  0.00475253,  0.09351596])
```

2.12.25 3.12.25 HSV Colourspace

```
>>> colour.RGB_to_HSV([0.45620519, 0.03081071, 0.04091952])
array([ 0.99603944,  0.93246304,  0.45620519])
```

2.12.26 3.12.26 IHLS Colourspace

```
>>> colour.RGB_to_IHLS([0.45620519, 0.03081071, 0.04091952])
array([ 6.26236117,  0.12197943,  0.42539448])
```

2.12.27 3.12.27 Prismatic Colourspace

```
>>> colour.RGB_to_Prismatic([0.25, 0.50, 0.75])
array([ 0.75      ,  0.16666667,  0.33333333,  0.5       ])
```

2.12.28 3.12.28 RGB Colourspace and Transformations

```
>>> XYZ = [0.21638819, 0.12570000, 0.03847493]
>>> illuminant_XYZ = [0.34570, 0.35850]
>>> illuminant_RGB = [0.31270, 0.32900]
>>> chromatic_adaptation_transform = "Bradford"
>>> matrix_XYZ_to_RGB = [
...     [3.24062548, -1.53720797, -0.49862860],
...     [-0.96893071, 1.87575606, 0.04151752],
...     [0.05571012, -0.20402105, 1.05699594],
... ]
>>> colour.XYZ_to_RGB(
...     XYZ,
...     illuminant_XYZ,
...     illuminant_RGB,
...     matrix_XYZ_to_RGB,
...     chromatic_adaptation_transform,
... )
array([ 0.45595571,  0.03039702,  0.04087245])
```

2.12.29 3.12.29 RGB Colourspace Derivation

```
>>> p = [0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700]
>>> w = [0.32168, 0.33767]
>>> colour.normalised_primary_matrix(p, w)
array([[  9.52552396e-01,   0.00000000e+00,   9.36786317e-05],
       [  3.43966450e-01,   7.28166097e-01,  -7.21325464e-02],
       [  0.00000000e+00,   0.00000000e+00,   1.00882518e+00]])
```

2.12.30 3.12.30 RGB Colourspaces

```
>>> sorted(colour.RGB_COLOURSPACES)
['ACES2065-1',
 'ACEScc',
 'ACEScct',
 'ACEScg',
 'ACESproxy',
 'ARRI Wide Gamut 3',
 'ARRI Wide Gamut 4',
 'Adobe RGB (1998)',
 'Adobe Wide Gamut RGB',
 'Apple RGB',
 'Best RGB',
 'Beta RGB',
 'Blackmagic Wide Gamut',
 'CIE RGB',
 'Cinema Gamut',
 'ColorMatch RGB',
 'DCDM XYZ',
 'DCI-P3',
 'DCI-P3-P',
 'DJI D-Gamut',
 'DRAGONcolor',
 'DRAGONcolor2',
 'DaVinci Wide Gamut',
 'Display P3',
 'Don RGB 4',
 'EBU Tech. 3213-E',
 'ECI RGB v2',
 'ERIMM RGB',
 'Ekta Space PS 5',
 'F-Gamut',
 'FilmLight E-Gamut',
 'ITU-R BT.2020',
 'ITU-R BT.470 - 525',
 'ITU-R BT.470 - 625',
 'ITU-R BT.709',
 'ITU-T H.273 - 22 Unspecified',
 'ITU-T H.273 - Generic Film',
 'Max RGB',
 'N-Gamut',
 'NTSC (1953)',
 'NTSC (1987)',
 'P3-D65',
 'PLASA ANSI E1.54',
```

(continues on next page)

(continued from previous page)

```
 'Pal/Secam',
 'ProPhoto RGB',
 'Protune Native',
 'REDWideGamutRGB',
 'REDcolor',
 'REDcolor2',
 'REDcolor3',
 'REDcolor4',
 'RIMM RGB',
 'ROMM RGB',
 'Russell RGB',
 'S-Gamut',
 'S-Gamut3',
 'S-Gamut3.Cine',
 'SMPTE 240M',
 'SMPTE C',
 'Sharp RGB',
 'V-Gamut',
 'Venice S-Gamut3',
 'Venice S-Gamut3.Cine',
 'Xtreme RGB',
 'aces',
 'adobe1998',
 'prophoto',
 'sRGB']
```

2.12.31 3.12.31 OETFs

```
>>> sorted(colour.OETFS)
['ARIB STD-B67',
 'Blackmagic Film Generation 5',
 'DaVinci Intermediate',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ITU-R BT.601',
 'ITU-R BT.709',
 'ITU-T H.273 IEC 61966-2',
 'ITU-T H.273 Log',
 'ITU-T H.273 Log Sqrt',
 'SMPTE 240M']
```

2.12.32 3.12.32 EOTFs

```
>>> sorted(colour.EOTFS)
['DCDM',
 'DICOM GSDF',
 'ITU-R BT.1886',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ITU-T H.273 ST.428-1',
 'SMPTE 240M',
```

(continues on next page)

(continued from previous page)

```
 'ST 2084',
 'sRGB']
```

2.12.33 3.12.33 OOTFs

```
>>> sorted(colour.OOTFS)
['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ']
```

2.12.34 3.12.34 Log Encoding / Decoding

```
>>> sorted(colour.LOG_ENCODINGS)
['ACEScc',
 'ACEScct',
 'ACESproxy',
 'ARRI LogC3',
 'ARRI LogC4',
 'Canon Log',
 'Canon Log 2',
 'Canon Log 3',
 'Cineon',
 'D-Log',
 'ERIMM RGB',
 'F-Log',
 'F-Log2',
 'Filmic Pro 6',
 'L-Log',
 'Log2',
 'Log3G10',
 'Log3G12',
 'N-Log',
 'PLog',
 'Panalog',
 'Protune',
 'REDLog',
 'REDLogFilm',
 'S-Log',
 'S-Log2',
 'S-Log3',
 'T-Log',
 'V-Log',
 'ViperLog']
```

2.12.35 3.12.35 CCTFs Encoding / Decoding

```
>>> sorted(colour.CCTF_ENCODINGS)
['ACEScc',
 'ACEScct',
 'ACESproxy',
 'ARRI LogC3',
 'ARRI LogC4',
 'ARIB STD-B67',
 'Canon Log',
 'Canon Log 2',
 'Canon Log 3',
 'Cineon',
 'D-Log',
 'DCDM',
 'DICOM GSDF',
 'ERIMM RGB',
 'F-Log',
 'F-Log2',
 'Filmic Pro 6',
 'Gamma 2.2',
 'Gamma 2.4',
 'Gamma 2.6',
 'ITU-R BT.1886',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ITU-R BT.601',
 'ITU-R BT.709',
 'Log2',
 'Log3G10',
 'Log3G12',
 'PLog',
 'Panalog',
 'ProPhoto RGB',
 'Protune',
 'REDLog',
 'REDLogFilm',
 'RIMM RGB',
 'ROMM RGB',
 'S-Log',
 'S-Log2',
 'S-Log3',
 'SMPTE 240M',
 'ST 2084',
 'T-Log',
 'V-Log',
 'ViperLog',
 'sRGB']
```

2.12.36 3.12.36 Recommendation ITU-T H.273 Code points for Video Signal Type Identification

```
>>> colour.COLOUR_PRIMARIES_ITUTH273.keys()
dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 22, 23])
>>> colour.COLOUR_PRIMARIES_ITUTH273.keys()
>>> description = colour.models.describe_video_signal_colour_primaries(1)
===============================================================================
*                                                                             *
*   Colour Primaries: 1                                                       *
*   -------------------                                                       *
*                                                                             *
*   Primaries          : [[ 0.64  0.33]                                       *
*                         [ 0.3   0.6 ]                                       *
*                         [ 0.15  0.06]]                                      *
*   Whitepoint         : [ 0.3127  0.329 ]                                    *
*   Whitepoint Name    : D65                                                  *
*   NPM                : [[ 0.4123908   0.35758434  0.18048079]               *
*                         [ 0.21263901  0.71516868  0.07219232]               *
*                         [ 0.01933082  0.11919478  0.95053215]]              *
*   NPM -1             : [[ 3.24096994 -1.53738318 -0.49861076]               *
*                         [-0.96924364  1.8759675   0.04155506]               *
*                         [ 0.05563008 -0.20397696  1.05697151]]              *
*   FFmpeg Constants : ['AVCOL_PRI_BT709', 'BT709']                           *
*                                                                             *
===============================================================================
>>> colour.TRANSFER_CHARACTERISTICS_ITUTH273.keys()
dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
>>> description = (
...     colour.models.describe_video_signal_transfer_characteristics(1)
... )
===============================================================================
*                                                                             *
*   Transfer Characteristics: 1                                               *
*   ---------------------------                                               *
*                                                                             *
*   Function           : <function oetf_BT709 at 0x165bb3550>                 *
*   FFmpeg Constants : ['AVCOL_TRC_BT709', 'BT709']                           *
*                                                                             *
===============================================================================
>>> colour.MATRIX_COEFFICIENTS_ITUTH273.keys()
dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
>>> description = colour.models.describe_video_signal_matrix_coefficients(
...     1
... )
===============================================================================
*                                                                             *
*   Matrix Coefficients: 1                                                    *
*   ----------------------                                                    *
*                                                                             *
*   Matrix Coefficients : [ 0.2126  0.0722]                                   *
*   FFmpeg Constants    : ['AVCOL_SPC_BT709', 'BT709']                        *
*                                                                             *
===============================================================================
```

2.13 3.13 Colour Notation Systems - colour.notation

2.13.1 3.13.1 Munsell Value

```
>>> colour.munsell_value(12.23634268)
4.0824437076525664
>>> sorted(colour.MUNSELL_VALUE_METHODS)
['ASTM D1535',
 'Ladd 1955',
 'McCamy 1987',
 'Moon 1943',
 'Munsell 1933',
 'Priest 1920',
 'Saunderson 1944',
 'astm2008']
```

2.13.2 3.13.2 Munsell Colour

```
>>> colour.xyY_to_munsell_colour([0.38736945, 0.35751656, 0.59362000])
'4.2YR 8.1/5.3'
>>> colour.munsell_colour_to_xyY("4.2YR 8.1/5.3")
array([ 0.38736945,  0.35751656,  0.59362    ])
```

2.14 3.14 Optical Phenomena - colour.phenomena

```
>>> colour.rayleigh_scattering_sd()
SpectralDistribution([[  3.60000000e+02,   5.99101337e-01],
                      [  3.61000000e+02,   5.92170690e-01],
                      [  3.62000000e+02,   5.85341006e-01],
                      ...
                      [  7.78000000e+02,   2.55208377e-02],
                      [  7.79000000e+02,   2.53887969e-02],
                      [  7.80000000e+02,   2.52576106e-02]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={'right': None, 'method': 'Constant', 'left': None}
↪)
```

2.15 3.15 Light Quality - colour.quality

2.15.1 3.15.1 Colour Fidelity Index

```
>>> colour.colour_fidelity_index(colour.SDS_ILLUMINANTS["FL2"])
70.120825477833037
>>> sorted(colour.COLOUR_FIDELITY_INDEX_METHODS)
['ANSI/IES TM-30-18', 'CIE 2017']
```

2.15.2 3.15.2 Colour Quality Scale

```
>>> colour.colour_quality_scale(colour.SDS_ILLUMINANTS["FL2"])
64.111703163816699
>>> sorted(colour.COLOUR_QUALITY_SCALE_METHODS)
['NIST CQS 7.4', 'NIST CQS 9.0']
```

2.15.3 3.15.3 Colour Rendering Index

```
>>> colour.colour_rendering_index(colour.SDS_ILLUMINANTS["FL2"])
64.233724121664807
```

2.15.4 3.15.4 Academy Spectral Similarity Index (SSI)

```
>>> colour.spectral_similarity_index(
...     colour.SDS_ILLUMINANTS["C"], colour.SDS_ILLUMINANTS["D65"]
... )
94.0
```

2.16 3.16 Spectral Up-Sampling & Recovery - `colour.recovery`

2.16.1 3.16.1 Reflectance Recovery

```
>>> colour.XYZ_to_sd([0.20654008, 0.12197225, 0.05136952])
SpectralDistribution([[  3.60000000e+02,   8.40144095e-02],
                      [  3.65000000e+02,   8.41264236e-02],
                      [  3.70000000e+02,   8.40057597e-02],
                      ...
                      [  7.70000000e+02,   4.46743012e-01],
                      [  7.75000000e+02,   4.46817187e-01],
                      [  7.80000000e+02,   4.46857696e-01]],
                     SpragueInterpolator,
                     {},
                     Extrapolator,
                     {'method': 'Constant', 'left': None, 'right': None})

>>> sorted(colour.REFLECTANCE_RECOVERY_METHODS)
['Jakob 2019', 'Mallett 2019', 'Meng 2015', 'Otsu 2018', 'Smits 1999']
```

2.16.2 3.16.2 Camera RGB Sensitivities Recovery

```
>>> illuminant = colour.colorimetry.SDS_ILLUMINANTS["D65"]
>>> sensitivities = colour.characterisation.MSDS_CAMERA_SENSITIVITIES[
...     "Nikon 5100 (NPL)"
... ]
>>> reflectances = [
...     sd.copy().align(
...         colour.recovery.SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017
...     )
```

(continues on next page)

(continued from previous page)

```
...     for sd in colour.characterisation.SDS_COLOURCHECKERS[
...         "BabelColor Average"
...     ].values()
... ]
>>> reflectances = colour.colorimetry.sds_and_msds_to_msds(reflectances)
>>> RGB = colour.colorimetry.msds_to_XYZ(
...     reflectances,
...     method="Integration",
...     cmfs=sensitivities,
...     illuminant=illuminant,
...     k=0.01,
...     shape=colour.recovery.SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017,
... )
>>> colour.recovery.RGB_to_msds_camera_sensitivities_Jiang2013(
...     RGB,
...     illuminant,
...     reflectances,
...     colour.recovery.BASIS_FUNCTIONS_DYER2017,
...     colour.recovery.SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017,
... )
RGB_CameraSensitivities([[  4.00000000e+02,   7.22815777e-03,   9.22506480e-03,
                          -9.88368972e-03],
                         [  4.10000000e+02,  -8.50457609e-03,   1.12777480e-02,
                            3.86248655e-03],
                         [  4.20000000e+02,   4.58191132e-02,   7.15520948e-02,
                            4.04068293e-01],
                         ...
                         [  6.80000000e+02,   4.08276173e-02,   5.55290476e-03,
                            1.39907862e-03],
                         [  6.90000000e+02,  -3.71437574e-03,   2.50935640e-03,
                            3.97652622e-04],
                         [  7.00000000e+02,  -5.62256563e-03,   1.56433970e-03,
                            5.84726936e-04]],
                        ['red', 'green', 'blue'],
                        SpragueInterpolator,
                        {},
                        Extrapolator,
                        {'method': 'Constant', 'left': None, 'right': None})
```

2.17 3.17 Correlated Colour Temperature Computation Methods - colour.temperature

```
>>> colour.uv_to_CCT([0.1978, 0.3122])
array([  6.50751282e+03,   3.22335875e-03])
>>> sorted(colour.UV_TO_CCT_METHODS)
['Krystek 1985', 'Ohno 2013', 'Planck 1900', 'Robertson 1968', 'ohno2013', 'robertson1968
↪']
>>> sorted(colour.XY_TO_CCT_METHODS)
['CIE Illuminant D Series',
 'Hernandez 1999',
 'Kang 2002',
 'McCamy 1992',
 'daylight',
```

(continues on next page)

(continued from previous page)

```
 'hernandez1999',
 'kang2002',
 'mccamy1992']
```

2.18 3.18 Colour Volume - colour.volume

```
>>> colour.RGB_colourspace_volume_MonteCarlo(
...     colour.RGB_COLOURSPACE_RGB["sRGB"]
... )
821958.30000000005
```

2.19 3.19 Geometry Primitives Generation - colour.geometry

```
>>> colour.primitive("Grid")
(array([ ([-0.5,  0.5,  0. ], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  0.,  1.]),
       ([ 0.5,  0.5,  0. ], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  0.,  1.]),
       ([-0.5, -0.5,  0. ], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  0.,  1.]),
       ([ 0.5, -0.5,  0. ], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  0.,  1.])],
      dtype=[('position', '<f4', (3,)), ('uv', '<f4', (2,)), ('normal', '<f4', (3,)), (
↪'colour', '<f4', (4,))]), array([[0, 2, 1],
       [2, 3, 1]], dtype=uint32), array([[0, 2],
       [2, 3],
       [3, 1],
       [1, 0]], dtype=uint32))
>>> sorted(colour.PRIMITIVE_METHODS)
['Cube', 'Grid']
>>> colour.primitive_vertices("Quad MPL")
array([[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  0.],
       [ 0.,  1.,  0.]])
>>> sorted(colour.PRIMITIVE_VERTICES_METHODS)
['Cube MPL', 'Grid MPL', 'Quad MPL', 'Sphere']
```

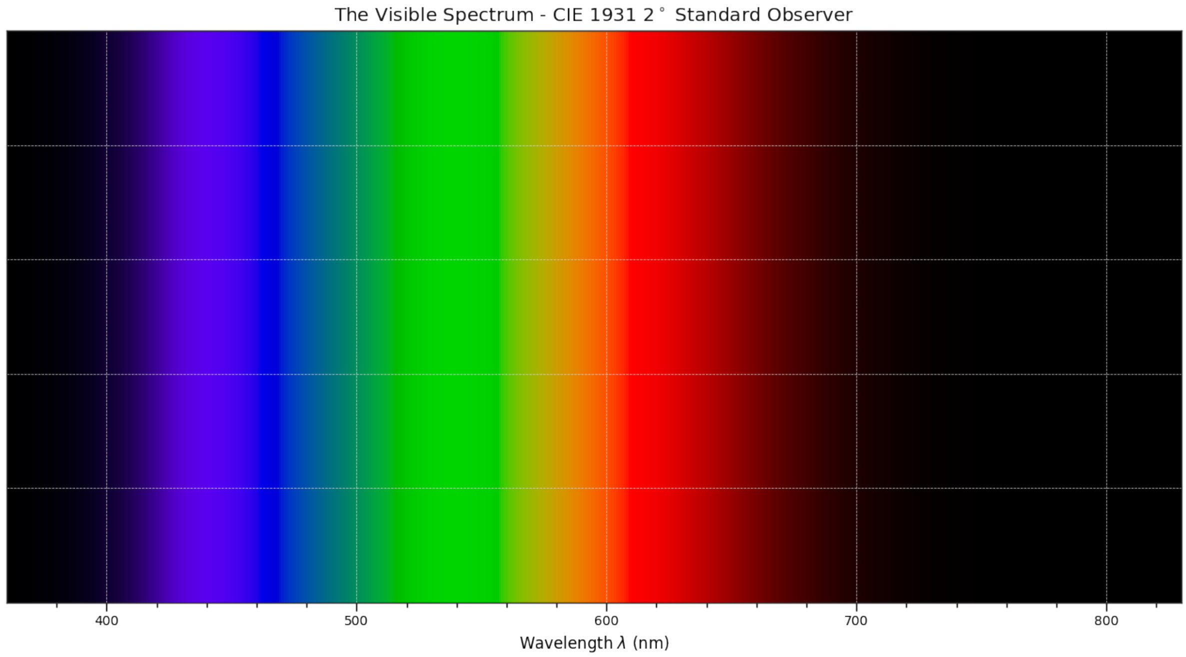
2.20 3.20 Plotting - colour.plotting

Most of the objects are available from the colour.plotting namespace:

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

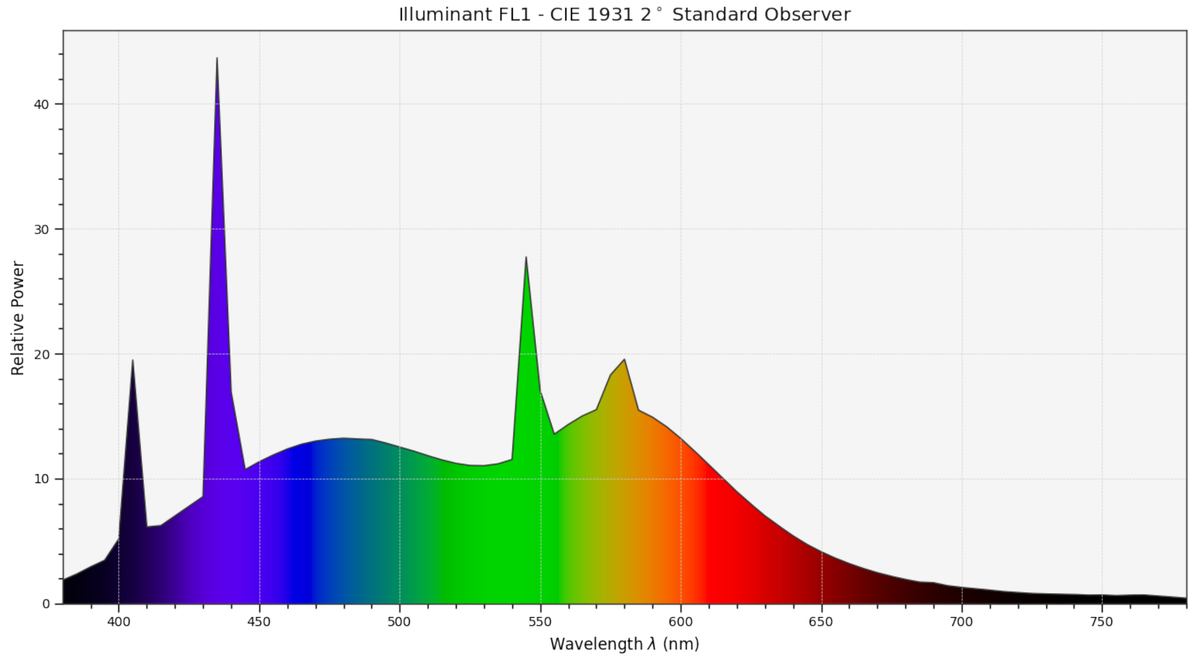
2.20.1 3.20.1 Visible Spectrum

```
>>> plot_visible_spectrum("CIE 1931 2 Degree Standard Observer")
```



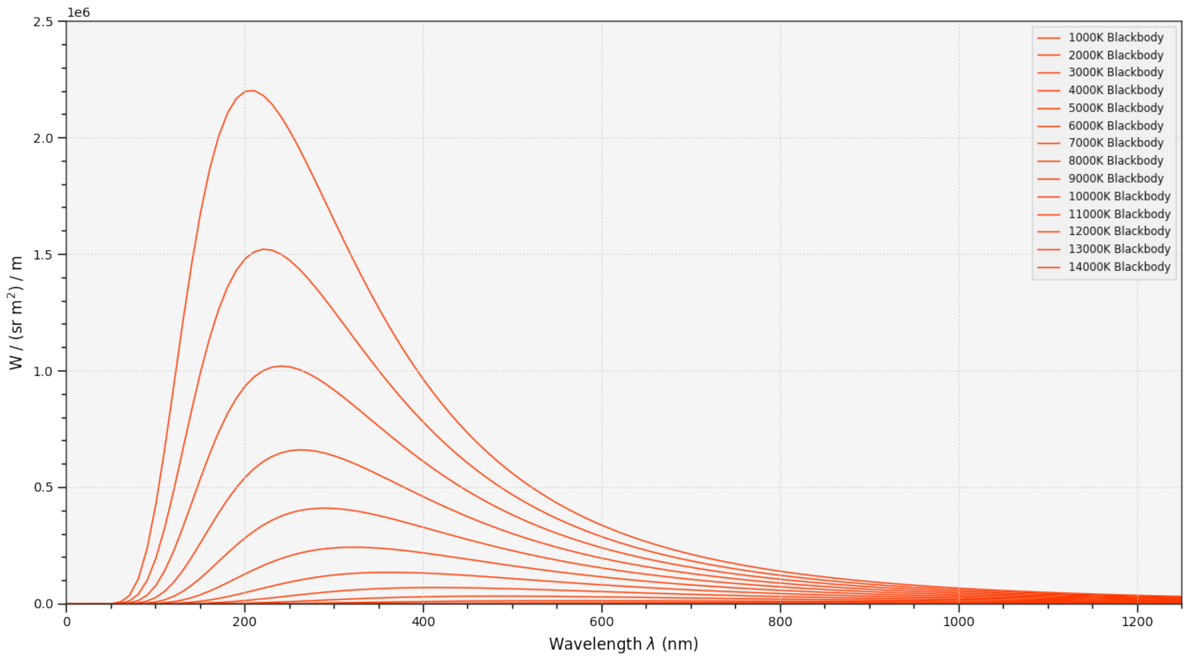
2.20.2 3.20.2 Spectral Distribution

```
>>> plot_single_illuminant_sd("FL1")
```



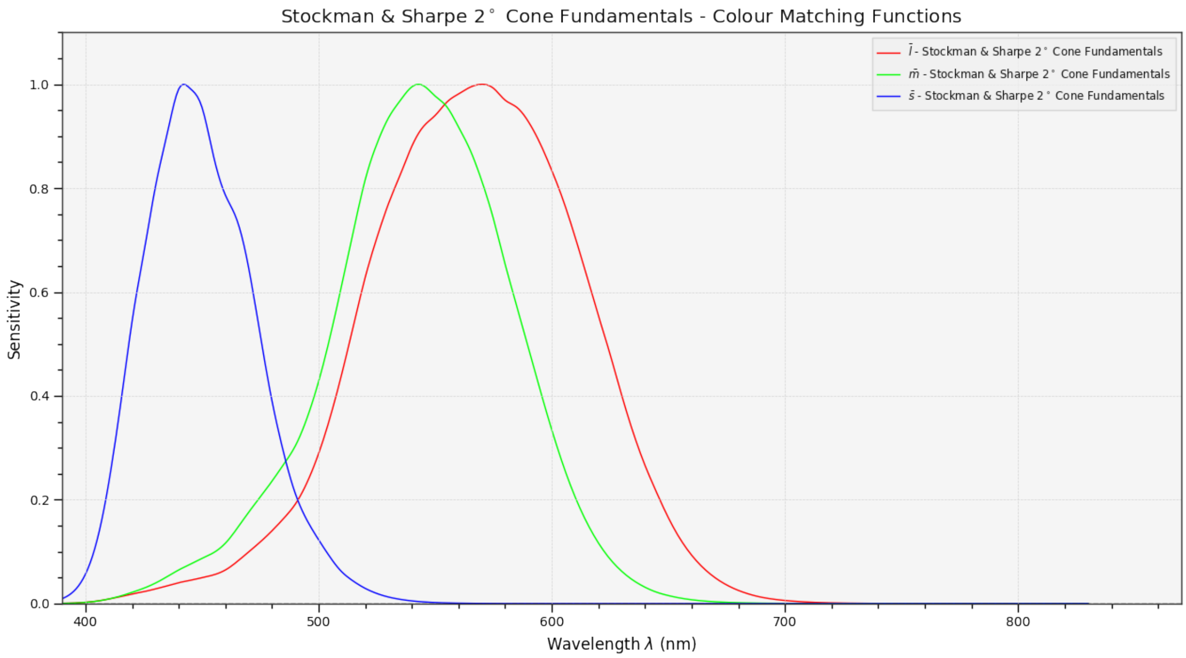
2.20.3 3.20.3 Blackbody

```
>>> blackbody_sds = [
...     colour.sd_blackbody(i, colour.SpectralShape(0, 10000, 10))
...     for i in range(1000, 15000, 1000)
... ]
>>> plot_multi_sds(
...     blackbody_sds,
...     y_label="W / (sr m$^2$) / m",
...     plot_kwargs={"use_sd_colours": True, "normalise_sd_colours": True},
...     legend_location="upper right",
...     bounding_box=(0, 1250, 0, 2.5e6),
... )
```



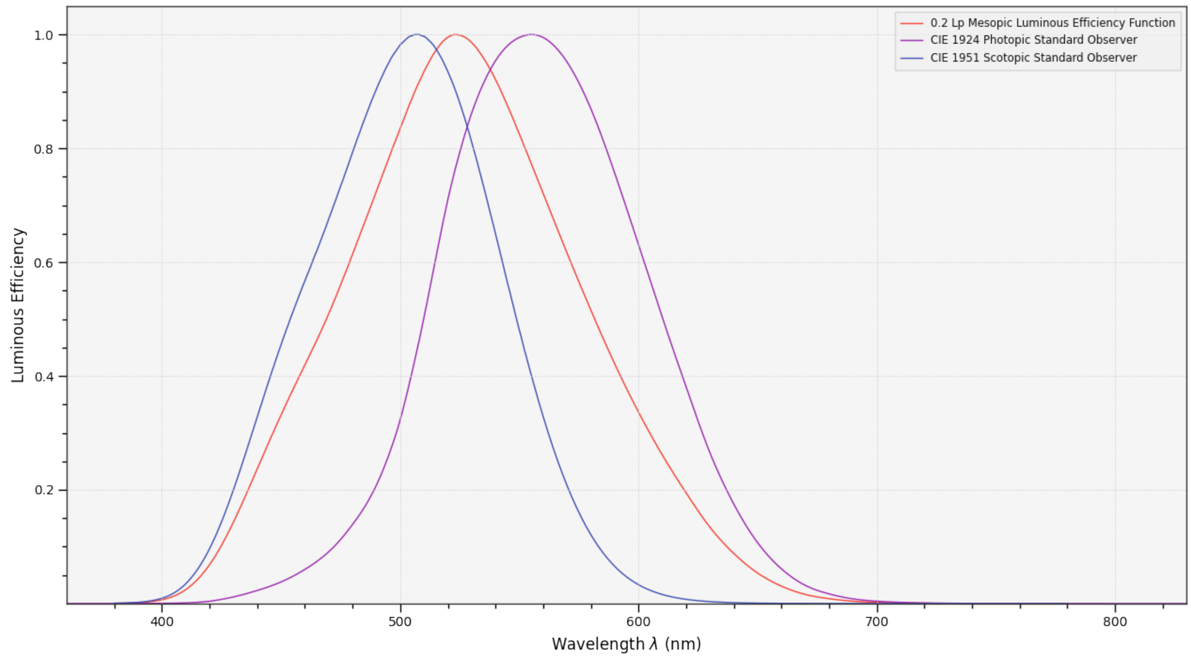
2.20.4 3.20.4 Colour Matching Functions

```
>>> plot_single_cmfs(
...     "Stockman & Sharpe 2 Degree Cone Fundamentals",
...     y_label="Sensitivity",
...     bounding_box=(390, 870, 0, 1.1),
... )
```



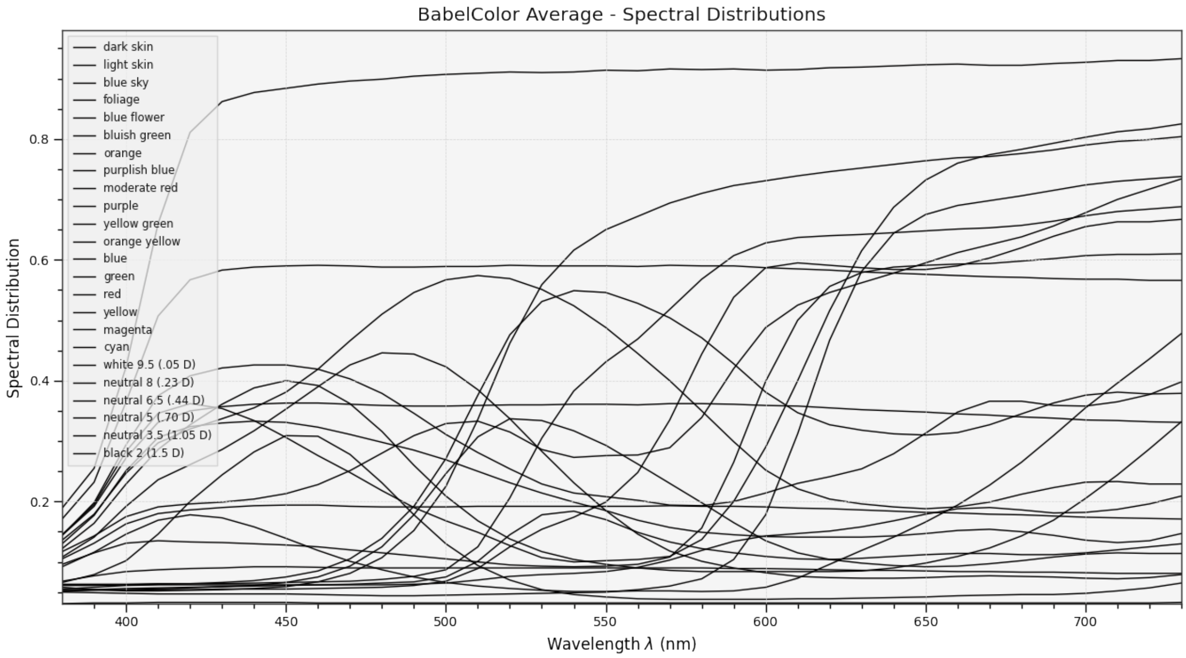
2.20.5 3.20.5 Luminous Efficiency

```
>>> sd_mesopic_luminous_efficiency_function = (
...     colour.sd_mesopic_luminous_efficiency_function(0.2)
... )
>>> plot_multi_sds(
...     (
...         sd_mesopic_luminous_efficiency_function,
...         colour.PHOTOPIC_LEFS["CIE 1924 Photopic Standard Observer"],
...         colour.SCOTOPIC_LEFS["CIE 1951 Scotopic Standard Observer"],
...     ),
...     y_label="Luminous Efficiency",
...     legend_location="upper right",
...     y_tighten=True,
...     margins=(0, 0, 0, 0.1),
... )
```



2.20.6 3.20.6 Colour Checker

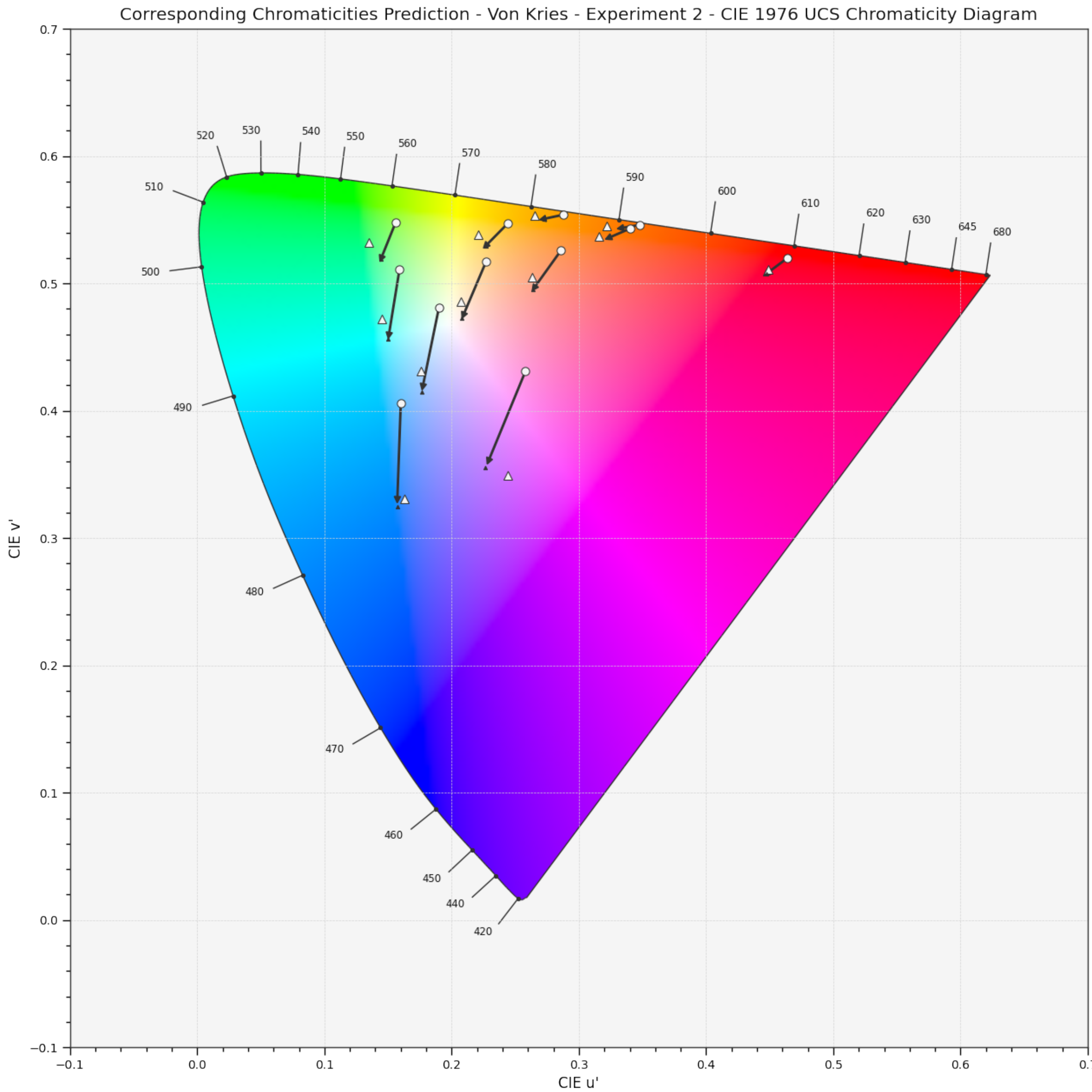
```
>>> from colour.characterisation.dataset.colour_checkers.sds import (
...     COLOURCHECKER_INDEXES_TO_NAMES_MAPPING,
... )
>>> plot_multi_sds(
...     [
...         colour.SDS_COLOURCHECKERS["BabelColor Average"][value]
...         for key, value in sorted(
...             COLOURCHECKER_INDEXES_TO_NAMES_MAPPING.items()
...         )
...     ],
...     plot_kwargs={
...         "use_sd_colours": True,
...     },
...     title=("BabelColor Average - " "Spectral Distributions"),
... )
```




```
>>> plot_single_colour_checker(
...     "ColorChecker 2005", text_kwargs={"visible": False}
... )
```

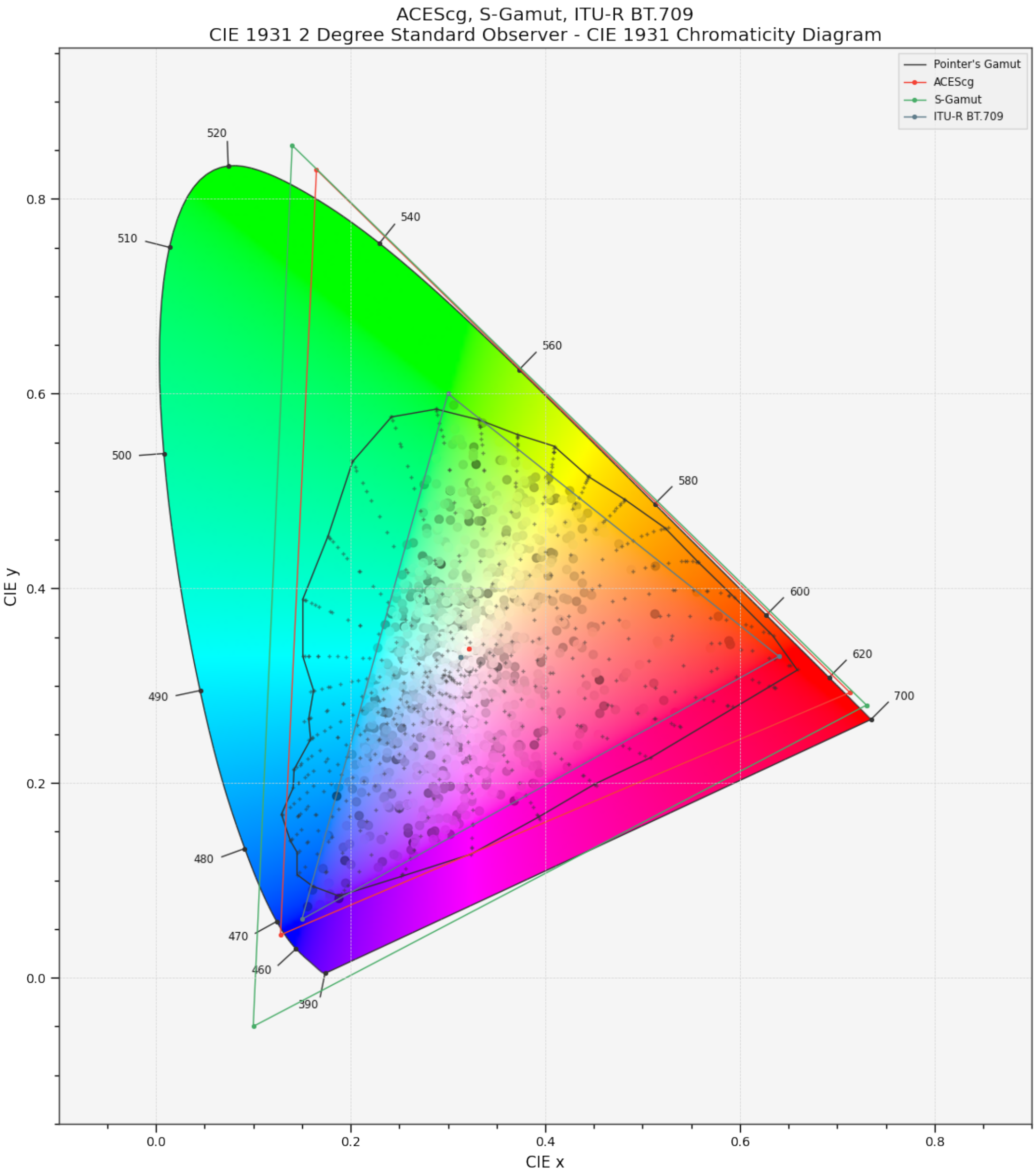
2.20.7 3.20.7 Chromaticities Prediction

```
>>> plot_corresponding_chromaticities_prediction(
...     2, "Von Kries", "Bianco 2010"
... )
```



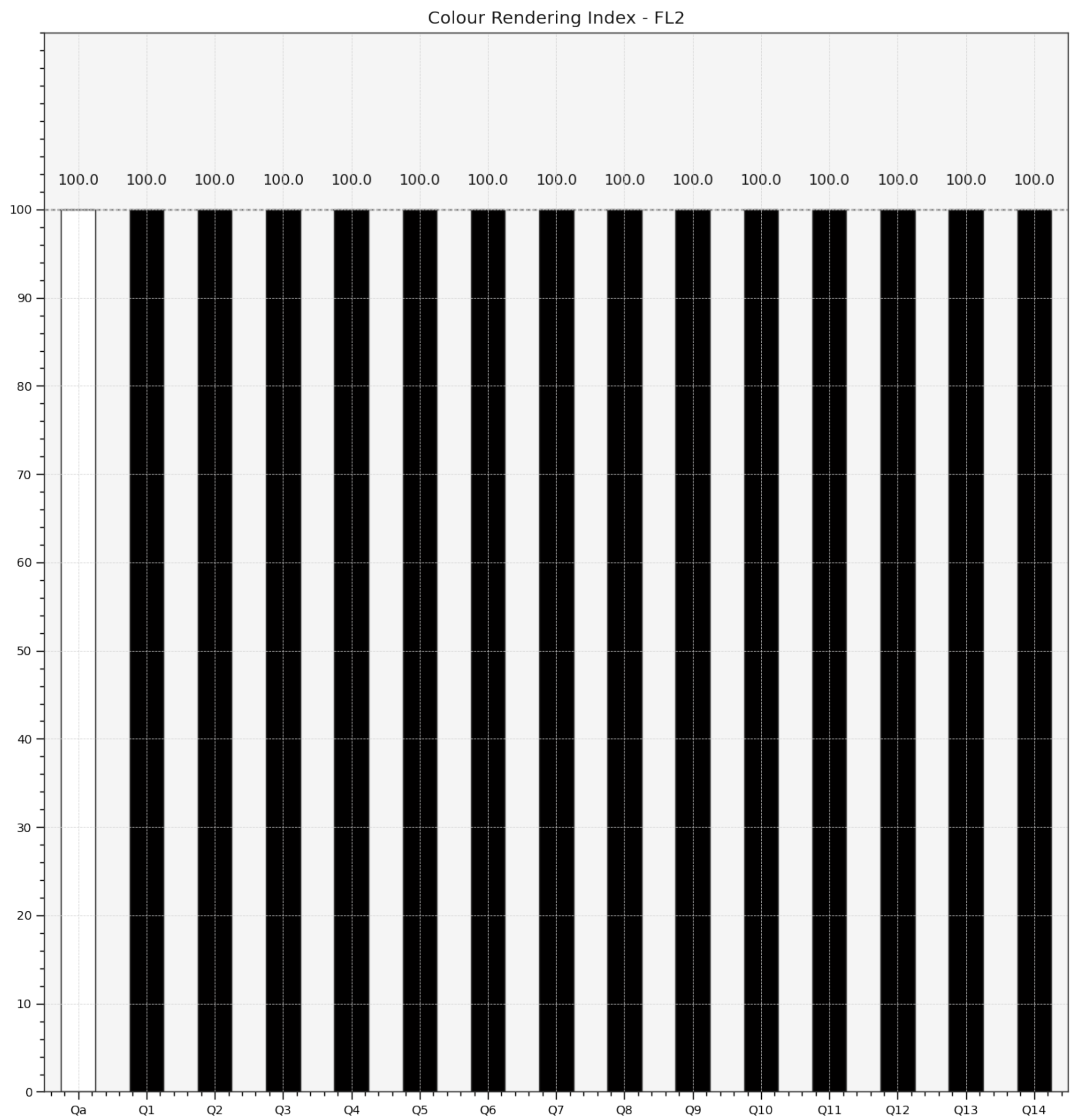
2.20.8 3.20.8 Chromaticities

```
>>> import numpy as np
>>> RGB = np.random.random((32, 32, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(
...     RGB,
...     "ITU-R BT.709",
...     colourspaces=["ACEScg", "S-Gamut"],
...     show_pointer_gamut=True,
... )
```



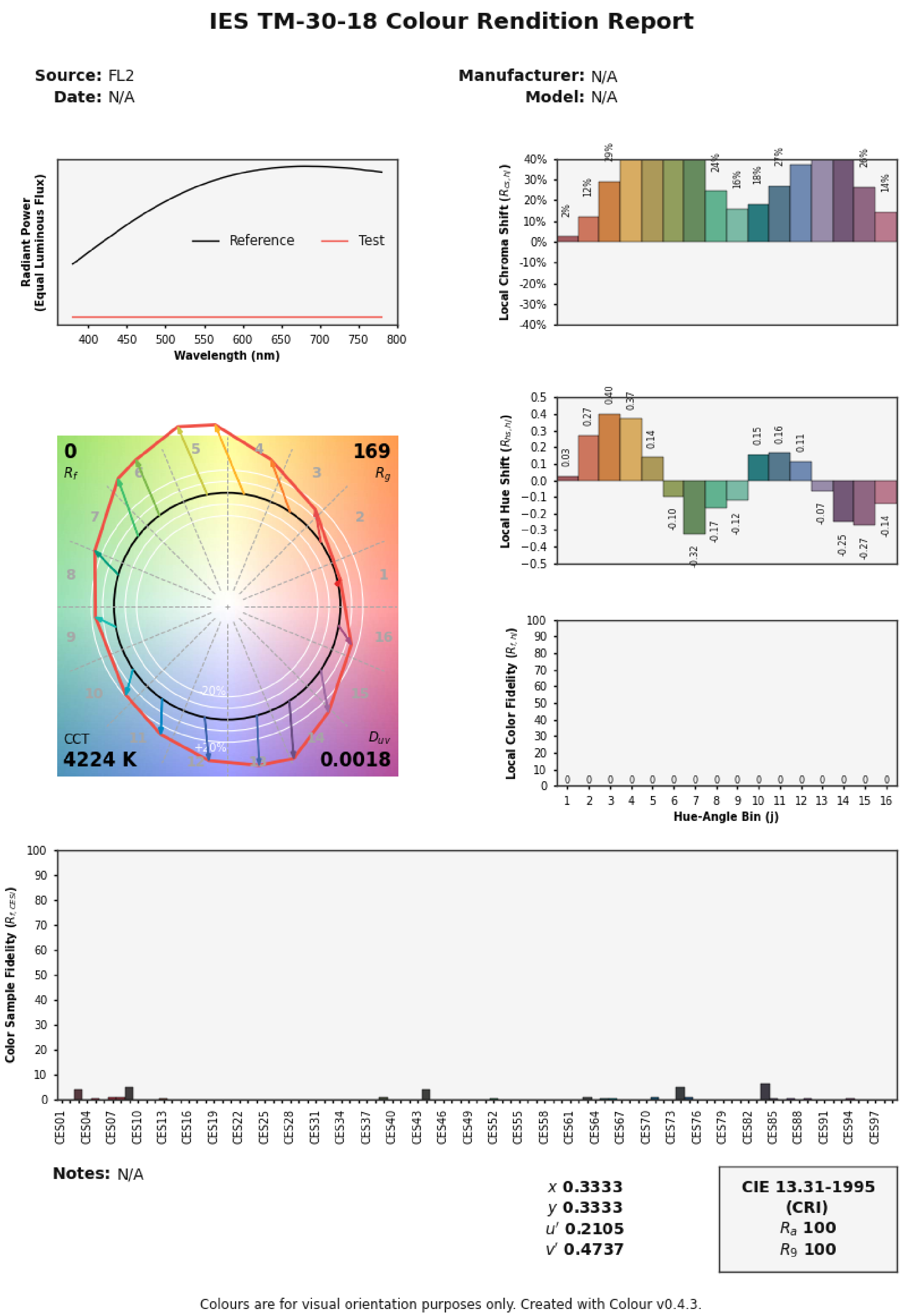
2.20.9 3.20.9 Colour Rendering Index

```
>>> plot_single_sd_colour_rendering_index_bars(
...     colour.SDS_ILLUMINANTS["FL2"]
... )
```



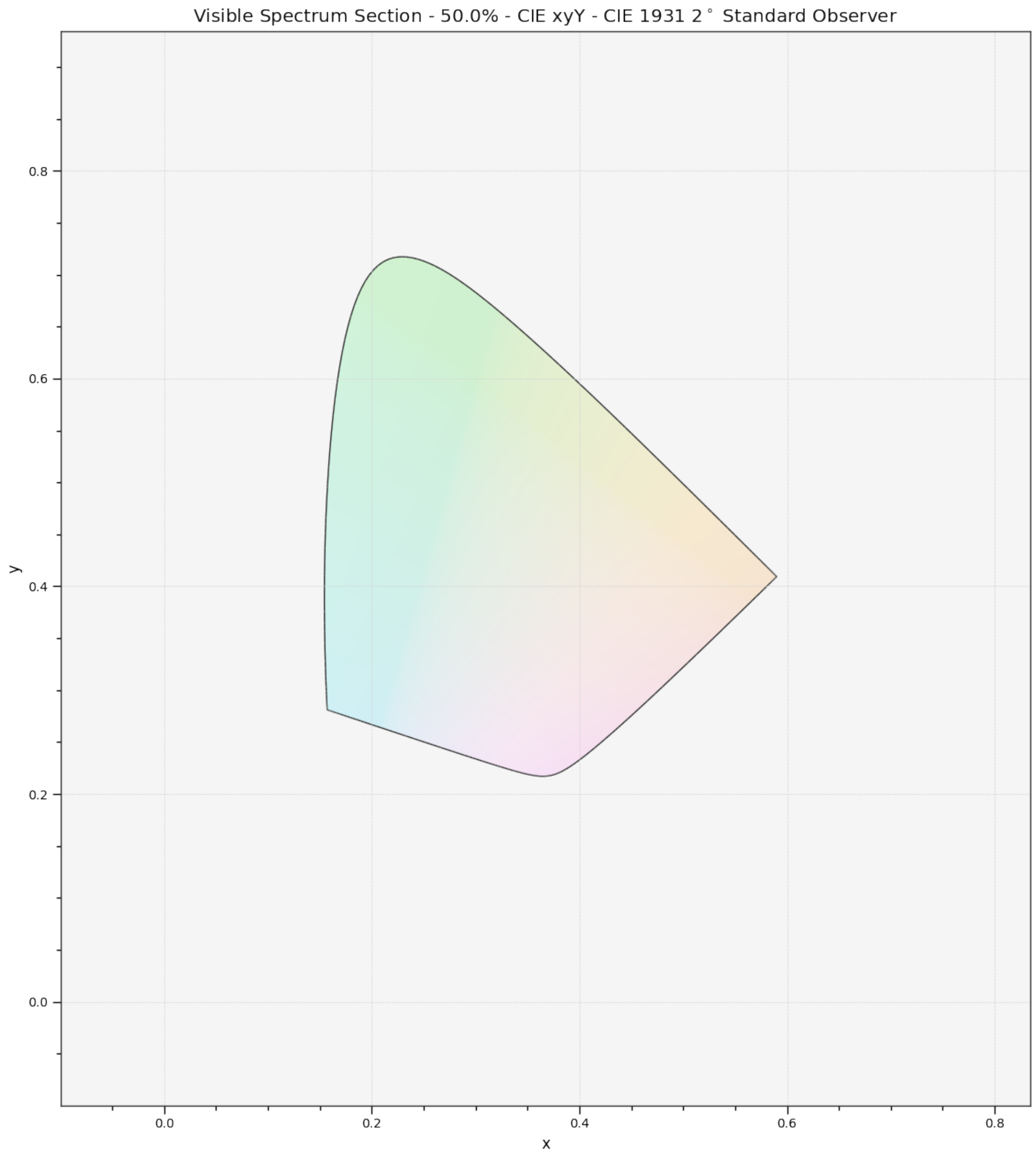
2.20.10 3.20.10 ANSI/IES TM-30-18 Colour Rendition Report

```
>>> plot_single_sd_colour_rendition_report(colour.SDS_ILLUMINANTS["FL2"])
```

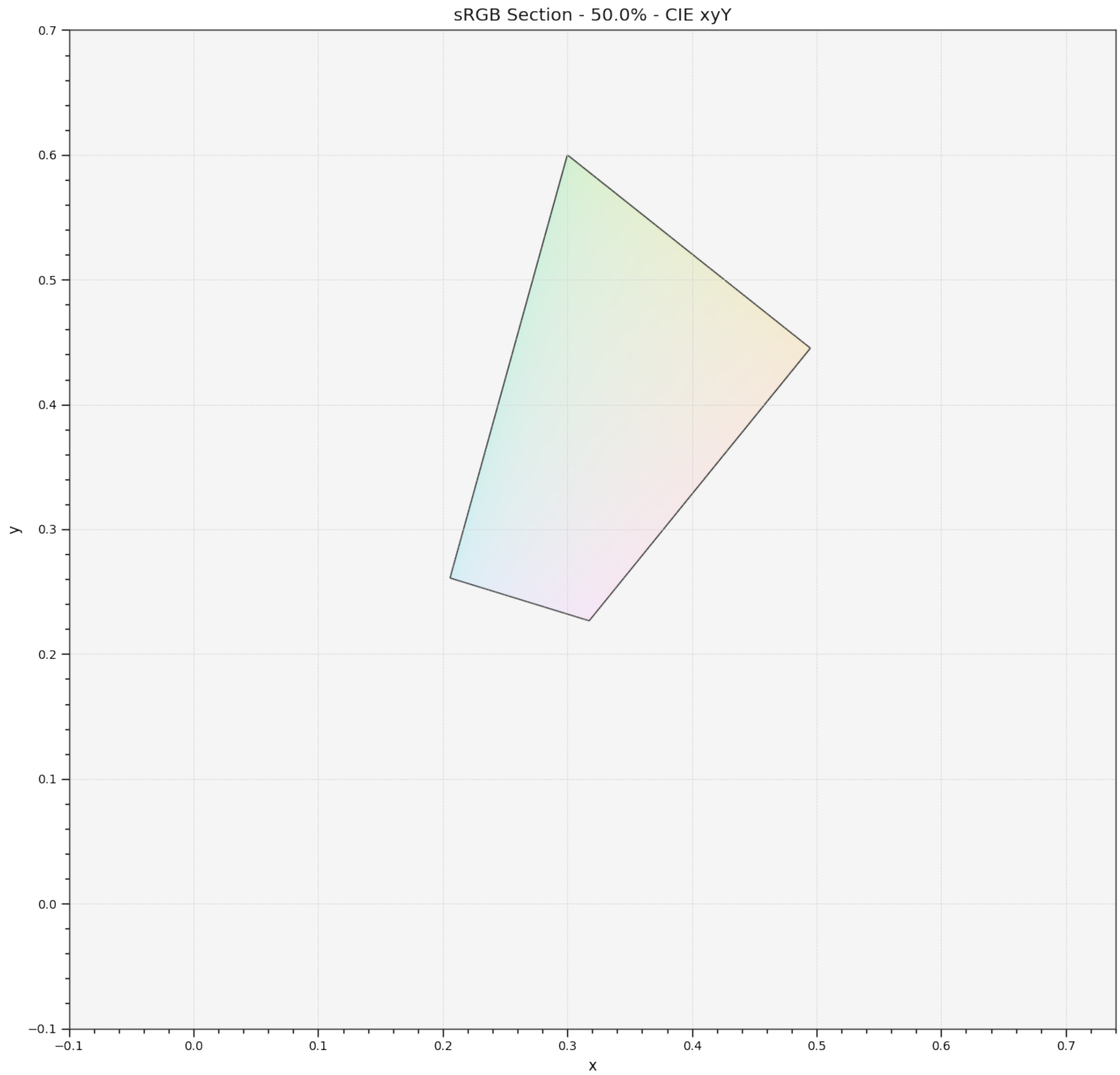

IES TM-30-18 Colour Rendition Report
Source: FL2
Date: N/A
Manufacturer: N/A
Model: N/A
Radiant Power (Equal Luminous Flux)
Reference
Test
Wavelength (nm)
Local Chroma Shift ($R_{cs,hj}$)
Local Hue Shift ($R_{hs,hj}$)
Local Color Fidelity ($R_{f,hj}$)
Hue-Angle Bin (j)
0
R_f
169
R_g
CCT
4224 K
D_{uv}
0.0018
Color Sample Fidelity ($R_{f,CESi}$)
Notes: N/A
x 0.3333
y 0.3333
u' 0.2105
v' 0.4737
CIE 13.31-1995
(CRI)
R_a 100
R_9 100
Colours are for visual orientation purposes only. Created with Colour v0.4.3.

2.20.11 3.20.11 Gamut Section

```
>>> plot_visible_spectrum_section(
...     section_colours="RGB", section_opacity=0.15
... )
```

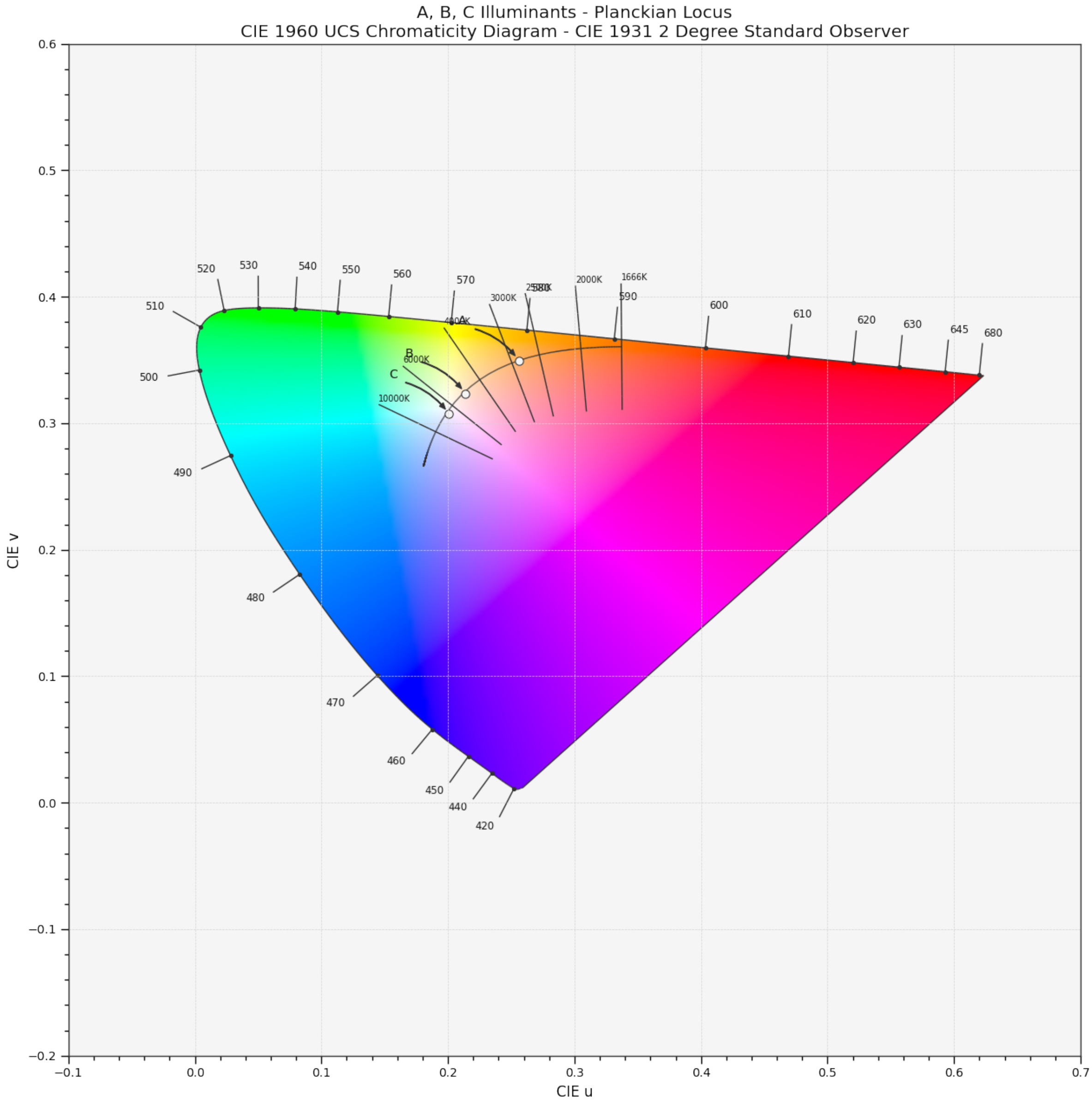



```
>>> plot_RGB_colourspace_section(
...     "sRGB", section_colours="RGB", section_opacity=0.15
... )
```



2.20.12 3.20.12 Colour Temperature

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(
...     ["A", "B", "C"]
... )
```


A, B, C Illuminants - Planckian Locus
CIE 1960 UCS Chromaticity Diagram - CIE 1931 2 Degree Standard Observer
CIE v
CIE u
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
510
520
530
540
550
560
570
590
600
610
620
630
645
680
500
490
480
470
460
450
440
420
1666K
2000K
3000K
10000K
6000K
A
B
C

CHAPTER

THREE

4 USER GUIDE

3.1 User Guide

The user guide provides an overview of **Colour** and explains important concepts and features, details can be found in the API Reference.

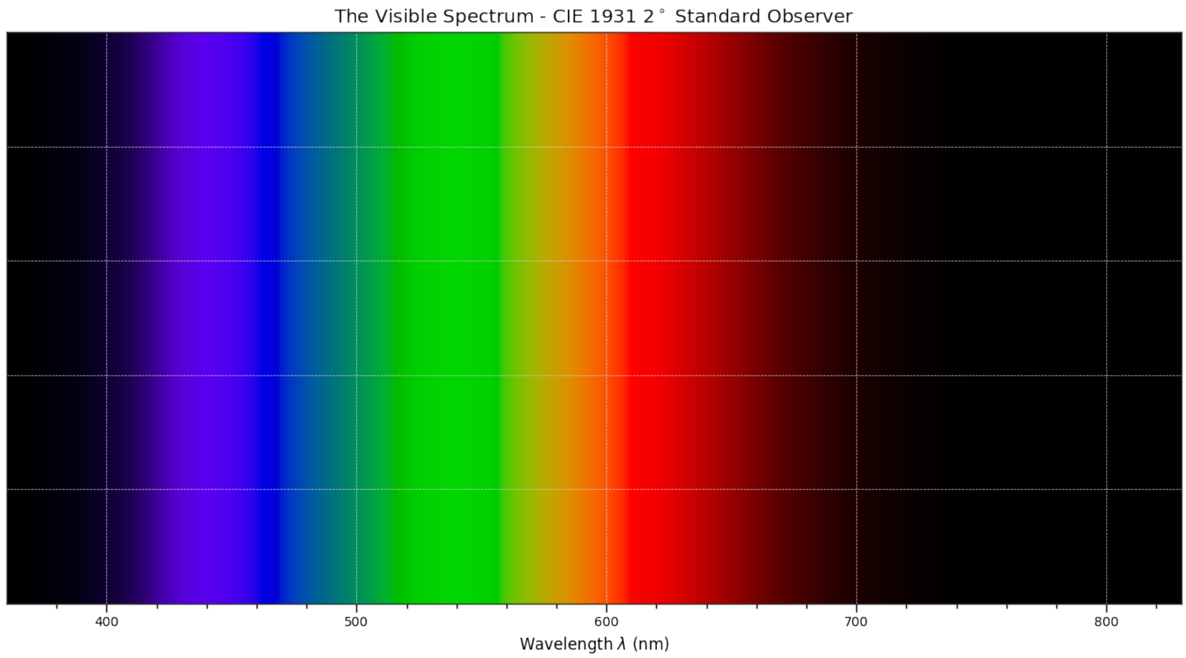
3.1.1 Tutorial

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

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

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

```
from colour.plotting import *

colour_style()

plot_visible_spectrum()
```



Overview

Colour is organised around various sub-packages:

- *adaptation*: Chromatic adaptation models and transformations.
- *algebra*: Algebra utilities.
- *appearance*: Colour appearance models.
- *biochemistry*: Biochemistry computations.
- *blindness*: Colour vision deficiency models.
- *characterisation*: Colour correction, camera and display characterisation.
- *colorimetry*: Core objects for colour computations.
- *constants*: *CIE* and *CODATA* constants.
- *continuous*: Base objects for continuous data representation.
- *contrast*: Objects for contrast sensitivity computation.
- *corresponding*: Corresponding colour chromaticities computations.
- *difference*: Colour difference computations.
- *geometry*: Geometry primitives generation.
- *graph*: Graph for automatic colour conversions.
- *hints*: Type hints for annotations.
- *io*: Input / output objects for reading and writing data.
- *models*: Colour models.
- *notation*: Colour notation systems.
- *phenomena*: Computation of various optical phenomena.
- *plotting*: Diagrams, figures, etc...
- *quality*: Colour quality computation.

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

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

```
import colour

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

```
['domain_range_scale', 'get_domain_range_scale', 'set_domain_range_scale', 'CHROMATIC_
↪ADAPTATION_METHODS', 'CHROMATIC_ADAPTATION_TRANSFORMS', '...']
```

The various sub-packages also expose their public API:

```
from pprint import pprint

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

```
Adaptation
['CHROMATIC_ADAPTATION_TRANSFORMS',
 'CAT_BIANCO2010',
 'CAT_BRADFORD',
 'CAT_CAT02',
 'CAT_CAT02_BRILL2008',
 '...']
```

(continues on next page)

(continued from previous page)

```
Algebra
['get_sdiv_mode',
 'set_sdiv_mode',
 'sdiv_mode',
 'sdiv',
 'is_spow_enabled',
 '...']


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


Biochemistry
['REACTION_RATE_MICHAELISMENTEN_METHODS',
 'reaction_rate_MichaelisMenten',
 'SUBSTRATE_CONCENTRATION_MICHAELISMENTEN_METHODS',
 'substrate_concentration_MichaelisMenten',
 'reaction_rate_MichaelisMenten_Michaelis1913',
 '...']


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


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


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


Constants
```

(continues on next page)

(continued from previous page)

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


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


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


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


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


Geometry
['ellipse_coefficients_general_form',
 'ellipse_coefficients_canonical_form',
 'point_at_angle_on_ellipse',
 'ellipse_fitting_Halir1998',
 'ELLIPSE_FITTING_METHODS',
 '...']


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

(continues on next page)

(continued from previous page)

```
Hints
['ArrayLike', 'NDArray', 'ModuleType', 'Any', 'Callable', '...']


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


Models
['COLOURSPACE_MODELS',
 'COLOURSPACE_MODELS_AXIS_LABELS',
 'COLOURSPACE_MODELS_DOMAIN_RANGE_SCALE_1_TO_REFERENCE',
 'Jab_to_JCh',
 'JCh_to_Jab',
 '...']


Notation
['CSS_COLOR_3_BASIC',
 'CSS_COLOR_3_EXTENDED',
 'CSS_COLOR_3',
 'MUNSELL_COLOURS_ALL',
 'MUNSELL_COLOURS_1929',
 '...']


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


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


Quality
['SDS_TCS',
 'SDS_VS',
 'ColourRendering_Specification_CIE2017',
 'colour_fidelity_index_CIE2017',
 'ColourQuality_Specification_ANSIIESTM3018',
 '...']
```

(continues on next page)

(continued from previous page)

```
Recovery
['SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017',
 'BASIS_FUNCTIONS_DYER2017',
 'SPECTRAL_SHAPE_sRGB_MALLETT2019',
 'MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019',
 'SPECTRAL_SHAPE_OTSU2018',
 '...']


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


Utilities
['Lookup',
 'Structure',
 'CanonicalMapping',
 'LazyCanonicalMapping',
 'Node',
 '...']


Volume
['OPTIMAL_COLOUR_STIMULI_ILLUMINANTS',
 'is_within_macadam_limits',
 'is_within_mesh_volume',
 'is_within_pointer_gamut',
 'generate_pulse_waves',
 '...']
```

The codebase is documented and most docstrings have usage examples:

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

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

Parameters
----------
CCT_D_uv
    Correlated colour temperature :math:`T_{cp}`, :math:`\Delta_{uv}`.
cmfs
    Standard observer colour matching functions, default to the
    *CIE 1931 2 Degree Standard Observer*.

Returns
-------
:class:`numpy.ndarray`
    *CIE UCS* colourspace *uv* chromaticity coordinates.
```

(continues on next page)

(continued from previous page)

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

Examples
--------
>>> from colour import MSDS_CMFS, SPECTRAL_SHAPE_DEFAULT
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SPECTRAL_SHAPE_DEFAULT)
... )
>>> CCT_D_uv = np.array([6507.4342201047066, 0.003223690901513])
>>> CCT_to_uv_Ohno2013(CCT_D_uv, cmfs)  # doctest: +ELLIPSIS
array([ 0.1977999...,  0.3122004...])
```

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

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

```
['SpectralShape',
 'SPECTRAL_SHAPE_DEFAULT',
 'SpectralDistribution',
 'MultiSpectralDistributions',
 'reshape_sd',
 'reshape_msds',
 'sds_and_msds_to_sds',
 'sds_and_msds_to_msds',
 'planck_law',
 'blackbody_spectral_radiance',
 'sd_blackbody',
 'rayleigh_jeans_law',
 'sd_rayleigh_jeans',
 'LMS_ConeFundamentals',
 'RGB_ColourMatchingFunctions',
 'XYZ_ColourMatchingFunctions',
 'CCS_ILLUMINANTS',
 'MSDS_CMFS',
 'MSDS_CMFS_LMS',
 'MSDS_CMFS_RGB',
 'MSDS_CMFS_STANDARD_OBSERVER',
 'SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES',
 'SDS_ILLUMINANTS',
 'SDS_LEFS',
 'SDS_LEFS_PHOTOPIC',
 'SDS_LEFS_SCOTOPIC',
 'TVS_ILLUMINANTS',
 'TVS_ILLUMINANTS_HUNTERLAB',
 'CCS_LIGHT_SOURCES',
 'SDS_LIGHT_SOURCES',
 'sd_constant',
 'sd_zeros',
 'sd_ones',
 'msds_constant',
```

(continues on next page)

(continued from previous page)

```
'msds_zeros',
'msds_ones',
'SD_GAUSSIAN_METHODS',
'sd_gaussian',
'sd_gaussian_normal',
'sd_gaussian_fwhm',
'SD_SINGLE_LED_METHODS',
'sd_single_led',
'sd_single_led_Ohno2005',
'SD_MULTI_LEDS_METHODS',
'sd_multi_leds',
'sd_multi_leds_Ohno2005',
'SD_TO_XYZ_METHODS',
'MSDS_TO_XYZ_METHODS',
'sd_to_XYZ',
'msds_to_XYZ',
'SPECTRAL_SHAPE_ASTME308',
'handle_spectral_arguments',
'lagrange_coefficients_ASTME2022',
'tristimulus_weighting_factors_ASTME2022',
'adjust_tristimulus_weighting_factors_ASTME308',
'sd_to_XYZ_integration',
'sd_to_XYZ_tristimulus_weighting_factors_ASTME308',
'sd_to_XYZ_ASTME308',
'msds_to_XYZ_integration',
'msds_to_XYZ_ASTME308',
'wavelength_to_XYZ',
'spectral_uniformity',
'BANDPASS_CORRECTION_METHODS',
'bandpass_correction',
'bandpass_correction_Stearns1988',
'sd_CIE_standard_illuminant_A',
'sd_CIE_illuminant_D_series',
'daylight_locus_function',
'sd_mesopic_luminous_efficiency_function',
'mesopic_weighting_function',
'LIGHTNESS_METHODS',
'lightness',
'lightness_Glasser1958',
'lightness_Wyszecki1963',
'lightness_CIE1976',
'lightness_Fairchild2010',
'lightness_Fairchild2011',
'lightness_Abebe2017',
'intermediate_lightness_function_CIE1976',
'LUMINANCE_METHODS',
'luminance',
'luminance_Newhall1943',
'luminance_ASTMD1535',
'luminance_CIE1976',
'luminance_Fairchild2010',
'luminance_Fairchild2011',
'luminance_Abebe2017',
'intermediate_luminance_function_CIE1976',
'dominant_wavelength',
'complementary_wavelength',
```

(continues on next page)

(continued from previous page)

```
 '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',
 'WHITENESS_METHODS',
 'whiteness',
 'whiteness_Berger1959',
 'whiteness_Taube1960',
 'whiteness_Stensby1968',
 'whiteness_ASTME313',
 'whiteness_Ganz1979',
 'whiteness_CIE2004',
 'YELLOWNESS_METHODS',
 'yellowness',
 'yellowness_ASTMD1925',
 'yellowness_ASTME313_alternative',
 'YELLOWNESS_COEFFICIENTS_ASTME313',
 'yellowness_ASTME313']
```

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

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

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

From Spectral Distribution

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

```
# Defining a sample spectral distribution data.
data_sample = {
    380: 0.048,
    385: 0.051,
    390: 0.055,
```

(continues on next page)

(continued from previous page)

```
    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,
```

(continues on next page)

(continued from previous page)

```
    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,
}

sd = colour.SpectralDistribution(data_sample, name="Sample")
print(repr(sd))
```

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

(continues on next page)

(continued from previous page)

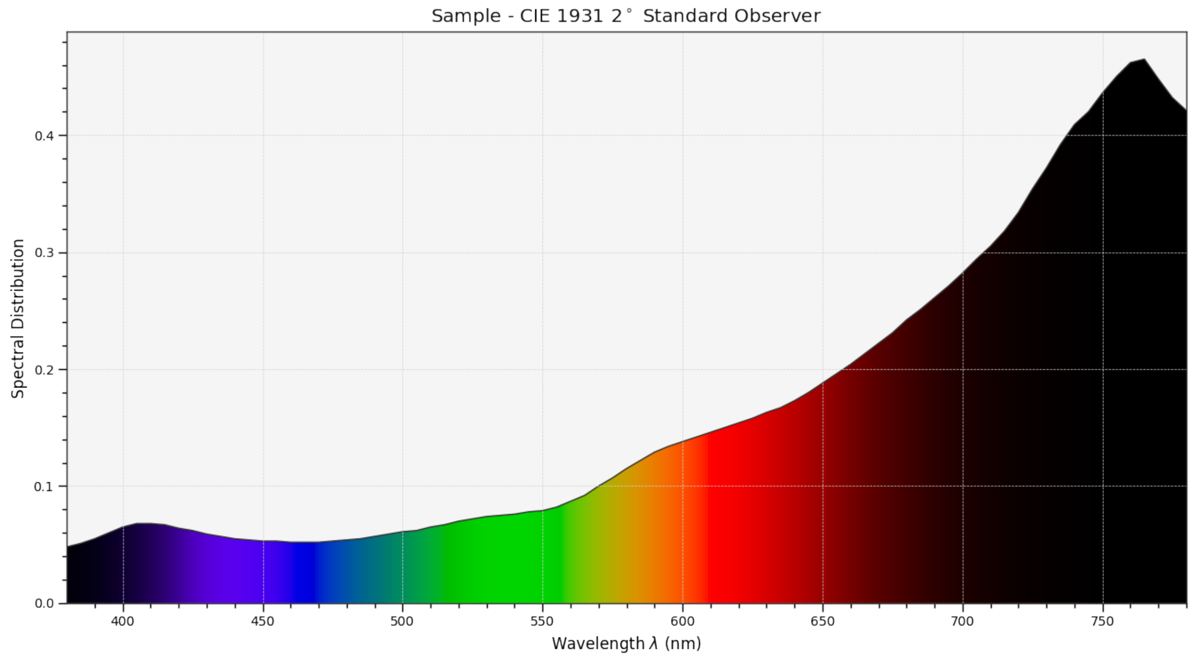
```
                       [  5.25000000e+02,   7.20000000e-02],
                       [  5.30000000e+02,   7.40000000e-02],
                       [  5.35000000e+02,   7.50000000e-02],
                       [  5.40000000e+02,   7.60000000e-02],
                       [  5.45000000e+02,   7.80000000e-02],
                       [  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})
```

(continues on next page)

(continued from previous page)

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

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



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

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

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

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

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

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

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

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

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

```
0.0
1.0
```

(continues on next page)

(continued from previous page)

```
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
```

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

Colour defines three convenient objects to create constant spectral distributions:

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

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

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

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

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

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

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

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

```
print(repr(colour.SPECTRAL_SHAPE_DEFAULT))
```

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

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

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

```
1.0
```

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

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

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

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

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

```
"Ones Filled Spectral Distribution"
1.0

"x2 Constant Multiplied"
2.0

"+ Spectral Distribution"
2.0
```

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

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

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

```
True
```

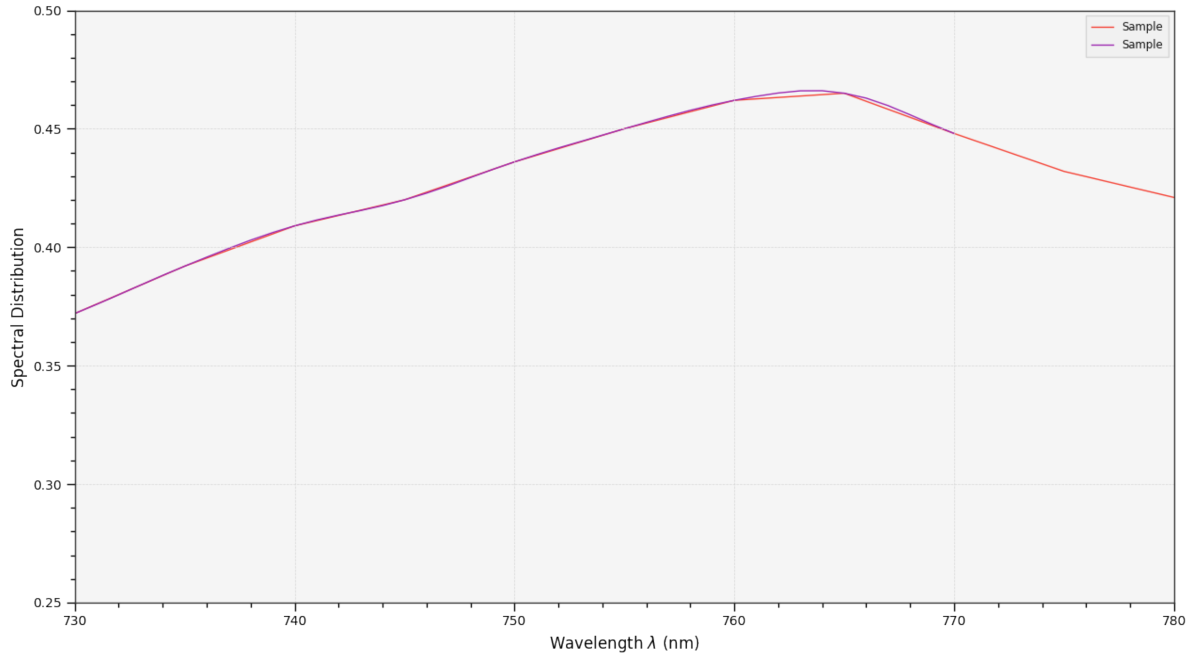
In this case, since the sample spectral distribution is uniform the interpolation defaults to the `colour.SpragueInterpolator` interpolator.

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

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

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

```
0.065809599999999996
```

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



Extrapolation although dangerous can be used to help aligning two spectral distributions together. *CIE publication CIE 15:2004 "Colorimetry"* recommends that unmeasured values may be set equal to the nearest measured value of the appropriate quantity in truncation [CIET14804f]:

```
# Extrapolating the copied sample spectral distribution.
sd_copy.extrapolate(colour.SpectralShape(340, 830, 1))
sd_copy[340], sd_copy[830]
```

```
(0.065000000000000002, 0.44800000000000018)
```

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

```
# Extrapolating the copied sample spectral distribution with *Linear* method.
sd_copy.extrapolate(
    colour.SpectralShape(340, 830, 1),
    extrapolator_kwargs={"method": "Linear", "right": 0},
)
```

(continues on next page)

(continued from previous page)

```
sd_copy[340], sd_copy[830]
```

```
(0.046999999999999348, 0.0)
```

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

```
# Aligning the cloned sample spectral distribution.
# The spectral distribution is first trimmed as above.
sd_copy.interpolate(colour.SpectralShape(400, 700, 1))
sd_copy.align(colour.SpectralShape(340, 830, 5))
sd_copy[340], sd_copy[830]
```

```
(0.065000000000000002, 0.28199999999999975)
```

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

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

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

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

The spectral distribution can be normalised with an arbitrary factor:

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

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

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

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

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

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

```
import numpy as np

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

```
Signal([[   0.,   10.],
        [   1.,   20.],
        [   2.,   30.],
        [   3.,   40.],
        [   4.,   50.],
        [   5.,   60.],
        [   6.,   70.],
        [   7.,   80.],
        [   8.,   90.],
        [   9.,  100.]],
       interpolator=KernelInterpolator,
       interpolator_kwargs={},
       extrapolator=Extrapolator,
       extrapolator_kwargs={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([ 94.74700025,  50.62829102,  72.93120155,  81.86179968,
        70.14736394,  83.11336665,  51.17649751,  71.00638621,
        86.94761009,  78.01845818])
```

Convert to Tristimulus Values

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

```
sd = colour.SpectralDistribution(data_sample)
cmfs = colour.MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
illuminant = colour.SDS_ILLUMINANTS["D65"]

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

```
[ 10.97085572   9.70278591   6.05562778]
```

From *CIE XYZ* Colourspace

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

```
# Displaying objects interacting directly with the *CIE XYZ* colourspace.
pprint(colour.COLOURSPACE_MODELS)
```

```
('CAM02LCD',
 'CAM02SCD',
 'CAM02UCS',
 'CAM16LCD',
 'CAM16SCD',
 'CAM16UCS',
 'CIE XYZ',
 'CIE xyY',
 'CIE Lab',
 'CIE Luv',
 'CIE UCS',
 'CIE UVW',
 'DIN99',
 'Hunter Lab',
 'Hunter Rdab',
 'ICaCb',
 'ICtCp',
 'IPT',
 'IPT Ragoo 2021',
 'IgPgTg',
 'Jzazbz',
 'OSA UCS',
 'Oklab',
 'hdr-CIELAB',
 'hdr-IPT',
 'Yrg')
```

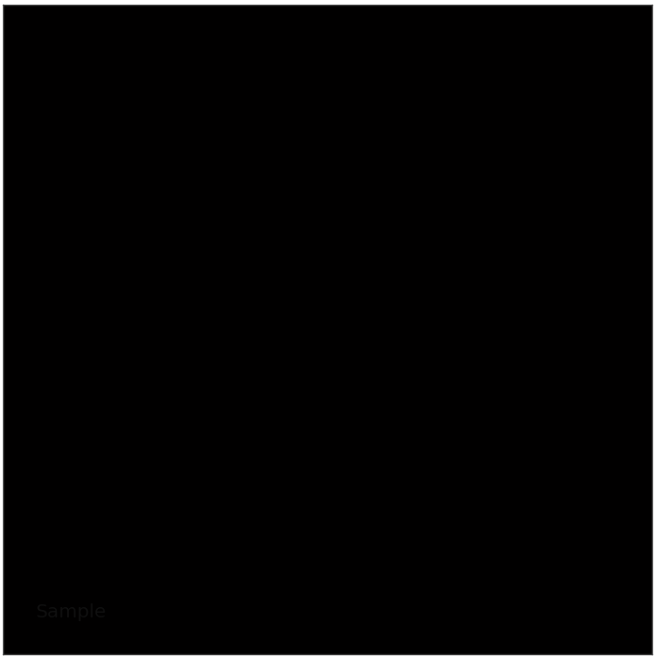
Convert to Display Colours

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

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

```
[ 0.45675795  0.30986982  0.24861924]
```

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



Generate Colour Rendition Charts

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

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

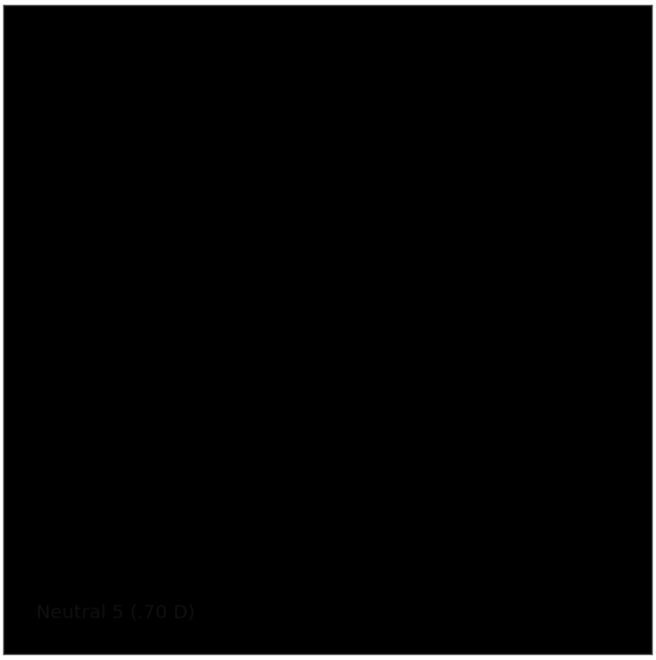
The `colour.characterisation` sub-package contains the dataset for various colour rendition charts:

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

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

```
['BabelColor Average', 'ColorChecker 1976', 'ColorChecker 2005', 'ColorChecker24 - After␣
↪November 2014', 'ColorChecker24 - Before November 2014', 'TE226 V2', 'babel_average',
↪'cc2005', 'cca2014', 'ccb2014']
['BabelColor Average', 'ColorChecker N Ohta', 'ISO 17321-1', 'babel_average', 'cc_ohta']
```

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

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

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



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

```
plot_single_colour_checker(
    colour_checker="ColorChecker 2005", text_kwargs={"visible": False}
)
```

Convert to Chromaticity Coordinates

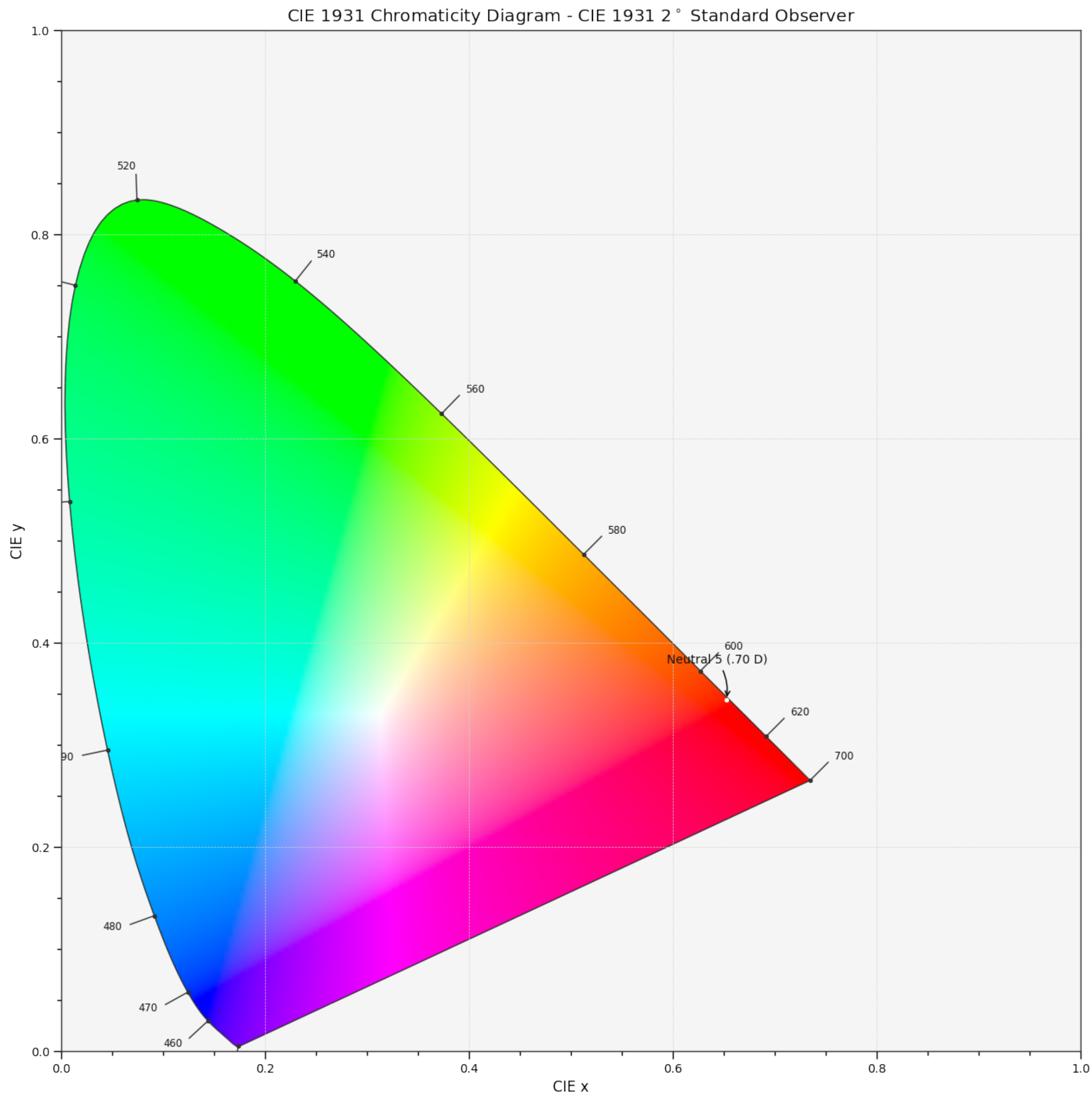
Given a spectral distribution, chromaticity coordinates *CIE xy* can be computed using the `colour.XYZ_to_xy` definition:

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

```
[ 0.31259787  0.32870029]
```

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

```
import matplotlib.pyplot as plt

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

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

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

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



See More

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

3.1.2 How-To

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

3.1.3 Basic Concepts

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

Object Name Categorisation

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

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

```
In [1]: import colour

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

Likewise, for the spectral distribution handling related attributes:

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

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

```
In [2]: colour.models.RGB_COLOURSPACE
                         RGB_COLOURSPACE_ACES2065_1           RGB_COLOURSPACE_ACESPROXY  ␣
↪          RGB_COLOURSPACE_APPLE_RGB             RGB_COLOURSPACE_BT470_525
                         RGB_COLOURSPACE_ACESCC               RGB_COLOURSPACE_ADOBE_
↪RGB1998          RGB_COLOURSPACE_BEST_RGB              RGB_COLOURSPACE_BT470_625
                         RGB_COLOURSPACE_ACESCCT              RGB_COLOURSPACE_ADOBE_WIDE_
↪GAMUT_RGB RGB_COLOURSPACE_BETA_RGB              RGB_COLOURSPACE_BT709              >
                         RGB_COLOURSPACE_ACESCG               RGB_COLOURSPACE_ARRI_WIDE_
↪GAMUT_3     RGB_COLOURSPACE_BT2020               RGB_COLOURSPACE_CIE_RGB
```

Abbreviations

The following abbreviations are in use in Colour:

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

N-Dimensional Array Support

Most of **Colour** definitions are fully vectorised and support n-dimensional array by leveraging Numpy.

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

```
import colour

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

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

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

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

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

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

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

```
import numpy as np

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

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

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

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

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

Which enables image processing:

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

Spectral Representation and Continuous Signal

Floating Point Wavelengths

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

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

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

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

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

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

```
(0.1999999998, 0.2)
```

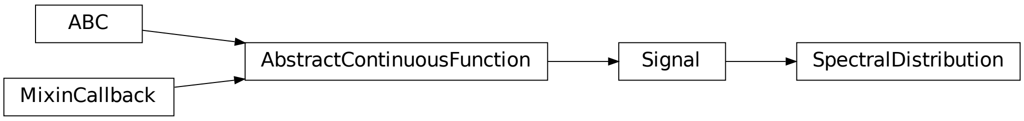
```
apm_2 = ArbitraryPrecisionMapping(data_1, key_decimals=7)
tuple(apm_2.keys())
```

```
(0.2,)
```

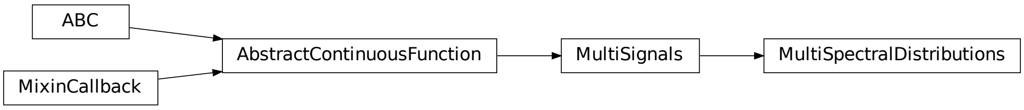
While functional, the approach was brittle and not elegant which triggered a significant amount of rework.

Continuous Signal

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



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



The `colour.continuous.Signal` class implements the `Signal.function()` method so that evaluating the function for any independent domain $x \in \mathbb{R}$ variable returns a corresponding range $y \in \mathbb{R}$ variable.

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

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

```
data = {
    500: 0.0651,
    520: 0.0705,
    540: 0.0772,
    560: 0.0870,
    580: 0.1128,
    600: 0.1360,
}
sd = colour.SpectralDistribution(data)
sd[555.5]
```

```
0.083453673782958995
```

Getting, Setting, Indexing and Slicing

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

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

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

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

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

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

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

```
0.083135180664062502,
```

Multiple wavelength values can be retrieved as follows:

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

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

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

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

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

```
sd[:]
```

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

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

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

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

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

```
0.065100000000000019
```

Instead, the values can be set indirectly:

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

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

Domain-Range Scales

Note: This section contains important information.

Colour adopts 4 main input domains and output ranges:

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

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

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

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

Scale - Reference

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

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

Scale - 1

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

- *Scalars* in domain-range *[0, 10]*, e.g *Munsell Value* are scaled by *10*.
- *Percentages* in domain-range *[0, 100]* are scaled by *100*.
- *Degrees* in domain-range *[0, 360]* are scaled by *360*.
- *Integers* in domain-range *[0, 2**n -1]* where *n* is the bit depth are scaled by *2**n -1*.
- *Dimensionless* values are unaffected and are indicated with *DN*.
- *Unaffected* values are unaffected and are indicated with *UN*.

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

Understanding the Domain-Range Scale of an Object

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

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

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

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

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

Working with the Domain-Range Scales

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

```
import colour

colour.get_domain_range_scale()
```

```
u'reference'
```

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

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

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

```
colour.set_domain_range_scale("1")

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

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

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

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

```
colour.set_domain_range_scale("Reference")

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

```
File "<ipython-input-...>", line 4, in <module>
  E_o2)
File "/colour-science/colour/colour/adaptation/cie1994.py", line 134, in chromatic_
↪adaptation_CIE1994
  warning(('"Y_o" luminance factor must be in [18, 100] domain, '
/colour-science/colour/colour/utilities/verbose.py:207: ColourWarning: "Y_o" luminance␣
↪factor must be in [18, 100] domain, unpredictable results may occur!
  warn(*args, **kwargs)
array([ 0.17171825,  0.13731098,  0.09972054])
```

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

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

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

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

```
[ 0.24033795  0.21156212  0.17643012]
```

Multiprocessing on Windows with Domain-Range Scales

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

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

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

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

Safe Power and Division

Colour default handling of fractional power and zero-division occurring during practical applications is managed via varous definitions and context managers.

Safe Power

NaNs generation occurs when a negative number a is raised to the fractional power p. This can be avoided using the colour.algebra.spow() definition that raises to the power as follows: $sign(a) * |a|^p$.

To the extent possible, the colour.algebra.spow() definition has been used throughout the codebase. The default behaviour is controlled with the following definitions:

- colour.algebra.is_spow_enabled()
- colour.algebra.set_spow_enabled()
- colour.algebra.spow_enable() (Context Manager & Decorator)

Safe Division

NaNs and +/- infs generation occurs when a number a is divided 0. This can be avoided using the `colour.algebra.sdiv()` definition. It has been used wherever deemed relevant in the codebase. The default behaviour is controlled with the following definitions:

- `colour.algebra.get_sdiv_mode()`
- `colour.algebra.set_sdiv_mode()`
- `colour.algebra.sdiv_mode()` (Context Manager & Decorator)

The following modes are available:

- `Numpy`: The current *Numpy* zero-division handling occurs.
- `Ignore`: Zero-division occurs silently.
- `Warning`: Zero-division occurs with a warning.
- `Ignore Zero Conversion`: Zero-division occurs silently and NaNs or +/- infs values are converted to zeros. See `numpy.nan_to_num()` definition for more details.
- `Warning Zero Conversion`: Zero-division occurs with a warning and NaNs or +/- infs values are converted to zeros. See `numpy.nan_to_num()` definition for more details.
- `Ignore Limit Conversion`: Zero-division occurs silently and NaNs or +/- infs values are converted to zeros or the largest +/- finite floating point values representable by the division result `numpy.dtype`. See `numpy.nan_to_num()` definition for more details.
- `Warning Limit Conversion`: Zero-division occurs with a warning and NaNs or +/- infs values are converted to zeros or the largest +/- finite floating point values representable by the division result `numpy.dtype`.

```
colour.algebra.get_sdiv_mode()
```

```
'Ignore Zero Conversion'
```

```
colour.algebra.set_sdiv_mode("Numpy")
colour.UCS_to_uv([0, 0, 0])
```

```
/Users/kelsolaar/Documents/Development/colour-science/colour/colour/algebra/common.
↪py:317: RuntimeWarning: invalid value encountered in true_divide
  c = a / b
array([ nan,  nan])
```

```
colour.algebra.set_sdiv_mode("Ignore Zero Conversion")
colour.UCS_to_uv([0, 0, 0])
```

```
array([ 0.,  0.])
```

3.1.4 Advanced Concepts

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

Environment

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

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

Caching

Colour uses various internal caches to improve speed and prevent redundant processes, notably for spectral related computations.

The internal caches are managed with the *colour.utilities.CACHE_REGISTRY* cache registry object:

```
import colour

print(colour.utilities.CACHE_REGISTRY)
```

```
{'colour.colorimetry.spectrum._CACHE_RESHAPED_SDS_AND_MSDS': '0 item(s)',
 'colour.colorimetry.tristimulus_values._CACHE_LAGRANGE_INTERPOLATING_COEFFICIENTS': '0 '

↪'item(s)',
 'colour.colorimetry.tristimulus_values._CACHE_SD_TO_XYZ': '0 item(s)',
 'colour.colorimetry.tristimulus_values._CACHE_TRISTIMULUS_WEIGHTING_FACTORS': '0 '
                                                                               'item(s)',
 'colour.quality.cfi2017._CACHE_TCS_CIE2017': '0 item(s)',
 'colour.volume.macadam_limits._CACHE_OPTIMAL_COLOUR_STIMULI_XYZ': '0 item(s)',
 'colour.volume.macadam_limits._CACHE_OPTIMAL_COLOUR_STIMULI_XYZ_TRIANGULATIONS': '0 '
                                                                                  'item(s)
↪',
 'colour.volume.spectrum._CACHE_OUTER_SURFACE_XYZ': '0 item(s)',
 'colour.volume.spectrum._CACHE_OUTER_SURFACE_XYZ_POINTS': '0 item(s)'}
```

See *colour.utilities.CacheRegistry* class documentation for more information on how to manage the cache registry.

Using Colour without Scipy

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

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

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

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

```
import sys
from mock_for_colour import MockModule

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

import colour

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

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

Or directly using the `mock_scipy_for_colour` definition:

```
from mock_for_colour import mock_scipy_for_colour

mock_scipy_for_colour()

import colour

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

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

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

3.1.5 Bibliography

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]
- [CIET13805c]
- [Dji17]
- [FiLMiCInc17]
- [Han03]
- [Hou15]
- [Laurent12]
- [Mac35]
- [Mac42]
- [MorovivcL00]
- [MunsellCScienceb]
- [Poi80]
- [RenewableRDCenter03]
- [SWD05]
- [Sir18]
- [SHF00]
- [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee20]
- [WEydelbergVileshin02]
- [War16]
- [WS00b]
- [WS00h]
- [WS00g]
- [WS00e]

CHAPTER

FOUR

5 API REFERENCE

4.1 API Reference

4.1.1 Colour

Chromatic Adaptation

Chromatic Adaptation

colour

chromatic_adaptation(XYZ, XYZ_w, XYZ_wr[, ...])	Adapt given stimulus from test viewing conditions to reference viewing conditions.
CHROMATIC_ADAPTATION_METHODS	Supported chromatic adaptation methods.
VIEWING_CONDITIONS_CMCCAT2000	Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

colour.chromatic_adaptation

colour.**chromatic_adaptation**(*XYZ: ArrayLike, XYZ_w: ArrayLike, XYZ_wr: ArrayLike, method: Union[Literal['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Zhai 2018', 'Von Kries'], str] = 'Von Kries', **kwargs: Any*) → NDArrayFloat

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

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of stimulus to adapt.
- **XYZ_w** (ArrayLike) – Test viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **XYZ_wr** (ArrayLike) – Reference viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **method** (Union[Literal['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Zhai 2018', 'Von Kries'], str]) – Computation method.
- **E_o1** – {colour.adaptation.chromatic_adaptation_CIE1994()}, Test illuminance E_{o1} in cd/m^2.
- **E_o2** – {colour.adaptation.chromatic_adaptation_CIE1994()}, Reference illuminance E_{o2} in cd/m^2.
- **n** – {colour.adaptation.chromatic_adaptation_CIE1994()}, Noise component in fundamental primary system.

- **Y_o** – {`colour.adaptation.chromatic_adaptation_CIE1994()`}, Luminance factor Y_o of achromatic background normalised to domain [0.18, 1] in **'Reference'** domain-range scale.
- **direction** – {`colour.adaptation.chromatic_adaptation_CMCCAT2000()`}, Chromatic adaptation direction.
- **L_A1** – {`colour.adaptation.chromatic_adaptation_CMCCAT2000()`}, Luminance of test adapting field L_{A1} in cd/m^2.
- **L_A2** – {`colour.adaptation.chromatic_adaptation_CMCCAT2000()`}, Luminance of reference adapting field L_{A2} in cd/m^2.
- **surround** – {`colour.adaptation.chromatic_adaptation_CMCCAT2000()`}, Surround viewing conditions induction factors.
- **discount_illuminant** – {`colour.adaptation.chromatic_adaptation_Fairchild1990()`}, Truth value indicating if the illuminant should be discounted.
- **Y_n** – {`colour.adaptation.chromatic_adaptation_Fairchild1990()`}, Luminance Y_n of test adapting stimulus in cd/m^2.
- **D_b** – {`colour.adaptation.chromatic_adaptation_Zhai2018()`}, Degree of adaptation D_β of input illuminant β.
- **D_d** – {`colour.adaptation.chromatic_adaptation_Zhai2018()`}, Degree of adaptation D_Δ of output illuminant Δ.
- **transform** – {`colour.adaptation.chromatic_adaptation_VonKries()`, `colour.adaptation.chromatic_adaptation_Zhai2018()`}, Chromatic adaptation transform.
- **XYZ_wo** – {`colour.adaptation.chromatic_adaptation_Zhai2018()`}, Baseline illuminant (BI) o.
- **kwargs** (`Any`) –

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

Return type `numpy.ndarray`

Notes

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

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

References

[CIET13294], [Fai91], [Fai13f], [Fai13a], [LLRH02], [WRC12d]

Examples

Von Kries chromatic adaptation:

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

CIE 1994 chromatic adaptation, requires extra *kwargs*:

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

CMCCAT2000 chromatic adaptation, requires extra *kwargs*:

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

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

```
>>> XYZ = np.array([0.1953, 0.2307, 0.2497])
>>> Y_n = 200
>>> chromatic_adaptation(
...     XYZ, XYZ_w, XYZ_wr, method="Fairchild 1990", Y_n=Y_n
... )
...
array([ 0.2332526...,  0.2332455...,  0.7611593...])
```

Zhai and Luo (2018) chromatic adaptation:

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> chromatic_adaptation(XYZ, XYZ_w, XYZ_wr, method="Zhai 2018")
...
```

(continues on next page)

(continued from previous page)

```
array([ 0.2163881...,  0.1257    ,  0.0384749...])
>>> chromatic_adaptation(
...     XYZ,
...     XYZ_w,
...     XYZ_wr,
...     method="Zhai 2018",
...     D_b=0.9,
...     XYZ_wo=np.array([100, 100, 100]),
... )
...
array([ 0.2152436...,  0.1253522...,  0.0388406...])
```

colour.CHROMATIC_ADAPTATION_METHODS

colour.**CHROMATIC_ADAPTATION_METHODS = CanonicalMapping({'CIE 1994': ..., 'CMCCAT2000': ..., 'Fairchild 1990': ..., 'Von Kries': ..., 'Zhai 2018': ...})**

Supported chromatic adaptation methods.

References

[CIET13294], [Fai91], [Fai13f], [Fai13a], [LLRH02], [WRC12d], [ZL18]

colour.VIEWING_CONDITIONS_CMCCAT2000

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

Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

[LLRH02], [WRC12d]

Dataset

colour

CHROMATIC_ADAPTATION_TRANSFORMS	Chromatic adaptation transforms.

colour.CHROMATIC_ADAPTATION_TRANSFORMS

colour.**CHROMATIC_ADAPTATION_TRANSFORMS = CanonicalMapping({'XYZ Scaling': ..., 'Von Kries': ..., 'Bradford': ..., 'Sharp': ..., 'Fairchild': ..., 'CMCCAT97': ..., 'CMCCAT2000': ..., 'CAT02': ..., 'CAT02 Brill 2008': ..., 'CAT16': ..., 'Bianco 2010': ..., 'Bianco PC 2010': ...})**

Chromatic adaptation transforms.

References

[BS10], [BS08], [Fai], [LPLMartinezverdu07], [LLW+17], [Lin09a], [WRC12e], [WRC12d], [Wikipedia07c]

Fairchild (1990)

`colour.adaptation`

`chromatic_adaptation_Fairchild1990`(XYZ_1, ...)	Adapt 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: ArrayLike*, *XYZ_n: ArrayLike*, *XYZ_r: ArrayLike*, *Y_n: ArrayLike*, *discount_illuminant: bool = False*) → NDArrayFloat

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

Parameters

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

Returns Adapted *CIE XYZ_2* tristimulus values of stimulus.

Return type `numpy.ndarray`

Notes

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

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

References

[Fai91], [Fai13f]

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, ...)	Adapt 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: ArrayLike*, *xy_o1: ArrayLike*, *xy_o2: ArrayLike*, *Y_o: ArrayLike*, *E_o1: ArrayLike*, *E_o2: ArrayLike*, *n: ArrayLike = 1*) → NDArrayFloat

Adapt given stimulus *CIE XYZ_1* tristimulus values from test viewing conditions to reference viewing conditions using *CIE 1994* chromatic adaptation model.

Parameters

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

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

Return type `numpy.ndarray`

Notes

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

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

References

[CIET13294]

Examples

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

CMCCAT2000

`colour.adaptation`

`chromatic_adaptation_CMCCAT2000`(XYZ, XYZ_w, ...)	Adapt given stimulus *CIE XYZ* tristimulus values using given viewing conditions.
`VIEWING_CONDITIONS_CMCCAT2000`	Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

colour.adaptation.chromatic_adaptation_CMCCAT2000

`colour.adaptation.`**`chromatic_adaptation_CMCCAT2000`**(*XYZ: ArrayLike, XYZ_w: ArrayLike, XYZ_wr: ArrayLike, L_A1: ArrayLike, L_A2: ArrayLike, surround:* colour.adaptation.cmccat2000.InductionFactors_CMCCAT2000 *= VIEWING_CONDITIONS_CMCCAT2000['Average'], direction: Union[Literal['Forward', 'Inverse'], str] = 'Forward'*) → NDArrayFloat

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

This definition is a convenient wrapper around `colour.adaptation.chromatic_adaptation_forward_CMCCAT2000()` and `colour.adaptation.chromatic_adaptation_inverse_CMCCAT2000()`.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (ArrayLike) – Source viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **XYZ_wr** (ArrayLike) – Target viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **L_A1** (ArrayLike) – Luminance of test adapting field L_{A1} in cd/m^2.
- **L_A2** (ArrayLike) – Luminance of reference adapting field L_{A2} in cd/m^2.
- **surround** (colour.adaptation.cmccat2000.InductionFactors_CMCCAT2000) – Surround viewing conditions induction factors.
- **direction** (Union[Literal['Forward', 'Inverse'], str]) – Chromatic adaptation direction.

Returns Adapted stimulus *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

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

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

References

[LLRH02], [WRC12d]

Examples

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

Using the *CMCCAT2000* inverse model:

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

(continues on next page)

(continued from previous page)

```
...     XYZ, XYZ_w, XYZ_wr, L_A1, L_A2, direction="Inverse"
... )
...
array([ 22.48,  22.74,   8.54])
```

colour.adaptation.VIEWING_CONDITIONS_CMCCAT2000

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

Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

[LLRH02], [WRC12d]

Ancillary Objects

colour.adaptation

chromatic_adaptation_forward_CMCCAT2000(XYZ, ...)	Adapt given stimulus *CIE XYZ* tristimulus values from test viewing conditions to reference viewing conditions using *CMCCAT2000* forward chromatic adaptation model.
chromatic_adaptation_inverse_CMCCAT2000(...)	Adapt given stimulus corresponding colour *CIE XYZ* tristimulus values from reference viewing conditions to test viewing conditions using *CMCCAT2000* inverse chromatic adaptation model.
InductionFactors_CMCCAT2000(F)	*CMCCAT2000* chromatic adaptation model induction factors.

colour.adaptation.chromatic_adaptation_forward_CMCCAT2000

colour.adaptation.**chromatic_adaptation_forward_CMCCAT2000**(*XYZ: ArrayLike, XYZ_w: ArrayLike, XYZ_wr: ArrayLike, L_A1: ArrayLike, L_A2: ArrayLike, surround: colour.adaptation.cmccat2000.InductionFactors_CMC = VIEWING_CONDITIONS_CMCCAT2000['Average']*) → NDArrayFloat

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

Parameters

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

- **surround** (colour.adaptation.cmccat2000.InductionFactors_CMCCAT2000) – Surround viewing conditions induction factors.

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

Return type numpy.ndarray

Notes

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

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

References

[LLRH02], [WRC12d]

Examples

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

colour.adaptation.chromatic_adaptation_inverse_CMCCAT2000

colour.adaptation.**chromatic_adaptation_inverse_CMCCAT2000**(*XYZ_c: ArrayLike, XYZ_w: ArrayLike, XYZ_wr: ArrayLike, L_A1: ArrayLike, L_A2: ArrayLike, surround:* colour.adaptation.cmccat2000.InductionFactors_CMC *= VIEWING_CONDITIONS_CMCCAT2000['Average']*) → NDArrayFloat

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

Parameters

- **XYZ_c** (ArrayLike) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (ArrayLike) – Test viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **XYZ_wr** (ArrayLike) – Reference viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **L_A1** (ArrayLike) – Luminance of test adapting field L_{A1} in cd/m^2.

- **L_A2** (ArrayLike) – Luminance of reference adapting field L_{A2} in cd/m^2.
- **surround** (colour.adaptation.cmccat2000.InductionFactors_CMCCAT2000) – Surround viewing conditions induction factors.

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

Return type numpy.ndarray

Notes

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

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

References

[LLRH02], [WRC12d]

Examples

```
>>> XYZ_c = np.array([19.53, 23.07, 24.97])
>>> XYZ_w = np.array([111.15, 100.00, 35.20])
>>> XYZ_wr = np.array([94.81, 100.00, 107.30])
>>> L_A1 = 200
>>> L_A2 = 200
>>> chromatic_adaptation_inverse_CMCCAT2000(
...     XYZ_c, XYZ_w, XYZ_wr, L_A1, L_A2
... )
...
array([ 22.4839876...,  22.7419485...,   8.5393392...])
```

colour.adaptation.InductionFactors_CMCCAT2000

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

CMCCAT2000 chromatic adaptation model induction factors.

Parameters **F** (float) – F surround condition.

References

[LLRH02], [WRC12d]

Create new instance of InductionFactors_CMCCAT2000(F,)

__init__()

Methods

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

Attributes

F	Alias for field number 0

Von Kries

colour.adaptation

chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr)	Adapt given stimulus from test viewing conditions to reference viewing conditions.
CHROMATIC_ADAPTATION_TRANSFORMS	Chromatic adaptation transforms.

colour.adaptation.chromatic_adaptation_VonKries

colour.adaptation.**chromatic_adaptation_VonKries**(*XYZ: ArrayLike, XYZ_w: ArrayLike, XYZ_wr: ArrayLike, transform: Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str] = 'CAT02'*) → NDArrayFloat

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

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of stimulus to adapt.
- **XYZ_w** (ArrayLike) – Test viewing conditions *CIE XYZ* tristimulus values of whitepoint.
- **XYZ_wr** (ArrayLike) – Reference viewing conditions *CIE XYZ* tristimulus values of whitepoint.
- **transform** (Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]) – Chromatic adaptation transform.

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

Return type numpy.ndarray

Notes

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

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

References

[Fai13a]

Examples

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

Using Bradford method:

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

colour.adaptation.CHROMATIC_ADAPTATION_TRANSFORMS

`colour.adaptation.CHROMATIC_ADAPTATION_TRANSFORMS = CanonicalMapping({'XYZ Scaling': ..., 'Von Kries': ..., 'Bradford': ..., 'Sharp': ..., 'Fairchild': ..., 'CMCCAT97': ..., 'CMCCAT2000': ..., 'CAT02': ..., 'CAT02 Brill 2008': ..., 'CAT16': ..., 'Bianco 2010': ..., 'Bianco PC 2010': ...})`

Chromatic adaptation transforms.

References

[BS10], [BS08], [Fai], [LPLMartinezverdu07], [LLW+17], [Lin09a], [WRC12e], [WRC12d], [Wikipedia07c]

Dataset

`colour.adaptation`

CAT_BRADFORD	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_BIANCO2010	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_PC_BIANCO2010	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_CAT02_BRILL2008	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_CAT02	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_CAT16	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_CMCCAT2000	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_CMCCAT97	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_FAIRCHILD	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_SHARP	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_VON_KRIES	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_XYZ_SCALING	ndarray(shape, dtype=float, buffer=None, offset=0,

colour.adaptation.CAT_BRADFORD

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_BIANCO2010

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_PC_BIANCO2010

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_CAT02_BRILL2008

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_CAT02

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_CAT16

colour.adaptation.**CAT_CAT16 = array([[0.401288, 0.650173, -0.051461], [-0.250268, 1.204414, 0.045854], [-0.002079, 0.048952, 0.953127]])**

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_CMCCAT2000

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type* <*numpy.dtype.type*>.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_CMCCAT97

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_FAIRCHILD

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_SHARP

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_VON_KRIES

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type* <*numpy.dtype.type*>.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

colour.adaptation.CAT_XYZ_SCALING

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

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

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

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

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

Parameters

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

Attributes

T [ndarray] Transpose of the array.

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

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

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

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

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

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

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

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

shape [tuple of ints] Shape of the array.

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

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

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

See also:

`array` Construct an array.

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

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

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

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

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

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

Examples

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

First mode, *buffer* is None:

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

Second mode:

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

Ancillary Objects

colour.adaptation

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

colour.adaptation.matrix_chromatic_adaptation_VonKries

colour.adaptation.**matrix_chromatic_adaptation_VonKries**(*XYZ_w: ArrayLike, XYZ_wr: ArrayLike, transform: Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str] = 'CAT02'*) → NDArrayFloat

Compute the *chromatic adaptation* matrix from test viewing conditions to reference viewing conditions.

Parameters

- **XYZ_w** (ArrayLike) – Test viewing conditions *CIE XYZ* tristimulus values of whitepoint.
- **XYZ_wr** (ArrayLike) – Reference viewing conditions *CIE XYZ* tristimulus values of whitepoint.
- **transform** (Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]) – Chromatic adaptation transform.

Returns Chromatic adaptation matrix M_{cat}.

Return type numpy.ndarray

Notes

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

References

[Fai13a]

Examples

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

Using Bradford method:

```
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> method = "Bradford"
>>> matrix_chromatic_adaptation_VonKries(XYZ_w, XYZ_wr, method)
...
array([[ 1.0479297...,  0.0229468..., -0.0501922...],
       [ 0.0296278...,  0.9904344..., -0.0170738...],
       [-0.0092430...,  0.0150551...,  0.7518742...]])
```

Zhai and Luo (2018)

`colour.adaptation`

`chromatic_adaptation_Zhai2018`(XYZ_b, XYZ_wb, ...)	Adapt given sample colour XYZ_β tristimulus values from input viewing conditions under β illuminant to output viewing conditions under δ illuminant using *Zhai and Luo (2018)* chromatic adaptation model.

colour.adaptation.chromatic_adaptation_Zhai2018

colour.adaptation.**chromatic_adaptation_Zhai2018**(*XYZ_b: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], XYZ_wb: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], XYZ_wd: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], D_b: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]] = 1, D_d: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]] = 1, XYZ_wo: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]] = np.array([1, 1, 1]), transform: Union[Literal['CAT02', 'CAT16'], str] = 'CAT02'*) → numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]

Adapt given sample colour XYZ_{β} tristimulus values from input viewing conditions under β illuminant to output viewing conditions under δ illuminant using *Zhai and Luo (2018)* chromatic adaptation model.

According to the definition of D, a one-step CAT such as CAT02 can only be used to transform colors from an incomplete adapted field into a complete adapted field. When CAT02 are used to transform an incomplete to incomplete case, D has no baseline level to refer to. *Smet et al. (2017)* proposed a new concept of two-step CAT to replace the present CATs such as CAT02 with only one-step transform in order to define D more clearly. A two-step CAT involves an illuminant representing the baseline states between the test and reference illuminants for the calculation. In the first step the test color is transformed from test illuminant to the baseline illuminant (BI), and it is then transformed to the reference illuminant Degrees of adaptation under the other illuminants should be calculated relative to the adaptation under the BI. When D becomes lower towards zero, the adaptation point of the observer moves towards the BI. Therefore, the chromaticity of the BI should be an intrinsic property of the human vision system.

Parameters

- **XYZ_b** (`Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]`) – Sample colour XYZ_{β} under input illuminant β.

- **XYZ_wb** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Input illuminant β.
- **XYZ_wd** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Output illuminant δ.
- **D_b** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Degree of adaptation D_β of input illuminant β.
- **D_d** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Degree of adaptation D_δ of output illuminant δ.
- **XYZ_wo** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Baseline illuminant (BI) o.
- **transform** (Union[Literal['CAT02', 'CAT16'], str]) – Chromatic adaptation transform.

Returns Sample corresponding colour XYZ_δ tristimulus values under output illuminant D_δ.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ_b	[0, 1]	[0, 1]
XYZ_wb	[0, 1]	[0, 1]
XYZ_wd	[0, 1]	[0, 1]
XYZ_wo	[0, 1]	[0, 1]

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

References

[ZL18]

Examples

```
>>> XYZ_b = np.array([48.900, 43.620, 6.250])
>>> XYZ_wb = np.array([109.850, 100, 35.585])
>>> XYZ_wd = np.array([95.047, 100, 108.883])
>>> D_b = 0.9407
>>> D_d = 0.9800
>>> XYZ_wo = np.array([100, 100, 100])
>>> chromatic_adaptation_Zhai2018(
...     XYZ_b, XYZ_wb, XYZ_wd, D_b, D_d, XYZ_wo
... )
array([ 39.1856164...,  42.1546179...,  19.2367203...])
>>> XYZ_d = np.array([39.18561644, 42.15461798, 19.23672036])
>>> chromatic_adaptation_Zhai2018(
...     XYZ_d, XYZ_wd, XYZ_wb, D_d, D_b, XYZ_wo
... )
array([ 48.9 ,  43.62,   6.25])
```

Algebra

Extrapolation

`colour`

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

colour.Extrapolator

class colour.**Extrapolator**(*interpolator: ProtocolInterpolator | None = None, method: Literal['Linear', 'Constant'] | str = 'Linear', left: Real | None = None, right: Real | None = None, dtype: Type[DTypeReal] | None = None, *args: Any, **kwargs: Any*)

Bases: `object`

Extrapolate 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** (`ProtocolInterpolator | None`) – Interpolator object.
- **method** (`Literal['Linear', 'Constant'] | str`) – Extrapolation method.

- **left** (Real | None) – Value to return for x < xi[0].
- **right** (Real | None) – Value to return for x > xi[-1].
- **dtype** (Type[DTypeReal] | None) – Data type used for internal conversions.
- **args** (Any) –
- **kwargs** (Any) –

Return type None

Methods

- `__init__()`
- `__class__()`

Notes

- The interpolator must define x and y properties.

References

[sastanin], [WRC12b]

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 *ArrayLike* 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)
```

(continues on next page)

(continued from previous page)

```
>>> extrapolator(np.array([0.1, 0.2, 8, 9]))
array([ 0.,  0.,  3.,  3.])
```

__init__(*interpolator: ProtocolInterpolator | None = None, method: Literal['Linear', 'Constant'] | str = 'Linear', left: Real | None = None, right: Real | None = None, dtype: Type[DTypeReal] | None = None, *args: Any, **kwargs: Any*) → None

Parameters

- **interpolator** (ProtocolInterpolator | None) –
- **method** (Literal['Linear', 'Constant'] | str) –
- **left** (Real | None) –
- **right** (Real | None) –
- **dtype** (Type[DTypeReal] | None) –
- **args** (Any) –
- **kwargs** (Any) –

Return type None

property interpolator: colour.hints.ProtocolInterpolator

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

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

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

Return type *ProtocolInterpolator*

__weakref__

list of weak references to the object (if defined)

property method: Union[Literal['Linear', 'Constant'], str]

Getter and setter property for the extrapolation method.

Parameters **value** – Value to set the extrapolation method. with.

Returns Extrapolation method.

Return type str

property left: Real | None

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

Parameters **value** – Left value to return for x < xi[0].

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

Return type None or Real

property right: Real | None

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

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

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

Return type None or Real

__call__(*x: ArrayLike*) → NDArrayFloat

Evaluate the Extrapolator at given point(s).

Parameters **x** (ArrayLike) – Point(s) to evaluate the Extrapolator at.

Returns Extrapolated points value(s).

Return type numpy.ndarray

Interpolation

colour

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

colour.KernelInterpolator

class colour.**KernelInterpolator**(*x: ArrayLike, y: ArrayLike, window: float = 3, kernel: Callable = kernel_lanczos, kernel_kwargs: dict | None = None, padding_kwargs: dict | None = None, dtype: Type[DTypeReal] | None = None, *args: Any, **kwargs: Any*)

Bases: object

Kernel based interpolation of a 1-D function.

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

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

Parameters

- **x** (ArrayLike) – Independent x variable values corresponding with y variable.
- **y** (ArrayLike) – Dependent and already known y variable values to interpolate.
- **window** (float) – Width of the window in samples on each side.
- **kernel** (Callable) – Kernel to use for interpolation.
- **kernel_kwargs** (dict | None) – Arguments to use when calling the kernel.
- **padding_kwargs** (dict | None) – Arguments to use when padding y variable values with the np.pad() definition.
- **dtype** (Type[DTypeReal] | None) – Data type used for internal conversions.
- **args** (Any) –
- **kwargs** (Any) –

Return type None

Attributes

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

Methods

- __init__()
- __call__()

References

[BB09], [Wikipedia05a]

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 *ArrayLike* variable:

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

Using a different *lanczos* kernel:

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

Using a different window size:

```
>>> f = KernelInterpolator(
...     x, y, window=16, kernel=kernel_lanczos, kernel_kwargs={"a": 16}
... )
>>> f([0.25, 0.75])
array([ 5.3961792...,  5.6521093...])
```

__init__(*x: ArrayLike*, *y: ArrayLike*, *window: float = 3*, *kernel: Callable = kernel_lanczos*, *kernel_kwargs: dict | None = None*, *padding_kwargs: dict | None = None*, *dtype: Type[DTypeReal] | None = None*, **args: Any*, ***kwargs: Any*) → None

Parameters

- **x** (ArrayLike) –
- **y** (ArrayLike) –
- **window** (float) –
- **kernel** (Callable) –
- **kernel_kwargs** (dict | None) –
- **padding_kwargs** (dict | None) –
- **dtype** (Type[DTypeReal] | None) –
- **args** (Any) –
- **kwargs** (Any) –

Return type None

property x: NDArrayFloat

Getter and setter property for the independent x variable.

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

Returns Independent x variable.

Return type numpy.ndarray

property y: NDArrayFloat

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

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

Returns Dependent and already known y variable.

Return type numpy.ndarray

property window: float

Getter and setter property for the window.

Parameters **value** – Value to set the window with.

Returns Window.

Return type float

property kernel: Callable

Getter and setter property for the kernel callable.

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

Returns Kernel callable.

Return type Callable

property kernel_kwargs: dict

Getter and setter property for the kernel call time arguments.

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

Returns Kernel call time arguments.

Return type dict

property padding_kwargs: dict

Getter and setter property for the kernel call time arguments.

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

Returns Kernel call time arguments.

Return type dict

__call__(*x: ArrayLike*) → NDArrayFloat

Evaluate the interpolator at given point(s).

Parameters **x** (ArrayLike) – Point(s) to evaluate the interpolant at.

Returns Interpolated value(s).

Return type numpy.ndarray

__weakref__

list of weak references to the object (if defined)

colour.NearestNeighbourInterpolator

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

Bases: colour.algebra.interpolation.KernelInterpolator

A nearest-neighbour interpolator.

Parameters

- **dtype** – Data type used for internal conversions.
- **padding_kwargs** – Arguments to use when padding y variable values with the np.pad() definition.
- **window** – Width of the window in samples on each side.
- **x** – Independent x variable values corresponding with y variable.
- **y** – Dependent and already known y variable values to interpolate.
- **args** (Any) –
- **kwargs** (Any) –

Return type None

Methods

- __init__()

__init__(**args: Any*, ***kwargs: Any*) → None

Parameters

- **args** (Any) –
- **kwargs** (Any) –

Return type None

colour.LinearInterpolator

class colour.**LinearInterpolator**(*x: ArrayLike*, *y: ArrayLike*, *dtype: Type[DTypeReal] | None = None*, **args: Any*, ***kwargs: Any*)

Bases: object

Interpolate linearly a 1-D function.

Parameters

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

- **dtype** (Type[DTypeReal] | None) – Data type used for internal conversions.
- **args** (Any) –
- **kwargs** (Any) –

Return type None

Attributes

- x
- y

Methods

- __init__()
- __call__()

Notes

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

Examples

Interpolating a single numeric variable:

```
>>> y = np.array(
...     [5.9200, 9.3700, 10.8135, 4.5100, 69.5900, 27.8007, 86.0500]
... )
>>> x = np.arange(len(y))
>>> f = LinearInterpolator(x, y)
>>> f(0.5)
7.64...
```

Interpolating an *ArrayLike* variable:

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

__init__(*x: ArrayLike*, *y: ArrayLike*, *dtype: Type[DTypeReal]* | *None* = *None*, **args: Any*, ***kwargs: Any*) → None

Parameters

- **x** (ArrayLike) –
- **y** (ArrayLike) –
- **dtype** (Type[DTypeReal] | None) –
- **args** (Any) –
- **kwargs** (Any) –

Return type None

property x: NDArrayFloat

Getter and setter property for the independent x variable.

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

Returns Independent x variable.

Return type numpy.ndarray

property y: NDArrayFloat

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

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

Returns Dependent and already known y variable.

Return type numpy.ndarray

__call__(*x: ArrayLike*) → NDArrayFloat

Evaluate the interpolating polynomial at given point(s).

Parameters **x** (ArrayLike) – Point(s) to evaluate the interpolant at.

Returns Interpolated value(s).

Return type numpy.ndarray

__weakref__

list of weak references to the object (if defined)

colour.NullInterpolator

class colour.**NullInterpolator**(*x: ArrayLike, y: ArrayLike, absolute_tolerance: float = 10e-7, relative_tolerance: float = 10e-7, default: float = np.nan, dtype: Type[DTypeReal] | None = None, *args: Any, **kwargs: Any*)

Bases: object

Perform 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** (ArrayLike) – Independent x variable values corresponding with y variable.
- **y** (ArrayLike) – Dependent and already known y variable values to interpolate.
- **absolute_tolerance** (float) – Absolute tolerance.
- **relative_tolerance** (float) – Relative tolerance.
- **default** (float) – Default value for interpolation outside tolerances.
- **dtype** (Type[DTypeReal] | None) – Data type used for internal conversions.
- **args** (Any) –
- **kwargs** (Any) –

Return type None

Attributes

- x
- y
- relative_tolerance
- absolute_tolerance
- default

Methods

- __init__()
- __call__()

Examples

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

__init__(*x: ArrayLike, y: ArrayLike, absolute_tolerance: float = 10e-7, relative_tolerance: float = 10e-7, default: float = np.nan, dtype: Type[DTypeReal] | None = None, *args: Any, **kwargs: Any*) → None

Parameters

- **x** (ArrayLike) –
- **y** (ArrayLike) –
- **absolute_tolerance** (float) –
- **relative_tolerance** (float) –
- **default** (float) –
- **dtype** (Type[DTypeReal] | None) –
- **args** (Any) –
- **kwargs** (Any) –

Return type None

__weakref__

list of weak references to the object (if defined)

property x: NDArrayFloat

Getter and setter property for the independent x variable.

Parameters value – Value to set the independent x variable with.

Returns Independent x variable.

Return type numpy.ndarray

property y: NDArrayFloat

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

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

Returns Dependent and already known y variable.

Return type numpy.ndarray

property relative_tolerance: float

Getter and setter property for the relative tolerance.

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

Returns Relative tolerance.

Return type float

property absolute_tolerance: float

Getter and setter property for the absolute tolerance.

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

Returns Absolute tolerance.

Return type float

property default: float

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

Parameters **value** – Value to set the default value with.

Returns Default value.

Return type float

__call__(*x: ArrayLike*) → NDArrayFloat

Evaluate the interpolator at given point(s).

Parameters **x** (ArrayLike) – Point(s) to evaluate the interpolant at.

Returns Interpolated value(s).

Return type numpy.ndarray

colour.PchipInterpolator

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

Bases: scipy.interpolate._cubic.PchipInterpolator

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

Attributes

- y

Methods

- `__init__()`

Notes

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

  Parameters

  - **x** (ArrayLike) –
  - **y** (ArrayLike) –
  - **args** (Any) –
  - **kwargs** (Any) –

  Return type None

`__init__`(*x: ArrayLike*, *y: ArrayLike*, **args: Any*, ***kwargs: Any*) → None

Parameters

- **x** (ArrayLike) –
- **y** (ArrayLike) –
- **args** (Any) –
- **kwargs** (Any) –

Return type None

property y: NDArrayFloat

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

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

Returns Dependent and already known y variable.

Return type `numpy.ndarray`

colour.SpragueInterpolator

class colour.**SpragueInterpolator**(*x: ArrayLike*, *y: ArrayLike*, *dtype: Type[DTypeReal] | None = None*, **args: Any*, ***kwargs: Any*)

Bases: `object`

Construct 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** (ArrayLike) – Independent x variable values corresponding with y variable.
- **y** (ArrayLike) – Dependent and already known y variable values to interpolate.
- **dtype** (Type[DTypeReal] | None) – Data type used for internal conversions.

- **args** (Any) –
- **kwargs** (Any) –

Return type None

Attributes

- x
- y

Methods

- `__init__()`
- `__call__()`

Notes

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

References

[CIET13805b], [WRC12c]

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 *ArrayLike* variable:

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

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

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

SPRAGUE_C_COEFFICIENTS, (4, 6)

References

[CIET13805c]

`__init__`(*x: ArrayLike*, *y: ArrayLike*, *dtype: Type[DTypeReal] | None = None*, **args: Any*, ***kwargs: Any*) → None

Parameters

- **x** (`ArrayLike`) –
- **y** (`ArrayLike`) –
- **dtype** (`Type[DTypeReal] | None`) –
- **args** (`Any`) –
- **kwargs** (`Any`) –

Return type None

`property x: NDArrayFloat`

Getter and setter property for the independent x variable.

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

Returns Independent x variable.

Return type `numpy.ndarray`

`property y: NDArrayFloat`

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

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

Returns Dependent and already known y variable.

Return type `numpy.ndarray`

`__call__`(*x: ArrayLike*) → NDArrayFloat

Evaluate the interpolating polynomial at given point(s).

Parameters **x** (`ArrayLike`) – Point(s) to evaluate the interpolant at.

Returns Interpolated value(s).

Return type `numpy.ndarray`

`__weakref__`

list of weak references to the object (if defined)

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

colour.lagrange_coefficients

colour.**lagrange_coefficients**(*r: float*, *n: int = 4*) → NDArrayFloat

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

Parameters

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

Return type numpy.ndarray

References

[Fai85], [Wikipedia03a]

Examples

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

colour.TABLE_INTERPOLATION_METHODS

colour.**TABLE_INTERPOLATION_METHODS = CanonicalMapping({'Trilinear': ..., 'Tetrahedral': ...})**

Supported table interpolation methods.

References

[Boub], [Kir06]

colour.table_interpolation

colour.**table_interpolation**(*V_xyz: ArrayLike*, *table: ArrayLike*, *method: Union[Literal['Trilinear', 'Tetrahedral'], str] = 'Trilinear'*) → NDArrayFloat

Perform interpolation of given V_{xyz} values using given interpolation table.

Parameters

- **V_xyz** (ArrayLike) – V_{xyz} values to interpolate.
- **table** (ArrayLike) – 4-Dimensional (NxNxNx3) interpolation table.
- **method** (Union[Literal['Trilinear', 'Tetrahedral'], str]) – Interpolation method.

Returns Interpolated V_{xyz} values.

Return type numpy.ndarray

References

[Boub], [Kir06]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "..",
...     "io",
...     "luts",
...     "tests",
...     "resources",
...     "iridas_cube",
...     "Colour_Correct.cube",
... )
>>> LUT = colour.read_LUT(path)
>>> table = LUT.table
>>> prng = np.random.RandomState(4)
>>> V_xyz = colour.algebra.random_triplet_generator(3, random_state=prng)
>>> print(V_xyz)
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation(V_xyz, table)
array([[ 1.0120664...,  0.7539146...,  1.0228540...],
       [ 0.5075794...,  0.6479459...,  0.1066404...],
       [ 1.0976519...,  0.1785998...,  0.2299897...]])
>>> table_interpolation(V_xyz, table, method="Tetrahedral")
...
array([[ 1.0196197...,  0.7674062...,  1.0311751...],
       [ 0.5105603...,  0.6466722...,  0.1077296...],
       [ 1.1178206...,  0.1762039...,  0.2209534...]])
```

Interpolation Kernels

`colour`

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

colour.kernel_nearest_neighbour

colour.**kernel_nearest_neighbour**(*x: ArrayLike*) → NDArrayFloat

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

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

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

Return type numpy.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: ArrayLike*) → NDArrayFloat

Return the *linear* kernel evaluated at given samples.

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

Returns The *linear* kernel evaluated at given samples.

Return type numpy.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: ArrayLike*, *a: float = 3*) → NDArrayFloat

Return the *sinc* kernel evaluated at given samples.

Parameters

- **x** (ArrayLike) – Samples at which to evaluate the *sinc* kernel.
- **a** (float) – Size of the *sinc* kernel.

Returns The *sinc* kernel evaluated at given samples.

Return type numpy.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: ArrayLike*, *a: float = 3*) → NDArrayFloat

Return the *lanczos* kernel evaluated at given samples.

Parameters

- **x** (ArrayLike) – Samples at which to evaluate the *lanczos* kernel.
- **a** (float) – Size of the *lanczos* kernel.

Returns The *lanczos* kernel evaluated at given samples.

Return type numpy.ndarray

References

[Wikipedia05a]

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: ArrayLike*, *a: float = 0.5*, *b: float = 0.0*) → NDArrayFloat

Return 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** (ArrayLike) – Samples at which to evaluate the *cardinal spline* kernel.
- **a** (float) – a control parameter.

- **b** (float) – b control parameter.

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

Return type numpy.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.        ])
```

Ancillary Objects

colour.algebra

table_interpolation_trilinear(V_xyz, table)	Perform the trilinear interpolation of given V_{xyz} values using given interpolation table.
table_interpolation_tetrahedral(V_xyz, table)	Perform the tetrahedral interpolation of given V_{xyz} values using given interpolation table.

colour.algebra.table_interpolation_trilinear

colour.algebra.**table_interpolation_trilinear**(*V_xyz: ArrayLike*, *table: ArrayLike*) → NDArrayFloat

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

Parameters

- **V_xyz** (ArrayLike) – V_{xyz} values to interpolate.
- **table** (ArrayLike) – 4-Dimensional (NxNxNx3) interpolation table.

Returns Interpolated V_{xyz} values.

Return type numpy.ndarray

References

[Boub]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "..",
...     "io",
...     "luts",
...     "tests",
...     "resources",
...     "iridas_cube",
```

(continues on next page)

(continued from previous page)

```
...     "Colour_Correct.cube",
... )
>>> LUT = colour.read_LUT(path)
>>> table = LUT.table
>>> prng = np.random.RandomState(4)
>>> V_xyz = colour.algebra.random_triplet_generator(3, random_state=prng)
>>> print(V_xyz)
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation_trilinear(V_xyz, table)
array([[ 1.0120664...,  0.7539146...,  1.0228540...],
       [ 0.5075794...,  0.6479459...,  0.1066404...],
       [ 1.0976519...,  0.1785998...,  0.2299897...]])
```

colour.algebra.table_interpolation_tetrahedral

colour.algebra.**table_interpolation_tetrahedral**(*V_xyz: ArrayLike*, *table: ArrayLike*) → NDArrayFloat

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

Parameters

- **V_xyz** (ArrayLike) – V_{xyz} values to interpolate.
- **table** (ArrayLike) – 4-Dimensional (NxNxNx3) interpolation table.

Returns Interpolated V_{xyz} values.

Return type numpy.ndarray

References

[Kir06]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "..",
...     "io",
...     "luts",
...     "tests",
...     "resources",
...     "iridas_cube",
...     "Colour_Correct.cube",
... )
>>> LUT = colour.read_LUT(path)
>>> table = LUT.table
>>> prng = np.random.RandomState(4)
>>> V_xyz = colour.algebra.random_triplet_generator(3, random_state=prng)
>>> print(V_xyz)
[[ 0.9670298...  0.7148159...  0.9762744...]
```

(continues on next page)

(continued from previous page)

```
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation_tetrahedral(V_xyz, table)
array([[ 1.0196197...,  0.7674062...,  1.0311751...],
       [ 0.5105603...,  0.6466722...,  0.1077296...],
       [ 1.1178206...,  0.1762039...,  0.2209534...]])
```

Coordinates

colour.algebra

cartesian_to_spherical(a)	Transform given cartesian coordinates array xyz to spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth).
spherical_to_cartesian(a)	Transform given spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth) to cartesian coordinates array xyz.
cartesian_to_polar(a)	Transform given cartesian coordinates array xy to polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate).
polar_to_cartesian(a)	Transform given polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate) to cartesian coordinates array xy.
cartesian_to_cylindrical(a)	Transform given cartesian coordinates array xyz to cylindrical coordinates array $\rho\phi z$ (radial distance, azimuth and height).
cylindrical_to_cartesian(a)	Transform given cylindrical coordinates array $\rho\phi z$ (radial distance, azimuth and height) to cartesian coordinates array xyz.

colour.algebra.cartesian_to_spherical

colour.algebra.**cartesian_to_spherical**(*a: ArrayLike*) → NDArrayFloat

Transform given cartesian coordinates array xyz to spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth).

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

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

Return type numpy.ndarray

References

[Wikipedia06a], [Wikipedia05e]

Examples

```
>>> a = np.array([3, 1, 6])
>>> cartesian_to_spherical(a)
array([ 6.7823299...,  0.4850497...,  0.3217505...])
```

colour.algebra.spherical_to_cartesian

colour.algebra.**spherical_to_cartesian**(*a: ArrayLike*) → NDArrayFloat

Transform given spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth) to cartesian coordinates array xyz.

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

Returns Cartesian coordinates array xyz.

Return type numpy.ndarray

References

[Wikipedia06a], [Wikipedia05e]

Examples

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

colour.algebra.cartesian_to_polar

colour.algebra.**cartesian_to_polar**(*a: ArrayLike*) → NDArrayFloat

Transform given cartesian coordinates array xy to polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate).

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

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

Return type numpy.ndarray

References

[Wikipedia06a], [Wikipedia05e]

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: ArrayLike*) → NDArrayFloat

Transform given polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate) to cartesian coordinates array xy.

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

Returns Cartesian coordinates array xy.

Return type numpy.ndarray

References

[Wikipedia06a], [Wikipedia05e]

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: ArrayLike*) → NDArrayFloat

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

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

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

Return type numpy.ndarray

References

[Wikipedia06a], [Wikipedia05e]

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: ArrayLike*) → NDArrayFloat

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

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

Returns Cartesian coordinates array xyz.

Return type numpy.ndarray

References

[Wikipedia06a], [Wikipedia05e]

Examples

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

Random

colour.algebra

random_triplet_generator(size[, limits, ...])	Return a generator yielding random triplets.

colour.algebra.random_triplet_generator

colour.algebra.**random_triplet_generator**(*size: int, limits: ArrayLike = np.array([[0, 1], [0, 1], [0, 1]]), random_state: numpy.random.mtrand.RandomState = RANDOM_STATE*) → NDArrayFloat

Return a generator yielding random triplets.

Parameters

- **size** (int) – Generator size.
- **limits** (ArrayLike) – Random values limits on each triplet axis.

- **random_state** (numpy.random.mtrand.RandomState) – Mersenne Twister pseudo-random number generator.

Returns Random triplet generator.

Return type numpy.ndarray

Notes

- The test is assuming that np.random.RandomState() definition will return the same sequence no matter which *OS* or *Python* version is used. There is however no formal promise about the *prng* sequence reproducibility of either *Python* or *Numpy* implementations, see [Laurent12].

Examples

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

Regression

colour.algebra

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

colour.algebra.least_square_mapping_MoorePenrose

colour.algebra.**least_square_mapping_MoorePenrose**(*y: ArrayLike*, *x: ArrayLike*) → NDArrayFloat

Compute the *least-squares* mapping from dependent variable y to independent variable x using *Moore-Penrose* inverse.

Parameters

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

Returns *Least-squares* mapping.

Return type numpy.ndarray

References

[FMH15]

Examples

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

Common

`colour.algebra`

get_sdiv_mode()	Return *Colour* safe division mode.
set_sdiv_mode(mode)	Set *Colour* safe division function mode.
sdiv_mode([mode])	Define a context manager and decorator temporarily setting *Colour* safe division function mode.
sdiv(a, b)	Divide given array b with array b while handling zero-division.
is_spow_enabled()	Return whether *Colour* safe / symmetrical power function is enabled.
set_spow_enable(enable)	Set *Colour* safe / symmetrical power function enabled state.
spow_enable(enable)	Define a context manager and decorator temporarily setting *Colour* safe / symmetrical power function enabled state.
spow(a, p)	Raise given array a to the power p as follows: $sign(a) * \|a\|^p$.
normalise_vector(a)	Normalise given vector a.
normalise_maximum(a[, axis, factor, clip])	Normalise given array a values by a maximum value and optionally clip them between.
vector_dot(m, v)	Perform the dot product of the matrix array m with the vector array v.
matrix_dot(a, b)	Perform the dot product of the matrix array a with the matrix array b.
euclidean_distance(a, b)	Return the *Euclidean* distance between point array a and point array b.
manhattan_distance(a, b)	Return the *Manhattan* (or *City-Block*) distance between point array a and point array b.
linear_conversion(a, old_range, new_range)	Perform a simple linear conversion of given array a between the old and new ranges.
linstep_function(x[, a, b, clip])	Perform a simple linear interpolation between given array a and array b using x array.
lerp(x[, a, b, clip])	Perform a simple linear interpolation between given array a and array b using x array.
smoothstep_function(x[, a, b, clip])	Evaluate the *smoothstep* sigmoid-like function on array x.
smooth(x[, a, b, clip])	Evaluate the *smoothstep* sigmoid-like function on array x.
is_identity(a)	Return whether a array is an identity matrix.
eigen_decomposition(a[, eigen_w_v_count, ...])	Return the eigen-values w and eigen-vectors v of given array a in given order.

colour.algebra.get_sdiv_mode

colour.algebra.**get_sdiv_mode**() → Literal['Numpy', 'Ignore', 'Warning', 'Raise', 'Ignore Zero Conversion', 'Warning Zero Conversion', 'Ignore Limit Conversion', 'Warning Limit Conversion']

Return *Colour* safe division mode.

Returns *Colour* safe division mode, see colour.algebra.sdiv() definition for an explanation about the possible modes.

Return type str

Examples

```
>>> with sdiv_mode("Numpy"):
...     get_sdiv_mode()
...
'numpy'
>>> with sdiv_mode("Ignore Zero Conversion"):
...     get_sdiv_mode()
...
'ignore zero conversion'
```

colour.algebra.set_sdiv_mode

colour.algebra.**set_sdiv_mode**(*mode: Union[Literal['Numpy', 'Ignore', 'Warning', 'Raise', 'Ignore Zero Conversion', 'Warning Zero Conversion', 'Ignore Limit Conversion', 'Warning Limit Conversion'], str]*)

Set *Colour* safe division function mode.

Parameters mode (Union[Literal['Numpy', 'Ignore', 'Warning', 'Raise', 'Ignore Zero Conversion', 'Warning Zero Conversion', 'Ignore Limit Conversion', 'Warning Limit Conversion'], str]) – *Colour* safe division mode, see colour.algebra.sdiv() definition for an explanation about the possible modes.

Examples

```
>>> with sdiv_mode(get_sdiv_mode()):
...     print(get_sdiv_mode())
...     set_sdiv_mode("Raise")
...     print(get_sdiv_mode())
...
ignore zero conversion
raise
```

colour.algebra.sdiv_mode

class colour.algebra.**sdiv_mode**(*mode: Optional[Literal['Numpy', 'Ignore', 'Warning', 'Raise', 'Ignore Zero Conversion', 'Warning Zero Conversion', 'Ignore Limit Conversion', 'Warning Limit Conversion']] = None*)

Define a context manager and decorator temporarily setting *Colour* safe division function mode.

Parameters mode (Literal['Numpy', 'Ignore', 'Warning', 'Raise', 'Ignore Zero Conversion', 'Warning Zero Conversion', 'Ignore Limit Conversion', 'Warning Limit Conversion'] | None) – *Colour* safe division function mode, see colour.algebra.sdiv() definition for an explanation about the possible modes.

Return type None

__init__(*mode: Optional[Literal['Numpy', 'Ignore', 'Warning', 'Raise', 'Ignore Zero Conversion', 'Warning Zero Conversion', 'Ignore Limit Conversion', 'Warning Limit Conversion']] = None*) → None

Parameters mode (Optional[Literal['Numpy', 'Ignore', 'Warning', 'Raise', 'Ignore Zero Conversion', 'Warning Zero Conversion', 'Ignore Limit Conversion', 'Warning Limit Conversion']]) –

Return type None

Methods

`__init__([mode])`	

colour.algebra.sdiv

`colour.algebra.sdiv(a: ArrayLike, b: ArrayLike) → NDArrayFloat`

Divide given array b with array b while handling zero-division.

This definition avoids NaNs and +/- infs generation when array b is equal to zero. This behaviour can be controlled with the `colour.algebra.set_sdiv_mode()` definition or with the `sdiv_mode()` context manager. The following modes are available:

- `Numpy`: The current *Numpy* zero-division handling occurs.
- `Ignore`: Zero-division occurs silently.
- `Warning`: Zero-division occurs with a warning.
- `Ignore Zero Conversion`: Zero-division occurs silently and NaNs or +/- infs values are converted to zeros. See `numpy.nan_to_num()` definition for more details.
- `Warning Zero Conversion`: Zero-division occurs with a warning and NaNs or +/- infs values are converted to zeros. See `numpy.nan_to_num()` definition for more details.
- `Ignore Limit Conversion`: Zero-division occurs silently and NaNs or +/- infs values are converted to zeros or the largest +/- finite floating point values representable by the division result `numpy.dtype`. See `numpy.nan_to_num()` definition for more details.
- `Warning Limit Conversion`: Zero-division occurs with a warning and NaNs or +/- infs values are converted to zeros or the largest +/- finite floating point values representable by the division result `numpy.dtype`.

Parameters

- **a** (`ArrayLike`) – Numerator array a.
- **b** (`ArrayLike`) – Denominator array b.

Returns Array b safely divided by a.

Return type `np.float` or `numpy.ndarray`

Examples

```
>>> a = np.array([0, 1, 2])
>>> b = np.array([2, 1, 0])
>>> sdiv(a, b)
array([ 0.,  1.,  0.])
>>> try:
...     with sdiv_mode("Raise"):
...         sdiv(a, b)
... except Exception as error:
...     error
...
FloatingPointError('divide by zero encountered in...divide')
>>> with sdiv_mode("Ignore Zero Conversion"):
...     sdiv(a, b)
...
```

(continues on next page)

(continued from previous page)

```
array([ 0.,  1.,  0.])
>>> with sdiv_mode("Warning Zero Conversion"):
...     sdiv(a, b)
...
array([ 0.,  1.,  0.])
>>> with sdiv_mode("Ignore Limit Conversion"):
...     sdiv(a, b)
...
array([  0.00000000e+000,   1.00000000e+000,   1.79769313e+308])
>>> with sdiv_mode("Warning Limit Conversion"):
...     sdiv(a, b)
...
array([  0.00000000e+000,   1.00000000e+000,   1.79769313e+308])
```

colour.algebra.is_spow_enabled

colour.algebra.**is_spow_enabled**() → bool

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

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

Return type bool

Examples

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

colour.algebra.set_spow_enable

colour.algebra.**set_spow_enable**(*enable: bool*)

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

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

Examples

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

colour.algebra.spow_enable

class colour.algebra.**spow_enable**(*enable: bool*)

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

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

Return type None

__init__(*enable: bool*) → None

Parameters **enable** (bool) –

Return type None

Methods

__init__(enable)	

colour.algebra.spow

colour.algebra.**spow**(*a: ArrayLike*, *p: ArrayLike*) → NDArrayFloat

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

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

Parameters

- **a** (ArrayLike) – Array a.
- **p** (ArrayLike) – Power p.

Returns Array a safely raised to the power p.

Return type np.float or numpy.ndarray

Examples

```
>>> np.power(-2, 0.15)
nan
>>> spow(-2, 0.15)
-1.1095694...
>>> spow(0, 0)
0.0
```

colour.algebra.normalise_vector

colour.algebra.**normalise_vector**(*a: ArrayLike*) → NDArrayFloat

Normalise given vector a.

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

Returns Normalised vector a.

Return type numpy.ndarray

Examples

```
>>> a = np.array([0.20654008, 0.12197225, 0.05136952])
>>> normalise_vector(a)
array([ 0.8419703...,  0.4972256...,  0.2094102...])
```

colour.algebra.normalise_maximum

colour.algebra.**normalise_maximum**(*a: ArrayLike*, *axis: int | None = None*, *factor: float = 1*, *clip: bool = True*) → NDArrayFloat

Normalise given array a values by a maximum value and optionally clip them between.

Parameters

- **a** (ArrayLike) – Array a to normalise.
- **axis** (int | None) – Normalization axis.
- **factor** (float) – Normalization factor.
- **clip** (bool) – Clip values to domain [0, 'factor'].

Returns Maximum normalised array a.

Return type numpy.ndarray

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> normalise_maximum(a)
array([ 1.        ,  0.6564384...,  0.4576822...])
```

colour.algebra.vector_dot

colour.algebra.**vector_dot**(*m: ArrayLike*, *v: ArrayLike*) → NDArrayFloat

Perform the dot product of the matrix array m with the vector array v.

This definition is a convenient wrapper around `np.einsum()` with the following subscripts: *'...ij,...j->...i'*.

Parameters

- **m** (ArrayLike) – Matrix array m.
- **v** (ArrayLike) – Vector array v.

Returns Transformed vector array v.

Return type numpy.ndarray

Examples

```
>>> m = np.array(
...     [
...         [0.7328, 0.4296, -0.1624],
...         [-0.7036, 1.6975, 0.0061],
...         [0.0030, 0.0136, 0.9834],
...     ]
... )
>>> m = np.reshape(np.tile(m, (6, 1)), (6, 3, 3))
>>> v = np.array([0.20654008, 0.12197225, 0.05136952])
>>> v = np.tile(v, (6, 1))
>>> vector_dot(m, v)
array([[ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...]])
```

colour.algebra.matrix_dot

colour.algebra.**matrix_dot**(*a: ArrayLike, b: ArrayLike*) → NDArrayFloat

Perform the dot product of the matrix array a with the matrix array b.

This definition is a convenient wrapper around `np.einsum()` with the following subscripts: *'...ij,...jk->...ik'*.

Parameters

- **a** (ArrayLike) – Matrix array a.
- **b** (ArrayLike) – Matrix array b.

Return type numpy.ndarray

Examples

```
>>> a = np.array(
...     [
...         [0.7328, 0.4296, -0.1624],
...         [-0.7036, 1.6975, 0.0061],
...         [0.0030, 0.0136, 0.9834],
...     ]
... )
>>> a = np.reshape(np.tile(a, (6, 1)), (6, 3, 3))
>>> b = a
>>> matrix_dot(a, b)
array([[[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
```

(continues on next page)

(continued from previous page)

```
       [-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.algebra.euclidean_distance

colour.algebra.**euclidean_distance**(*a: ArrayLike*, *b: ArrayLike*) → NDArrayFloat

Return the *Euclidean* distance between point array a and point array b.

For a two-dimensional space, the metric is as follows:

$E_D = [(x_a - x_b)^2 + (y_a - y_b)^2]^{1/2}$

Parameters

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

Returns *Euclidean* distance.

Return type np.float or numpy.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.manhattan_distance

colour.algebra.**manhattan_distance**(*a: ArrayLike*, *b: ArrayLike*) → NDArrayFloat

Return the *Manhattan* (or *City-Block*) distance between point array a and point array b.

For a two-dimensional space, the metric is as follows:

$M_D = |x_a - x_b| + |y_a - y_b|$

Parameters

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

Returns *Manhattan* distance.

Return type np.float or numpy.ndarray

Examples

```
>>> a = np.array([100.00000000, 21.57210357, 272.22819350])
>>> b = np.array([100.00000000, 426.67945353, 72.39590835])
>>> manhattan_distance(a, b)
604.9396351...
```

colour.algebra.linear_conversion

`colour.algebra.linear_conversion`(*a: ArrayLike*, *old_range: ArrayLike*, *new_range: ArrayLike*) → NDArrayFloat

Perform a simple linear conversion of given array a between the old and new ranges.

Parameters

- **a** (`ArrayLike`) – Array a to perform the linear conversion onto.
- **old_range** (`ArrayLike`) – Old range.
- **new_range** (`ArrayLike`) – New range.

Returns Linear conversion result.

Return type `numpy.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.algebra.linstep_function

`colour.algebra.linstep_function`(*x: ArrayLike*, *a: ArrayLike = 0*, *b: ArrayLike = 1*, *clip: bool = False*) → NDArrayFloat

Perform a simple linear interpolation between given array a and array b using x array.

Parameters

- **x** (`ArrayLike`) – Array x value to use to interpolate between array a and array b.
- **a** (`ArrayLike`) – Array a, the start of the range in which to interpolate.
- **b** (`ArrayLike`) – Array b, the end of the range in which to interpolate.
- **clip** (`bool`) – Whether to clip the output values to range [`a`, `b`].

Returns Linear interpolation result.

Return type `numpy.ndarray`

Examples

```
>>> a = 0
>>> b = 2
>>> linstep_function(0.5, a, b)
1.0
```

colour.algebra.lerp

colour.algebra.**lerp**(*x: ArrayLike*, *a: ArrayLike = 0*, *b: ArrayLike = 1*, *clip: bool = False*) → NDArrayFloat

Perform a simple linear interpolation between given array a and array b using x array.

Parameters

- **x** (ArrayLike) – Array x value to use to interpolate between array a and array b.
- **a** (ArrayLike) – Array a, the start of the range in which to interpolate.
- **b** (ArrayLike) – Array b, the end of the range in which to interpolate.
- **clip** (bool) – Whether to clip the output values to range [a, b].

Returns Linear interpolation result.

Return type numpy.ndarray

Examples

```
>>> a = 0
>>> b = 2
>>> linstep_function(0.5, a, b)
1.0
```

colour.algebra.smoothstep_function

colour.algebra.**smoothstep_function**(*x: ArrayLike*, *a: ArrayLike = 0*, *b: ArrayLike = 1*, *clip: bool = False*) → NDArrayFloat

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

Parameters

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

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

Return type numpy.ndarray

Examples

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

colour.algebra.smooth

colour.algebra.**smooth**(*x: ArrayLike*, *a: ArrayLike = 0*, *b: ArrayLike = 1*, *clip: bool = False*) → NDArrayFloat

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

Parameters

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

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

Return type numpy.ndarray

Examples

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

colour.algebra.is_identity

colour.algebra.**is_identity**(*a: ArrayLike*) → bool

Return whether a array is an identity matrix.

Parameters **a** (ArrayLike) – Array a to test.

Returns Whether a array is an 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
```

colour.algebra.eigen_decomposition

colour.algebra.**eigen_decomposition**(*a: ArrayLike*, *eigen_w_v_count: int | None = None*, *descending_order: bool = True*, *covariance_matrix: bool = False*) → Tuple[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6d059ed3d0>, <sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6d059eec90>]

Return the eigen-values w and eigen-vectors v of given array a in given order.

Parameters

- **a** (ArrayLike) – Array to return the eigen-values w and eigen-vectors v for
- **eigen_w_v_count** (int | None) – Eigen-values w and eigen-vectors v count.
- **descending_order** (bool) – Whether to return the eigen-values w and eigen-vectors v in descending order.
- **covariance_matrix** (bool) – Whether to compute the eigen-values w and eigen-vectors v of the array a covariance matrix $A = a^T \cdot a$.

Returns Tuple of eigen-values w and eigen-vectors v. The eigenv-alues are in given order, each repeated according to its multiplicity. The column `v[:, i]` is the normalized eigen-vector corresponding to the eige-nvalue `w[i]`.

Return type tuple

Examples

```
>>> a = np.diag([1, 2, 3])
>>> w, v = eigen_decomposition(a)
>>> w
array([ 3.,  2.,  1.])
>>> v
array([[ 0.,  0.,  1.],
       [ 0.,  1.,  0.],
       [ 1.,  0.,  0.]])
>>> w, v = eigen_decomposition(a, 1)
>>> w
array([ 3.])
>>> v
array([[ 0.],
       [ 0.],
       [ 1.]])
>>> w, v = eigen_decomposition(a, descending_order=False)
>>> w
array([ 1.,  2.,  3.])
>>> v
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> w, v = eigen_decomposition(a, covariance_matrix=True)
>>> w
array([ 9.,  4.,  1.])
>>> v
array([[ 0.,  0.,  1.],
       [ 0.,  1.,  0.],
       [ 1.,  0.,  0.]])
```

Colour Appearance Models

ATD (1995)

colour

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

colour.XYZ_to_ATD95

colour.**XYZ_to_ATD95**(*XYZ: ArrayLike, XYZ_0: ArrayLike, Y_0: ArrayLike, k_1: ArrayLike, k_2: ArrayLike, sigma: ArrayLike = 300*) → *colour.appearance.atd95.CAM_Specification_ATD95*

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

Parameters

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

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

Return type colour.CAM_Specification_ATD95

Notes

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

Range	Scale - Reference	Scale - 1
CAM_Specification_ATD95.h	[0, 360]	[0, 1]

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

References

[Fai13e], [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)
CAM_Specification_ATD95(h=1.9089869..., C=1.2064060..., Q=0.1814003..., A_1=0.
↪1787931... T_1=0.0286942..., D_1=0.0107584..., A_2=0.0192182..., T_2=0.0205377...,␣
↪D_2=0.0107584...)
```

colour.CAM_Specification_ATD95

class colour.**CAM_Specification_ATD95**(*h: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, A_1: float | NDArrayFloat | None = <factory>, T_1: float | NDArrayFloat | None = <factory>, D_1: float | NDArrayFloat | None = <factory>, A_2: float | NDArrayFloat | None = <factory>, T_2: float | NDArrayFloat | None = <factory>, D_2: float | NDArrayFloat | None = <factory>*)

Define 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** (float | NDArrayFloat | None) – *Hue* angle H in degrees.
- **C** (float | NDArrayFloat | None) – 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** (float | NDArrayFloat | None) – Correlate of *brightness* Br.
- **A_1** (float | NDArrayFloat | None) – First stage A_1 response.
- **T_1** (float | NDArrayFloat | None) – First stage T_1 response.
- **D_1** (float | NDArrayFloat | None) – First stage D_1 response.
- **A_2** (float | NDArrayFloat | None) – Second stage A_2 response.
- **T_2** (float | NDArrayFloat | None) – Second stage A_2 response.
- **D_2** (float | NDArrayFloat | None) – Second stage D_2 response.

Return type None

Notes

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

References

[Fai13e], [Gut95]

__init__(*h: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, A_1: float | NDArrayFloat | None = <factory>, T_1: float | NDArrayFloat | None = <factory>, D_1: float | NDArrayFloat | None = <factory>, A_2: float | NDArrayFloat | None = <factory>, T_2: float | NDArrayFloat | None = <factory>, D_2: float | NDArrayFloat | None = <factory>*) → None

Parameters

- **h** (float | NDArrayFloat | None) –
- **C** (float | NDArrayFloat | None) –
- **Q** (float | NDArrayFloat | None) –
- **A_1** (float | NDArrayFloat | None) –
- **T_1** (float | NDArrayFloat | None) –
- **D_1** (float | NDArrayFloat | None) –
- **A_2** (float | NDArrayFloat | None) –
- **T_2** (float | NDArrayFloat | None) –
- **D_2** (float | NDArrayFloat | None) –

Return type None

Methods

`__init__`([h, C, Q, A_1, T_1, D_1, A_2, T_2, D_2])	
`arithmetical_operation`(a, operation[, in_place])	Perform given arithmetical operation with a operand on the `dataclass`-like class.

Attributes

fields	Getter property for the fields of the dataclass-like class.
items	Getter property for the dataclass-like class items, i.e. the field names and values.
keys	Getter property for the dataclass-like class keys, i.e. the field names.
values	Getter property for the dataclass-like class values, i.e. the field values.
h	
C	
Q	
A_1	
T_1	
D_1	
A_2	
T_2	
D_2	

CIECAM02

colour

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

colour.XYZ_to_CIECAM02

colour.**XYZ_to_CIECAM02**(*XYZ: ArrayLike, XYZ_w: ArrayLike, L_A: ArrayLike, Y_b: ArrayLike, surround: colour.appearance.ciecam02.InductionFactors_CIECAM02 = VIEWING_CONDITIONS_CIECAM02['Average'], discount_illuminant: bool = False, compute_H: bool = True*) → *colour.appearance.ciecam02.CAM_Specification_CIECAM02*

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

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of test sample / stimulus.

- **XYZ_w** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (ArrayLike) – 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** (ArrayLike) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (colour.appearance.ciecam02.InductionFactors_CIECAM02) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool) – Truth value indicating if the illuminant should be discounted.
- **compute_H** (bool) – Whether to compute *Hue* h quadrature H. H is rarely used, and expensive to compute.

Returns *CIECAM02* colour appearance model specification.

Return type colour.CAM_Specification_CIECAM02

Notes

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

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

References

[Fai04], [LL13], [MFH+02], [Wikipedia07d]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = VIEWING_CONDITIONS_CIECAM02["Average"]
>>> XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b, surround)
CAM_Specification_CIECAM02(J=41.7310911..., C=0.1047077..., h=219.0484326..., s=2.
↪3603053..., Q=195.3713259..., M=0.1088421..., H=278.0607358..., HC=None)
```

colour.CIECAM02_to_XYZ

colour.**CIECAM02_to_XYZ**(*specification: colour.appearance.ciecam02.CAM_Specification_CIECAM02, XYZ_w: ArrayLike, L_A: ArrayLike, Y_b: ArrayLike, surround: colour.appearance.ciecam02.InductionFactors_CIECAM02 = VIEWING_CONDITIONS_CIECAM02['Average'], discount_illuminant: bool = False*) → NDArrayFloat

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

Parameters

- **specification** (colour.appearance.ciecam02.CAM_Specification_CIECAM02) – *CIECAM02* colour appearance model specification. Correlate of *Lightness* J, correlate of *chroma* C or correlate of *colourfulness* M and *hue* angle h in degrees must be specified, e.g. JCh or JMh.
- **XYZ_w** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (ArrayLike) – 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** (ArrayLike) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (colour.appearance.ciecam02.InductionFactors_CIECAM02) – Surround viewing conditions.
- **discount_illuminant** (bool) – Discount the illuminant.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

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

Notes

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

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

References

[Fai04], [LL13], [MFH+02], [Wikipedia07d]

Examples

```
>>> specification = CAM_Specification_CIECAM02(
...     J=41.731091132513917, C=0.104707757171031, h=219.048432658311780
... )
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> CIECAM02_to_XYZ(specification, XYZ_w, L_A, Y_b)
array([ 19.01...,  20...   ,  21.78...])
```

colour.CAM_Specification_CIECAM02

class colour.**CAM_Specification_CIECAM02**(*J: float | NDArrayFloat | None = <factory>*, *C: float | NDArrayFloat | None = <factory>*, *h: float | NDArrayFloat | None = <factory>*, *s: float | NDArrayFloat | None = <factory>*, *Q: float | NDArrayFloat | None = <factory>*, *M: float | NDArrayFloat | None = <factory>*, *H: float | NDArrayFloat | None = <factory>*, *HC: float | NDArrayFloat | None = <factory>*)

Define the *CIECAM02* colour appearance model specification.

Parameters

- **J** (float | NDArrayFloat | None) – Correlate of *Lightness* J.
- **C** (float | NDArrayFloat | None) – Correlate of *chroma* C.
- **h** (float | NDArrayFloat | None) – *Hue* angle h in degrees.
- **s** (float | NDArrayFloat | None) – Correlate of *saturation* s.
- **Q** (float | NDArrayFloat | None) – Correlate of *brightness* Q.
- **M** (float | NDArrayFloat | None) – Correlate of *colourfulness* M.
- **H** (float | NDArrayFloat | None) – *Hue* h quadrature H.
- **HC** (float | NDArrayFloat | None) – *Hue* h composition H^C.

Return type None

References

[Fai04], [LL13], [MFH+02], [Wikipedia07d]

__init__(*J: float | NDArrayFloat | None = <factory>*, *C: float | NDArrayFloat | None = <factory>*, *h: float | NDArrayFloat | None = <factory>*, *s: float | NDArrayFloat | None = <factory>*, *Q: float | NDArrayFloat | None = <factory>*, *M: float | NDArrayFloat | None = <factory>*, *H: float | NDArrayFloat | None = <factory>*, *HC: float | NDArrayFloat | None = <factory>*) → None

Parameters

- **J** (float | NDArrayFloat | None) –

- **C** (float | NDArrayFloat | None) –
- **h** (float | NDArrayFloat | None) –
- **s** (float | NDArrayFloat | None) –
- **Q** (float | NDArrayFloat | None) –
- **M** (float | NDArrayFloat | None) –
- **H** (float | NDArrayFloat | None) –
- **HC** (float | NDArrayFloat | None) –

Return type None

Methods

`__init__`([J, C, h, s, Q, M, H, HC])	
`arithmetical_operation`(a, operation[, in_place])	Perform given arithmetical operation with *a* operand on the `dataclass`-like class.

Attributes

`fields`	Getter property for the fields of the `dataclass`-like class.
`items`	Getter property for the `dataclass`-like class items, i.e. the field names and values.
`keys`	Getter property for the `dataclass`-like class keys, i.e. the field names.
`values`	Getter property for the `dataclass`-like class values, i.e. the field values.
`J`	
`C`	
`h`	
`s`	
`Q`	
`M`	
`H`	
`HC`	

colour.VIEWING_CONDITIONS_CIECAM02

`colour.VIEWING_CONDITIONS_CIECAM02 = CanonicalMapping({'Average': ..., 'Dim': ..., 'Dark': ...})`

Reference *CIECAM02* colour appearance model viewing conditions.

References

[Fai04], [LL13], [MFH+02], [Wikipedia07d]

Ancillary Objects

`colour.appearance`

`CAM_KWARGS_CIECAM02_sRGB`	Default parameter values for the *CIECAM02* colour appearance model usage in the context of *sRGB*.
`InductionFactors_CIECAM02`(F, c, N_c)	*CIECAM02* colour appearance model induction factors.

colour.appearance.CAM_KWARGS_CIECAM02_sRGB

`colour.appearance.CAM_KWARGS_CIECAM02_sRGB = {'L_A': 4.074366543152521, 'XYZ_w': array([ 95.04559271, 100. , 108.90577508]), 'Y_b': 20, 'surround': InductionFactors_CIECAM02(F=1, c=0.69, N_c=1)}`

Default parameter values for the *CIECAM02* colour appearance model usage in the context of *sRGB*.

References

[Fai04], [InternationalECommission99], [LL13], [MFH+02], [Wikipedia07d]

colour.appearance.InductionFactors_CIECAM02

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

CIECAM02 colour appearance model induction factors.

Parameters

- **F** – Maximum degree of adaptation F.
- **c** – Exponential non-linearity c.
- **N_c** – Chromatic induction factor N_c.

References

[Fai04], [LL13], [MFH+02], [Wikipedia07d]

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

`__init__()`

Methods

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

Attributes

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

CIECAM16

`colour`

`XYZ_to_CIECAM16`(XYZ, XYZ_w, L_A, Y_b[, ...])	Compute the *CIECAM16* colour appearance model correlates from given *CIE XYZ* tristimulus values.
`CIECAM16_to_XYZ`(specification, XYZ_w, L_A, Y_b)	Convert from *CIECAM16* specification to *CIE XYZ* tristimulus values.
`CAM_Specification_CIECAM16`(J, C, h, s, Q, M, ...)	Define the *CIECAM16* colour appearance model specification.
`VIEWING_CONDITIONS_CIECAM16`	Reference *CIECAM16* colour appearance model viewing conditions.

colour.XYZ_to_CIECAM16

`colour.`**`XYZ_to_CIECAM16`**(*XYZ: ArrayLike, XYZ_w: ArrayLike, L_A: ArrayLike, Y_b: ArrayLike, surround:* colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.ciecam16.InductionFactors_CIECAM16 = *VIEWING_CONDITIONS_CIECAM16['Average'], discount_illuminant:* bool = *False, compute_H:* bool = *True*) → *colour.appearance.ciecam16.CAM_Specification_CIECAM16*

Compute the *CIECAM16* colour appearance model correlates from given *CIE XYZ* tristimulus values.

Parameters

- **XYZ** (`ArrayLike`) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (`ArrayLike`) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (`ArrayLike`) – 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** (`ArrayLike`) – Luminous factor of background Y_b such as $Y_b = 100 x L_b / L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (`colour.appearance.ciecam02.InductionFactors_CIECAM02` | `colour.appearance.ciecam16.InductionFactors_CIECAM16`) – Surround viewing conditions induction factors.

- **discount_illuminant** (bool) – Truth value indicating if the illuminant should be discounted.
- **compute_H** (bool) – Whether to compute *Hue* h quadrature H. H is rarely used, and expensive to compute.

Returns *CIECAM16* colour appearance model specification.

Return type colour.CAM_Specification_CIECAM16

Notes

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

Range	Scale - Reference	Scale - 1
CAM_Specification_CIECAM16.J	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.C	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.h	[0, 360]	[0, 1]
CAM_Specification_CIECAM16.s	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.Q	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.M	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.H	[0, 400]	[0, 1]

References

[CIED1CIED822]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = VIEWING_CONDITIONS_CIECAM16["Average"]
>>> XYZ_to_CIECAM16(XYZ, XYZ_w, L_A, Y_b, surround)
CAM_Specification_CIECAM16(J=41.7312079..., C=0.1033557..., h=217.0679597..., s=2.
↪3450150..., Q=195.3717089..., M=0.1074367..., H=275.5949861..., HC=None)
```

colour.CIECAM16_to_XYZ

colour.**CIECAM16_to_XYZ**(*specification:* colour.appearance.ciecam16.CAM_Specification_CIECAM16, *XYZ_w: ArrayLike, L_A: ArrayLike, Y_b: ArrayLike, surround:* colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.ciecam16.InductionFactors_CIECAM16 = *VIEWING_CONDITIONS_CIECAM16['Average'], discount_illuminant:* bool = *False*) → NDArrayFloat

Convert from *CIECAM16* specification to *CIE XYZ* tristimulus values.

Parameters

- **specification** (colour.appearance.ciecam16.CAM_Specification_CIECAM16) – *CIECAM16* 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** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (ArrayLike) – 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** (ArrayLike) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.ciecam16.InductionFactors_CIECAM16) – Surround viewing conditions.
- **discount_illuminant** (bool) – Discount the illuminant.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

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

Notes

Domain	Scale - Reference	Scale - 1
CAM_Specification_CIECAM16.J	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.C	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.h	[0, 360]	[0, 1]
CAM_Specification_CIECAM16.s	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.Q	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.M	[0, 100]	[0, 1]
CAM_Specification_CIECAM16.H	[0, 360]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

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

References

[CIED1CIED822]

Examples

```
>>> specification = CAM_Specification_CIECAM16(
...     J=41.731207905126638, C=0.103355738709070, h=217.067959767393010
... )
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> CIECAM16_to_XYZ(specification, XYZ_w, L_A, Y_b)
array([ 19.01...,  20...   ,  21.78...])
```

colour.CAM_Specification_CIECAM16

class colour.**CAM_Specification_CIECAM16**(*J: float | NDArrayFloat | None = <factory>*, *C: float | NDArrayFloat | None = <factory>*, *h: float | NDArrayFloat | None = <factory>*, *s: float | NDArrayFloat | None = <factory>*, *Q: float | NDArrayFloat | None = <factory>*, *M: float | NDArrayFloat | None = <factory>*, *H: float | NDArrayFloat | None = <factory>*, *HC: float | NDArrayFloat | None = <factory>*)

Define the *CIECAM16* colour appearance model specification.

Parameters

- **J** (float | NDArrayFloat | None) – Correlate of *Lightness* J.
- **C** (float | NDArrayFloat | None) – Correlate of *chroma* C.
- **h** (float | NDArrayFloat | None) – *Hue* angle h in degrees.
- **s** (float | NDArrayFloat | None) – Correlate of *saturation* s.
- **Q** (float | NDArrayFloat | None) – Correlate of *brightness* Q.
- **M** (float | NDArrayFloat | None) – Correlate of *colourfulness* M.
- **H** (float | NDArrayFloat | None) – *Hue* h quadrature H.
- **HC** (float | NDArrayFloat | None) – *Hue* h composition H^C.

Return type None

References

[CIED1CIED822]

__init__(*J: float | NDArrayFloat | None = <factory>*, *C: float | NDArrayFloat | None = <factory>*, *h: float | NDArrayFloat | None = <factory>*, *s: float | NDArrayFloat | None = <factory>*, *Q: float | NDArrayFloat | None = <factory>*, *M: float | NDArrayFloat | None = <factory>*, *H: float | NDArrayFloat | None = <factory>*, *HC: float | NDArrayFloat | None = <factory>*) → None

Parameters

- **J** (float | NDArrayFloat | None) –
- **C** (float | NDArrayFloat | None) –
- **h** (float | NDArrayFloat | None) –
- **s** (float | NDArrayFloat | None) –

- **Q** (float | NDArrayFloat | None) –
- **M** (float | NDArrayFloat | None) –
- **H** (float | NDArrayFloat | None) –
- **HC** (float | NDArrayFloat | None) –

Return type None

Methods

`__init__`([J, C, h, s, Q, M, H, HC])	
arithmetical_operation(a, operation[, in_place])	Perform given arithmetical operation with a operand on the dataclass-like class.

Attributes

fields	Getter property for the fields of the dataclass-like class.
items	Getter property for the dataclass-like class items, i.e. the field names and values.
keys	Getter property for the dataclass-like class keys, i.e. the field names.
values	Getter property for the dataclass-like class values, i.e. the field values.
J	
C	
h	
s	
Q	
M	
H	
HC	

colour.VIEWING_CONDITIONS_CIECAM16

`colour.VIEWING_CONDITIONS_CIECAM16 = CanonicalMapping({'Average': ..., 'Dim': ..., 'Dark': ...})`

Reference *CIECAM16* colour appearance model viewing conditions.

References

[CIED1CIED822]

Ancillary Objects

colour.appearance

InductionFactors_CIECAM16(F, c, N_c)	*CIECAM16* colour appearance model induction factors.

colour.appearance.InductionFactors_CIECAM16

class colour.appearance.**InductionFactors_CIECAM16**(*F, c, N_c*)

CIECAM16 colour appearance model induction factors.

Parameters

- **F** – Maximum degree of adaptation F.
- **c** – Exponential non-linearity c.
- **N_c** – Chromatic induction factor N_c.

Notes

- The *CIECAM16* colour appearance model induction factors are the same as *CIECAM02* colour appearance model.

References

[CIED1CIED822]

Create new instance of InductionFactors_CIECAM16(F, c, N_c)

__init__()

Methods

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

Attributes

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

CAM16

colour

XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b[, ...])	Compute the *CAM16* colour appearance model correlates from given *CIE XYZ* tristimulus values.
CAM16_to_XYZ(specification, XYZ_w, L_A, Y_b)	Convert from *CAM16* specification to *CIE XYZ* tristimulus values.
CAM_Specification_CAM16(J, C, h, s, Q, M, H, HC)	Define the *CAM16* colour appearance model specification.
VIEWING_CONDITIONS_CAM16	Reference *CAM16* colour appearance model viewing conditions.

colour.XYZ_to_CAM16

colour.**XYZ_to_CAM16**(*XYZ: ArrayLike*, *XYZ_w: ArrayLike*, *L_A: ArrayLike*, *Y_b: ArrayLike*, *surround: colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.cam16.InductionFactors_CAM16 = VIEWING_CONDITIONS_CAM16['Average']*, *discount_illuminant: bool = False*, *compute_H: bool = True*) → *colour.appearance.cam16.CAM_Specification_CAM16*

Compute the *CAM16* colour appearance model correlates from given *CIE XYZ* tristimulus values.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (ArrayLike) – 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** (ArrayLike) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.cam16.InductionFactors_CAM16) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool) – Truth value indicating if the illuminant should be discounted.
- **compute_H** (bool) – Whether to compute *Hue* h quadrature H. H is rarely used, and expensive to compute.

Returns *CAM16* colour appearance model specification.

Return type colour.CAM_Specification_CAM16

Notes

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

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

References

[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 = VIEWING_CONDITIONS_CAM16["Average"]
>>> XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b, surround)
CAM_Specification_CAM16(J=41.7312079..., C=0.1033557..., h=217.0679597..., s=2.
↪3450150..., Q=195.3717089..., M=0.1074367..., H=275.5949861..., HC=None)
```

colour.CAM16_to_XYZ

colour.**CAM16_to_XYZ**(*specification: colour.appearance.cam16.CAM_Specification_CAM16, XYZ_w: ArrayLike, L_A: ArrayLike, Y_b: ArrayLike, surround: colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.cam16.InductionFactors_CAM16 = VIEWING_CONDITIONS_CAM16['Average'], discount_illuminant: bool = False*) → NDArrayFloat

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

Parameters

- **specification** (colour.appearance.cam16.CAM_Specification_CAM16) – *CAM16* colour appearance model specification. Correlate of *Lightness* J, correlate of *chroma* C or correlate of *colourfulness* M and *hue* angle h in degrees must be specified, e.g. JCh or JMh.
- **XYZ_w** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (ArrayLike) – 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** (ArrayLike) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels

in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.

- **surround** (colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.cam16.InductionFactors_CAM16) – Surround viewing conditions.
- **discount_illuminant** (bool) – Discount the illuminant.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

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

Notes

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

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

References

[LLW+17]

Examples

```
>>> specification = CAM_Specification_CAM16(
...     J=41.731207905126638, C=0.103355738709070, h=217.067959767393010
... )
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> CAM16_to_XYZ(specification, XYZ_w, L_A, Y_b)
array([ 19.01...,  20...   ,  21.78...])
```

colour.CAM_Specification_CAM16

class colour.**CAM_Specification_CAM16**(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>*)

Define the *CAM16* colour appearance model specification.

Parameters

- **J** (float | NDArrayFloat | None) – Correlate of *Lightness* J.
- **C** (float | NDArrayFloat | None) – Correlate of *chroma* C.
- **h** (float | NDArrayFloat | None) – *Hue* angle h in degrees.
- **s** (float | NDArrayFloat | None) – Correlate of *saturation* s.
- **Q** (float | NDArrayFloat | None) – Correlate of *brightness* Q.
- **M** (float | NDArrayFloat | None) – Correlate of *colourfulness* M.
- **H** (float | NDArrayFloat | None) – *Hue* h quadrature H.
- **HC** (float | NDArrayFloat | None) – *Hue* h composition H^C.

Return type None

References

[LLW+17]

__init__(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>*) → None

Parameters

- **J** (float | NDArrayFloat | None) –
- **C** (float | NDArrayFloat | None) –
- **h** (float | NDArrayFloat | None) –
- **s** (float | NDArrayFloat | None) –
- **Q** (float | NDArrayFloat | None) –
- **M** (float | NDArrayFloat | None) –
- **H** (float | NDArrayFloat | None) –
- **HC** (float | NDArrayFloat | None) –

Return type None

Methods

`__init__([J, C, h, s, Q, M, H, HC])`	
`arithmetical_operation(a, operation[, in_place])`	Perform given arithmetical operation with a operand on the `dataclass`-like class.

Attributes

`fields`	Getter property for the fields of the `dataclass`-like class.
`items`	Getter property for the `dataclass`-like class items, i.e. the field names and values.
`keys`	Getter property for the `dataclass`-like class keys, i.e. the field names.
`values`	Getter property for the `dataclass`-like class values, i.e. the field values.
`J`	
`C`	
`h`	
`s`	
`Q`	
`M`	
`H`	
`HC`	

colour.VIEWING_CONDITIONS_CAM16

`colour.VIEWING_CONDITIONS_CAM16 = CanonicalMapping({'Average': ..., 'Dim': ..., 'Dark': ...})`

Reference *CAM16* colour appearance model viewing conditions.

References

[LLW+17]

Ancillary Objects

`colour.appearance`

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

colour.appearance.InductionFactors_CAM16

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

CAM16 colour appearance model induction factors.

Parameters

- **F** – Maximum degree of adaptation F.
- **c** – Exponential non-linearity c.
- **N_c** – Chromatic induction factor N_c.

Notes

- The *CAM16* colour appearance model induction factors are the same as *CIECAM02* colour appearance model.

References

[LLW+17]

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

__init__()

Methods

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

Attributes

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

Hellwig and Fairchild (2022) Colour Appearance Model

colour

XYZ_to_Hellwig2022(XYZ, XYZ_w, L_A, Y_b[, ...])	Compute the *Hellwig and Fairchild (2022)* colour appearance model correlates from given *CIE XYZ* tristimulus values.
Hellwig2022_to_XYZ(specification, XYZ_w, ...)	Convert from *Hellwig and Fairchild (2022)* specification to *CIE XYZ* tristimulus values.
CAM_Specification_Hellwig2022(J, C, h, s, Q, ...)	Define the *Hellwig and Fairchild (2022)* colour appearance model specification.
VIEWING_CONDITIONS_HELLWIG2022	Reference *Hellwig and Fairchild (2022)* colour appearance model viewing conditions.

colour.XYZ_to_Hellwig2022

colour.**XYZ_to_Hellwig2022**(*XYZ: ArrayLike, XYZ_w: ArrayLike, L_A: ArrayLike, Y_b: ArrayLike, surround: colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.hellwig2022.InductionFactors_Hellwig2022 = VIEWING_CONDITIONS_HELLWIG2022['Average'], discount_illuminant: bool = False, compute_H: bool = True*) → *colour.appearance.hellwig2022.CAM_Specification_Hellwig2022*

Compute the *Hellwig and Fairchild (2022)* colour appearance model correlates from given *CIE XYZ* tristimulus values.

This implementation supports the *Helmholtz-Kohlrausch* effect extension from [HSF22].

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (ArrayLike) – 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** (ArrayLike) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.hellwig2022.InductionFactors_Hellwig2022) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool) – Truth value indicating if the illuminant should be discounted.
- **compute_H** (bool) – Whether to compute *Hue* h quadrature H. H is rarely used, and expensive to compute.

Returns *Hellwig and Fairchild (2022)* colour appearance model specification.

Return type colour.CAM_Specification_Hellwig2022

Notes

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

Range	Scale - Reference	Scale - 1
CAM_Specification_Hellwig2022.J	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.C	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.h	[0, 360]	[0, 1]
CAM_Specification_Hellwig2022.s	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.Q	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.M	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.H	[0, 400]	[0, 1]
CAM_Specification_Hellwig2022.J_HK	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.Q_HK	[0, 100]	[0, 1]

References

[FH22], [HF22], [HSF22]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = VIEWING_CONDITIONS_HELLWIG2022["Average"]
>>> XYZ_to_Hellwig2022(XYZ, XYZ_w, L_A, Y_b, surround)
...
CAM_Specification_Hellwig2022(J=41.7312079..., C=0.0257636..., h=217.0679597..., s=0.
↪0608550..., Q=55.8523226..., M=0.0339889..., H=275.5949861..., HC=None, J_HK=41.
↪8802782..., Q_HK=56.0518358...)
```

colour.Hellwig2022_to_XYZ

colour.**Hellwig2022_to_XYZ**(*specification: colour.appearance.hellwig2022.CAM_Specification_Hellwig2022, XYZ_w: ArrayLike, L_A: ArrayLike, Y_b: ArrayLike, surround: colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.hellwig2022.InductionFactors_Hellwig2022 = VIEWING_CONDITIONS_HELLWIG2022['Average'], discount_illuminant: bool = False*) → NDArrayFloat

Convert from *Hellwig and Fairchild (2022)* specification to *CIE XYZ* tristimulus values.

This implementation supports the *Helmholtz-Kohlrausch* effect extension from [HSF22].

Parameters

- **specification** (colour.appearance.hellwig2022.CAM_Specification_Hellwig2022) – *Hellwig and Fairchild (2022)* 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** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (ArrayLike) – 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** (ArrayLike) – Luminous factor of background Y_b such as $Y_b = 100 x L_b / L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (colour.appearance.ciecam02.InductionFactors_CIECAM02 | colour.appearance.hellwig2022.InductionFactors_Hellwig2022) – Surround viewing conditions.
- **discount_illuminant** (bool) – Discount the illuminant.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

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

Notes

Domain	Scale - Reference	Scale - 1
CAM_Specification_Hellwig2022.J	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.C	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.h	[0, 360]	[0, 1]
CAM_Specification_Hellwig2022.s	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.Q	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.M	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.H	[0, 400]	[0, 1]
CAM_Specification_Hellwig2022.J_HK	[0, 100]	[0, 1]
CAM_Specification_Hellwig2022.Q_HK	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

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

References

[FH22], [HF22], [HSF22]

Examples

```
>>> specification = CAM_Specification_Hellwig2022(
...     J=41.731207905126638, C=0.025763615829912909, h=217.06795976739301
... )
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> Hellwig2022_to_XYZ(specification, XYZ_w, L_A, Y_b)
...
array([ 19.01...,  20...   ,  21.78...])
>>> specification = CAM_Specification_Hellwig2022(
...     J_HK=41.880278283880095,
...     C=0.025763615829912909,
...     h=217.06795976739301,
... )
>>> Hellwig2022_to_XYZ(specification, XYZ_w, L_A, Y_b)
...
array([ 19.01...,  20...   ,  21.78...])
```

colour.CAM_Specification_Hellwig2022

class colour.**CAM_Specification_Hellwig2022**(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>, J_HK: float | NDArrayFloat | None = <factory>, Q_HK: float | NDArrayFloat | None = <factory>*)

Define the *Hellwig and Fairchild (2022)* colour appearance model specification.

This specification supports the *Helmholtz-Kohlrausch* effect extension from [HSF22].

Parameters

- **J** (float | NDArrayFloat | None) – Correlate of *Lightness* J.
- **C** (float | NDArrayFloat | None) – Correlate of *chroma* C.
- **h** (float | NDArrayFloat | None) – *Hue* angle h in degrees.
- **s** (float | NDArrayFloat | None) – Correlate of *saturation* s.
- **Q** (float | NDArrayFloat | None) – Correlate of *brightness* Q.
- **M** (float | NDArrayFloat | None) – Correlate of *colourfulness* M.
- **H** (float | NDArrayFloat | None) – *Hue* h quadrature H.
- **HC** (float | NDArrayFloat | None) – *Hue* h composition H^C.
- **J_HK** (float | NDArrayFloat | None) – Correlate of *Lightness* J_{HK} accounting for *Helmholtz-Kohlrausch* effect.
- **Q_HK** (float | NDArrayFloat | None) – Correlate of *brightness* Q_{HK} accounting for *Helmholtz-Kohlrausch* effect.

Return type None

References

[FH22], [HF22], [HSF22]

__init__(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>, J_HK: float | NDArrayFloat | None = <factory>, Q_HK: float | NDArrayFloat | None = <factory>*) → None

Parameters

- **J** (float | NDArrayFloat | None) –
- **C** (float | NDArrayFloat | None) –
- **h** (float | NDArrayFloat | None) –
- **s** (float | NDArrayFloat | None) –
- **Q** (float | NDArrayFloat | None) –
- **M** (float | NDArrayFloat | None) –
- **H** (float | NDArrayFloat | None) –
- **HC** (float | NDArrayFloat | None) –
- **J_HK** (float | NDArrayFloat | None) –
- **Q_HK** (float | NDArrayFloat | None) –

Return type None

Methods

__init__([J, C, h, s, Q, M, H, HC, J_HK, Q_HK])	
arithmetical_operation(a, operation[, in_place])	Perform given arithmetical operation with a operand on the dataclass-like class.

Attributes

fields	Getter property for the fields of the dataclass-like class.
items	Getter property for the dataclass-like class items, i.e. the field names and values.
keys	Getter property for the dataclass-like class keys, i.e. the field names.
values	Getter property for the dataclass-like class values, i.e. the field values.
J	
C	
h	
s	
Q	
M	
H	
HC	
J_HK	
Q_HK	

colour.VIEWING_CONDITIONS_HELLWIG2022

`colour.VIEWING_CONDITIONS_HELLWIG2022 = CanonicalMapping({'Average': ..., 'Dim': ..., 'Dark': ...})`

Reference *Hellwig and Fairchild (2022)* colour appearance model viewing conditions.

References

[HF22]

Ancillary Objects

`colour.appearance`

`InductionFactors_Hellwig2022`(F, c, N_c)	*Hellwig and Fairchild (2022)* colour appearance model induction factors.

colour.appearance.InductionFactors_Hellwig2022

class `colour.appearance.InductionFactors_Hellwig2022`(*F*, *c*, *N_c*)

Hellwig and Fairchild (2022) colour appearance model induction factors.

Parameters

- **F** – Maximum degree of adaptation F.
- **c** – Exponential non-linearity c.
- **N_c** – Chromatic induction factor N_c.

Notes

- The *Hellwig and Fairchild (2022)* colour appearance model induction factors are the same as *CIECAM02* and *CAM16* colour appearance model.

References

[FH22], [HF22]

Create new instance of InductionFactors_Hellwig2022(F, c, N_c)

`__init__`()

Methods

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

Attributes

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

Hunt

`colour`

XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A[, ...])	Compute the *Hunt* colour appearance model correlates.
CAM_Specification_Hunt(J, C, h, s, Q, M, H, HC)	Define the *Hunt* colour appearance model specification.
VIEWING_CONDITIONS_HUNT	Reference *Hunt* colour appearance model viewing conditions.

colour.XYZ_to_Hunt

colour.**XYZ_to_Hunt**(*XYZ: ArrayLike, XYZ_w: ArrayLike, XYZ_b: ArrayLike, L_A: ArrayLike, surround: InductionFactors_Hunt = VIEWING_CONDITIONS_HUNT['Normal Scenes'], L_AS: ArrayLike | None = None, CCT_w: ArrayLike | None = None, XYZ_p: ArrayLike | None = None, p: ArrayLike | None = None, S: ArrayLike | None = None, S_w: ArrayLike | None = None, helson_judd_effect: bool = False, discount_illuminant: bool = True*) → *CAM_Specification_Hunt*

Compute the *Hunt* colour appearance model correlates.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **XYZ_b** (ArrayLike) – *CIE XYZ* tristimulus values of background.
- **L_A** (ArrayLike) – Adapting field *luminance* L_A in cd/m^2.
- **surround** (InductionFactors_Hunt) – Surround viewing conditions induction factors.
- **L_AS** (ArrayLike | None) – Scotopic luminance L_{AS} of the illuminant, approximated if not specified.
- **CCT_w** (ArrayLike | None) – Correlated color temperature T_{cp}: of the illuminant, needed to approximate L_{AS}.
- **XYZ_p** (ArrayLike | None) – *CIE XYZ* tristimulus values of proximal field, assumed to be equal to background if not specified.
- **p** (ArrayLike | None) – Simultaneous contrast / assimilation factor p with value normalised to domain [-1, 0] when simultaneous contrast occurs and normalised to domain [0, 1] when assimilation occurs.
- **S** (ArrayLike | None) – Scotopic response S to the stimulus, approximated using tristimulus values Y of the stimulus if not specified.
- **S_w** (ArrayLike | None) – 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) – Truth value indicating whether the *Helson-Judd* effect should be accounted for.

- **discount_illuminant** (bool) – Truth value indicating if the illuminant should be discounted.

Returns *Hunt* colour appearance model specification.

Return type colour.CAM_Specification_Hunt

Raises ValueError – If an illegal argument combination is specified.

Notes

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

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

References

[Fai13b], [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 = VIEWING_CONDITIONS_HUNT["Normal Scenes"]
>>> CCT_w = 6504
>>> XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A, surround, CCT_w=CCT_w)
...
CAM_Specification_Hunt(J=30.0462678..., C=0.1210508..., h=269.2737594..., s=0.
↪0199093..., Q=22.2097654..., M=0.1238964..., H=None, HC=None)
```

colour.CAM_Specification_Hunt

class colour.**CAM_Specification_Hunt**(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>*)

Define 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** (float | NDArrayFloat | None) – Correlate of *Lightness* J.

- **C** (float | NDArrayFloat | None) – Correlate of *chroma* C_94.
- **h** (float | NDArrayFloat | None) – *Hue* angle h_S in degrees.
- **s** (float | NDArrayFloat | None) – Correlate of *saturation* s.
- **Q** (float | NDArrayFloat | None) – Correlate of *brightness* Q.
- **M** (float | NDArrayFloat | None) – Correlate of *colourfulness* M_94.
- **H** (float | NDArrayFloat | None) – *Hue* h quadrature H.
- **HC** (float | NDArrayFloat | None) – *Hue* h composition H_C.

Return type None

Notes

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

References

[Fai13b], [Hun04]

__init__(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>*) → None

Parameters

- **J** (float | NDArrayFloat | None) –
- **C** (float | NDArrayFloat | None) –
- **h** (float | NDArrayFloat | None) –
- **s** (float | NDArrayFloat | None) –
- **Q** (float | NDArrayFloat | None) –
- **M** (float | NDArrayFloat | None) –
- **H** (float | NDArrayFloat | None) –
- **HC** (float | NDArrayFloat | None) –

Return type None

Methods

`__init__`([J, C, h, s, Q, M, H, HC])	
`arithmetical_operation`(a, operation[, in_place])	Perform given arithmetical operation with a operand on the `dataclass`-like class.

Attributes

fields	Getter property for the fields of the dataclass-like class.
items	Getter property for the dataclass-like class items, i.e. the field names and values.
keys	Getter property for the dataclass-like class keys, i.e. the field names.
values	Getter property for the dataclass-like class values, i.e. the field values.
J	
C	
h	
s	
Q	
M	
H	
HC	

colour.VIEWING_CONDITIONS_HUNT

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

Reference *Hunt* colour appearance model viewing conditions.

References

[Fai13b], [Hun04]

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'

Kim, Weyrich and Kautz (2009)

colour

XYZ_to_Kim2009(XYZ, XYZ_w, L_A[, media, ...])	Compute the *Kim, Weyrich and Kautz (2009)* colour appearance model correlates from given *CIE XYZ* tristimulus values.
Kim2009_to_XYZ(specification, XYZ_w, L_A[, ...])	Convert from *Kim, Weyrich and Kautz (2009)* specification to *CIE XYZ* tristimulus values.
CAM_Specification_Kim2009(J, C, h, s, Q, M, ...)	Define the *Kim, Weyrich and Kautz (2009)* colour appearance model specification.
MEDIA_PARAMETERS_KIM2009	Reference *Kim, Weyrich and Kautz (2009)* colour appearance model media parameters.
VIEWING_CONDITIONS_KIM2009	Reference *Kim, Weyrich and Kautz (2009)* colour appearance model viewing conditions.

colour.XYZ_to_Kim2009

colour.**XYZ_to_Kim2009**(*XYZ: ArrayLike, XYZ_w: ArrayLike, L_A: ArrayLike, media: colour.appearance.kim2009.MediaParameters_Kim2009 = MEDIA_PARAMETERS_KIM2009['CRT Displays'], surround: colour.appearance.kim2009.InductionFactors_Kim2009 = VIEWING_CONDITIONS_KIM2009['Average'], n_c: float = 0.57, discount_illuminant: bool = False, compute_H: bool = True*) → *colour.appearance.kim2009.CAM_Specification_Kim2009*

Compute the *Kim, Weyrich and Kautz (2009)* colour appearance model correlates from given *CIE XYZ* tristimulus values.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (ArrayLike) – Adapting field *luminance* L_A in cd/m^2, (often taken to be 20% of the luminance of a white object in the scene).
- **media** (colour.appearance.kim2009.MediaParameters_Kim2009) – Media parameters.
- **surround** (colour.appearance.kim2009.InductionFactors_Kim2009) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool) – Truth value indicating if the illuminant should be discounted.
- **compute_H** (bool) – Whether to compute *Hue* h quadrature H. H is rarely used, and expensive to compute.
- **n_c** (float) – Cone response sigmoidal curve modulating factor n_c.

Returns *Kim, Weyrich and Kautz (2009)* colour appearance model specification.

Return type colour.CAM_Specification_Kim2009

Notes

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

Range	Scale - Reference	Scale - 1
CAM_Specification_Kim2009.J	[0, 100]	[0, 1]
CAM_Specification_Kim2009.C	[0, 100]	[0, 1]
CAM_Specification_Kim2009.h	[0, 360]	[0, 1]
CAM_Specification_Kim2009.s	[0, 100]	[0, 1]
CAM_Specification_Kim2009.Q	[0, 100]	[0, 1]
CAM_Specification_Kim2009.M	[0, 100]	[0, 1]
CAM_Specification_Kim2009.H	[0, 400]	[0, 1]

References

[KWK09]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> media = MEDIA_PARAMETERS_KIM2009["CRT Displays"]
>>> surround = VIEWING_CONDITIONS_KIM2009["Average"]
>>> XYZ_to_Kim2009(XYZ, XYZ_w, L_A, media, surround)
...
CAM_Specification_Kim2009(J=28.8619089..., C=0.5592455..., h=219.0480667..., s=9.
↪3837797..., Q=52.7138883..., M=0.4641738..., H=278.0602824..., HC=None)
```

colour.Kim2009_to_XYZ

colour.**Kim2009_to_XYZ**(*specification:* colour.appearance.kim2009.CAM_Specification_Kim2009, *XYZ_w: ArrayLike*, *L_A: ArrayLike*, *media:* colour.appearance.kim2009.MediaParameters_Kim2009 = *MEDIA_PARAMETERS_KIM2009['CRT Displays']*, *surround:* colour.appearance.kim2009.InductionFactors_Kim2009 = *VIEWING_CONDITIONS_KIM2009['Average']*, *n_c:* *float* = *0.57*, *discount_illuminant:* *bool* = *False*) → NDArrayFloat

Convert from *Kim, Weyrich and Kautz (2009)* specification to *CIE XYZ* tristimulus values.

Parameters

- **specification** (colour.appearance.kim2009.CAM_Specification_Kim2009) – *Kim, Weyrich and Kautz (2009)* 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** (ArrayLike) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (ArrayLike) – Adapting field *luminance* L_A in cd/m^2, (often taken to be 20% of the luminance of a white object in the scene).

- **media** (colour.appearance.kim2009.MediaParameters_Kim2009) – Media parameters.
- **surroundl** – Surround viewing conditions induction factors.
- **discount_illuminant** (bool) – Discount the illuminant.
- **n_c** (float) – Cone response sigmoidal curve modulating factor n_c.
- **surround** (colour.appearance.kim2009.InductionFactors_Kim2009) –

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

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

Notes

Domain	Scale - Reference	Scale - 1
CAM_Specification_Kim2009.J	[0, 100]	[0, 1]
CAM_Specification_Kim2009.C	[0, 100]	[0, 1]
CAM_Specification_Kim2009.h	[0, 360]	[0, 1]
CAM_Specification_Kim2009.s	[0, 100]	[0, 1]
CAM_Specification_Kim2009.Q	[0, 100]	[0, 1]
CAM_Specification_Kim2009.M	[0, 100]	[0, 1]
CAM_Specification_Kim2009.H	[0, 360]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

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

References

[KWK09]

Examples

```
>>> specification = CAM_Specification_Kim2009(
...     J=28.861908975839647, C=0.5592455924373706, h=219.04806677662953
... )
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> media = MEDIA_PARAMETERS_KIM2009["CRT Displays"]
>>> surround = VIEWING_CONDITIONS_KIM2009["Average"]
>>> Kim2009_to_XYZ(specification, XYZ_w, L_A, media, surround)
...
array([ 19.0099995...,  19.9999999...,  21.7800000...])
```

colour.CAM_Specification_Kim2009

class colour.**CAM_Specification_Kim2009**(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>*)

Define the *Kim, Weyrich and Kautz (2009)* colour appearance model specification.

Parameters

- **J** (float | NDArrayFloat | None) – Correlate of *Lightness* J.
- **C** (float | NDArrayFloat | None) – Correlate of *chroma* C.
- **h** (float | NDArrayFloat | None) – *Hue* angle h in degrees.
- **s** (float | NDArrayFloat | None) – Correlate of *saturation* s.
- **Q** (float | NDArrayFloat | None) – Correlate of *brightness* Q.
- **M** (float | NDArrayFloat | None) – Correlate of *colourfulness* M.
- **H** (float | NDArrayFloat | None) – *Hue* h quadrature H.
- **HC** (float | NDArrayFloat | None) – *Hue* h composition H^C.

Return type None

References

[KWK09]

__init__(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>*) → None

Parameters

- **J** (float | NDArrayFloat | None) –
- **C** (float | NDArrayFloat | None) –
- **h** (float | NDArrayFloat | None) –
- **s** (float | NDArrayFloat | None) –
- **Q** (float | NDArrayFloat | None) –
- **M** (float | NDArrayFloat | None) –
- **H** (float | NDArrayFloat | None) –
- **HC** (float | NDArrayFloat | None) –

Return type None

Methods

`__init__([J, C, h, s, Q, M, H, HC])`	
`arithmetical_operation(a, operation[, in_place])`	Perform given arithmetical operation with a operand on the `dataclass`-like class.

Attributes

`fields`	Getter property for the fields of the `dataclass`-like class.
`items`	Getter property for the `dataclass`-like class items, i.e. the field names and values.
`keys`	Getter property for the `dataclass`-like class keys, i.e. the field names.
`values`	Getter property for the `dataclass`-like class values, i.e. the field values.
`J`	
`C`	
`h`	
`s`	
`Q`	
`M`	
`H`	
`HC`	

colour.MEDIA_PARAMETERS_KIM2009

```
colour.MEDIA_PARAMETERS_KIM2009 = CanonicalMapping({'High-luminance LCD Display': ...,
'Transparent Advertising Media': ..., 'CRT Displays': ..., 'Reflective Paper': ...,
'bright_lcd_display': ..., 'advertising_transparencies': ..., 'crt': ..., 'paper': ...})
```

Reference *Kim, Weyrich and Kautz (2009)* colour appearance model media parameters.

References

[KWK09]

Aliases:

- 'bright_lcd_display': 'High-luminance LCD Display'
- 'advertising_transparencies': 'Transparent Advertising Media'
- 'crt': 'CRT Displays'
- 'paper': 'Reflective Paper'

colour.VIEWING_CONDITIONS_KIM2009

colour.**VIEWING_CONDITIONS_KIM2009 = CanonicalMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**

Reference *Kim, Weyrich and Kautz (2009)* colour appearance model viewing conditions.

References

[KWK09]

Ancillary Objects

colour.appearance

InductionFactors_Kim2009(F, c, N_c)	*Kim, Weyrich and Kautz (2009)* colour appearance model induction factors.
MediaParameters_Kim2009(E)	*Kim, Weyrich and Kautz (2009)* colour appearance model media parameters.

colour.appearance.InductionFactors_Kim2009

class colour.appearance.**InductionFactors_Kim2009**(*F, c, N_c*)

Kim, Weyrich and Kautz (2009) colour appearance model induction factors.

Parameters

- **F** – Maximum degree of adaptation F.
- **c** – Exponential non-linearity c.
- **N_c** – Chromatic induction factor N_c.

Notes

- The *Kim, Weyrich and Kautz (2009)* colour appearance model induction factors are the same as *CIECAM02* colour appearance model.
- The *Kim, Weyrich and Kautz (2009)* colour appearance model separates the surround modelled by the colour.appearance.InductionFactors_Kim2009 class instance from the media, modeled with the colour.appearance.MediaParameters_Kim2009 class instance.

References

[KWK09]

Create new instance of InductionFactors_Kim2009(F, c, N_c)

__init__()

Methods

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

Attributes

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

colour.appearance.MediaParameters_Kim2009

class colour.appearance.**MediaParameters_Kim2009**(E)

Kim, Weyrich and Kautz (2009) colour appearance model media parameters.

Parameters **E** – Lightness prediction modulating parameter E.

References

[KWK09]

Return a new instance of the `colour.appearance.MediaParameters_Kim2009` class.

__init__()

Methods

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

Attributes

`E`	Alias for field number 0

$LLAB(l : c)$

`colour`

`XYZ_to_LLAB`(XYZ, XYZ_0, Y_b, L[, surround])	Compute the *:math:`LLAB(l:c)`* colour appearance model correlates.
`CAM_Specification_LLAB`(J, ...)	Define the *:math:`LLAB(l:c)`* colour appearance model specification.
`VIEWING_CONDITIONS_LLAB`	Reference $LLAB(l : c)$ colour appearance model viewing conditions.

colour.XYZ_to_LLAB

colour.**XYZ_to_LLAB**(*XYZ: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], XYZ_0: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], Y_b: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], L: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], surround: colour.appearance.llab.InductionFactors_LLAB = VIEWING_CONDITIONS_LLAB['Reference Samples & Images, Average Surround, Subtending < 4']*) → *colour.appearance.llab.CAM_Specification_LLAB*

Compute the *:math: `LLAB(l:c)`* colour appearance model correlates.

Parameters

- **XYZ** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_0** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – *CIE XYZ* tristimulus values of reference white.
- **Y_b** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Luminance factor of the background in cd/m^2.
- **L** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Absolute luminance L of reference white in cd/m^2.
- **surround** (colour.appearance.llab.InductionFactors_LLAB) – Surround viewing conditions induction factors.

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

Return type colour.CAM_Specification_LLAB

Notes

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

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

References

[Fai13h], [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 = VIEWING_CONDITIONS_LLAB["ref_average_4_minus"]
>>> XYZ_to_LLAB(XYZ, XYZ_0, Y_b, L, surround)
CAM_Specification_LLAB(J=37.3668650..., C=0.0089496..., h=270..., s=0.0002395...,
↪M=0.0190185..., HC=None, a=..., b=-0.0190185...)
```

colour.CAM_Specification_LLAB

class colour.**CAM_Specification_LLAB**(*J: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, C: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, h: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, s: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, M: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, HC: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, a: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, b: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>*)

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

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

Parameters

- **J** (`Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]`) – Correlate of *Lightness* L_L.
- **C** (`Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]`) – Correlate of *chroma* Ch_L.
- **h** (`Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]`) – *Hue* angle h_L in degrees.
- **s** (`Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]`) – Correlate of *saturation* s_L.
- **M** (`Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]`) – Correlate of *colourfulness* C_L.
- **HC** (`Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]`) – *Hue* h composition H^C.
- **a** (`Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]`) – Opponent signal A_L.
- **b** (`Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]`) – Opponent signal B_L.

Return type None

Notes

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

References

[Fai13h], [LLK96], [LM96]

__init__(*J: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, C: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, h: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, s: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, M: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, HC: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, a: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>, b: typing.Optional[typing.Union[float, numpy.ndarray[typing.Any, numpy.dtype[typing.Union[numpy.float16, numpy.float32, numpy.float64]]]]] = <factory>*) → None

Parameters

- **J** (Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]) –
- **C** (Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]) –
- **h** (Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]) –
- **s** (Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]) –
- **M** (Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]) –
- **HC** (Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]) –
- **a** (Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]) –
- **b** (Optional[Union[float, numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]]]) –

Return type None

Methods

`__init__`([J, C, h, s, M, HC, a, b])	
`arithmetical_operation`(a, operation[, in_place])	Perform given arithmetical operation with a operand on the `dataclass`-like class.

Attributes

`fields`	Getter property for the fields of the `dataclass`-like class.
`items`	Getter property for the `dataclass`-like class items, i.e. the field names and values.
`keys`	Getter property for the `dataclass`-like class keys, i.e. the field names.
`values`	Getter property for the `dataclass`-like class values, i.e. the field values.
`J`	
`C`	
`h`	
`s`	
`M`	
`HC`	
`a`	
`b`	

colour.VIEWING_CONDITIONS_LLAB

colour.**VIEWING_CONDITIONS_LLAB = CanonicalMapping({'Reference Samples & Images, Average Surround, Subtending > 4': ..., 'Reference Samples & Images, Average Surround, Subtending < 4': ..., 'Television & VDU Displays, Dim Surround': ..., 'Cut Sheet Transparency, Dim Surround': ..., '35mm Projection Transparency, Dark Surround': ..., 'ref_average_4_plus': ..., 'ref_average_4_minus': ..., 'tv_dim': ..., 'sheet_dim': ..., 'projected_dark': ...})**

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

References

[Fai13h], [LLK96], [LM96]

Aliases:

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

Ancillary Objects

colour.appearance

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

colour.appearance.InductionFactors_LLAB

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

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

Parameters

- **D** – *Discounting-the-Illuminant* factor D.
- **F_S** – Surround induction factor F_S.
- **F_L** – *Lightness* induction factor F_L.
- **F_C** – *Chroma* induction factor F_C.

References

[Fai13h], [LLK96], [LM96]

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

__init__()

Methods

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

Attributes

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

Nayatani (1995)

`colour`

`XYZ_to_Nayatani95`(XYZ, XYZ_n, Y_o, E_o, E_or)	Compute the *Nayatani (1995)* colour appearance model correlates.
`CAM_Specification_Nayatani95`(L_star_P, C, h, ...)	Define the *Nayatani (1995)* colour appearance model specification.

colour.XYZ_to_Nayatani95

`colour.`**`XYZ_to_Nayatani95`**(*XYZ: ArrayLike*, *XYZ_n: ArrayLike*, *Y_o: ArrayLike*, *E_o: ArrayLike*, *E_or: ArrayLike*, *n: ArrayLike = 1*) → *colour.appearance.nayatani95.CAM_Specification_Nayatani95*

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

Parameters

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

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

Return type `colour.CAM_Specification_Nayatani95`

Notes

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

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

References

[Fai13c], [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)
CAM_Specification_Nayatani95(L_star_P=49.9998829..., C=0.0133550..., h=257.5232268...
↪, s=0.0133550..., Q=62.6266734..., M=0.0167262..., H=None, HC=None, L_star_N=50.
↪0039154...)
```

colour.CAM_Specification_Nayatani95

class colour.**CAM_Specification_Nayatani95**(*L_star_P: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>, L_star_N: float | NDArrayFloat | None = <factory>*)

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

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

Parameters

- **L_star_P** (float | NDArrayFloat | None) – Correlate of *achromatic Lightness* $L_p^\star$.
- **C** (float | NDArrayFloat | None) – Correlate of *chroma* C.
- **h** (float | NDArrayFloat | None) – *Hue* angle θ in degrees.
- **s** (float | NDArrayFloat | None) – Correlate of *saturation* S.
- **Q** (float | NDArrayFloat | None) – Correlate of *brightness* B_r.
- **M** (float | NDArrayFloat | None) – Correlate of *colourfulness* M.
- **H** (float | NDArrayFloat | None) – *Hue* h quadrature H.

- **HC** (float | NDArrayFloat | None) – *Hue* h composition H_C.
- **L_star_N** (float | NDArrayFloat | None) – Correlate of *normalised achromatic Lightness* $L_n^\star$.

Return type None

Notes

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

References

[Fai13c], [NSY95]

__init__(*L_star_P: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>, L_star_N: float | NDArrayFloat | None = <factory>*) → None

Parameters

- **L_star_P** (float | NDArrayFloat | None) –
- **C** (float | NDArrayFloat | None) –
- **h** (float | NDArrayFloat | None) –
- **s** (float | NDArrayFloat | None) –
- **Q** (float | NDArrayFloat | None) –
- **M** (float | NDArrayFloat | None) –
- **H** (float | NDArrayFloat | None) –
- **HC** (float | NDArrayFloat | None) –
- **L_star_N** (float | NDArrayFloat | None) –

Return type None

Methods

`__init__`([L_star_P, C, h, s, Q, M, H, HC, ...])	
`arithmetical_operation`(a, operation[, in_place])	Perform given arithmetical operation with a operand on the `dataclass`-like class.

Attributes

`fields`	Getter property for the fields of the `dataclass`-like class.
`items`	Getter property for the `dataclass`-like class items, i.e. the field names and values.
`keys`	Getter property for the `dataclass`-like class keys, i.e. the field names.
`values`	Getter property for the `dataclass`-like class values, i.e. the field values.
`L_star_P`	
`C`	
`h`	
`s`	
`Q`	
`M`	
`H`	
`HC`	
`L_star_N`	

RLAB

`colour`

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

colour.XYZ_to_RLAB

`colour.`**`XYZ_to_RLAB`**(*XYZ: ArrayLike, XYZ_n: ArrayLike, Y_n: ArrayLike, sigma: ArrayLike = VIEWING_CONDITIONS_RLAB['Average'], D: ArrayLike = D_FACTOR_RLAB['Hard Copy Images']*) → *colour.appearance.rlab.CAM_Specification_RLAB*

Compute the *RLAB* model color appearance correlates.

Parameters

- **XYZ** (`ArrayLike`) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_n** (`ArrayLike`) – *CIE XYZ* tristimulus values of reference white.
- **Y_n** (`ArrayLike`) – Absolute adapting luminance in cd/m^2.

- **sigma** (ArrayLike) – Relative luminance of the surround, see colour.VIEWING_CONDITIONS_RLAB for reference.
- **D** (ArrayLike) – *Discounting-the-Illuminant* factor normalised to domain [0, 1].

Returns *RLAB* colour appearance model specification.

Return type *CAM_Specification_RLAB*

Notes

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

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

References

[Fai96], [Fai13d]

Examples

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

colour.CAM_Specification_RLAB

class colour.**CAM_Specification_RLAB**(*J: NDArrayFloat | None = <factory>, C: NDArrayFloat | None = <factory>, h: NDArrayFloat | None = <factory>, s: NDArrayFloat | None = <factory>, HC: NDArrayFloat | None = <factory>, a: NDArrayFloat | None = <factory>, b: NDArrayFloat | None = <factory>*)

Define 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** (NDArrayFloat | None) – Correlate of *Lightness* L^R.
- **C** (NDArrayFloat | None) – Correlate of *achromatic chroma* C^R.
- **h** (NDArrayFloat | None) – *Hue* angle h^R in degrees.
- **s** (NDArrayFloat | None) – Correlate of *saturation* s^R.
- **HC** (NDArrayFloat | None) – *Hue* h composition H^C.

- **a** (`NDArrayFloat | None`) – Red-green chromatic response a^R.
- **b** (`NDArrayFloat | None`) – Yellow-blue chromatic response b^R.

Return type None

Notes

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

References

[Fai96], [Fai13d]

`__init__`(*J: NDArrayFloat | None = <factory>, C: NDArrayFloat | None = <factory>, h: NDArrayFloat | None = <factory>, s: NDArrayFloat | None = <factory>, HC: NDArrayFloat | None = <factory>, a: NDArrayFloat | None = <factory>, b: NDArrayFloat | None = <factory>*) → None

Parameters

- **J** (`NDArrayFloat | None`) –
- **C** (`NDArrayFloat | None`) –
- **h** (`NDArrayFloat | None`) –
- **s** (`NDArrayFloat | None`) –
- **HC** (`NDArrayFloat | None`) –
- **a** (`NDArrayFloat | None`) –
- **b** (`NDArrayFloat | None`) –

Return type None

Methods

`__init__`([J, C, h, s, HC, a, b])	

Attributes

fields	Getter property for the fields of the dataclass-like class.
items	Getter property for the dataclass-like class items, i.e. the field names and values.
keys	Getter property for the dataclass-like class keys, i.e. the field names.
values	Getter property for the dataclass-like class values, i.e. the field values.
J	
C	
h	
s	
HC	
a	
b	

colour.VIEWING_CONDITIONS_RLAB

```
colour.VIEWING_CONDITIONS_RLAB = CanonicalMapping({'Average': ..., 'Dim': ..., 'Dark':
...})
```

Reference *RLAB* colour appearance model viewing conditions.

References

[Fai96], [Fai13d]

Ancillary Objects

colour.appearance

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

colour.appearance.D_FACTOR_RLAB

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

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

References

[Fai96], [Fai13d]

Aliases:

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

ZCAM

colour

XYZ_to_ZCAM(XYZ, XYZ_w, L_A, Y_b[, ...])	Compute the *ZCAM* colour appearance model correlates from given *CIE XYZ* tristimulus values.
ZCAM_to_XYZ(specification, XYZ_w, L_A, Y_b)	Convert from *ZCAM* specification to *CIE XYZ* tristimulus values.
CAM_Specification_ZCAM(J, C, h, s, Q, M, H, ...)	Define the *ZCAM* colour appearance model specification.
VIEWING_CONDITIONS_ZCAM	Reference *ZCAM* colour appearance model viewing conditions.

colour.XYZ_to_ZCAM

colour.**XYZ_to_ZCAM**(*XYZ: ArrayLike, XYZ_w: ArrayLike, L_A: ArrayLike, Y_b: ArrayLike, surround: colour.appearance.zcam.InductionFactors_ZCAM = VIEWING_CONDITIONS_ZCAM['Average'], discount_illuminant: bool = False, compute_H: bool = True*) → *colour.appearance.zcam.CAM_Specification_ZCAM*

Compute the *ZCAM* colour appearance model correlates from given *CIE XYZ* tristimulus values.

Parameters

- **XYZ** (ArrayLike) – Absolute *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (ArrayLike) – Absolute *CIE XYZ* tristimulus values of the white under reference illuminant.
- **L_A** (ArrayLike) – Test adapting field *luminance* L_A in cd/m^2 such as $L_A = L_w * Y_b/100$ (where L_w is luminance of the reference white and Y_b is the background luminance factor).
- **Y_b** (ArrayLike) – Luminous factor of background Y_b such as $Y_b = 100 * L_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (colour.appearance.zcam.InductionFactors_ZCAM) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool) – Truth value indicating if the illuminant should be discounted.
- **compute_H** (bool) – Whether to compute *Hue* h quadrature H. H is rarely used, and expensive to compute.

Returns *ZCAM* colour appearance model specification.

Return type colour.CAM_Specification_ZCAM

> **Warning:** The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- *Safdar, Hardeberg and Luo (2021)* does not specify how the chromatic adaptation to *CIE Standard Illuminant D65* in *Step 0* should be performed. A one-step *Von Kries* chromatic adaptation transform is not symmetrical or transitive when a degree of adaptation is involved. *Safdar, Hardeberg and Luo (2018)* uses *Zhai and Luo (2018)* two-steps chromatic adaptation transform, thus it seems sensible to adopt this transform for the *ZCAM* colour appearance model until more information is available. It is worth noting that a one-step *Von Kries* chromatic adaptation transform with support for degree of adaptation produces values closer to the supplemental document compared to the *Zhai and Luo (2018)* two-steps chromatic adaptation transform but then the *ZCAM* colour appearance model does not round-trip properly.
- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
XYZ	[UN]	[UN]
XYZ_tw	[UN]	[UN]
XYZ_rw	[UN]	[UN]

Range	Scale - Reference	Scale - 1
CAM_Specification_ZCAM.J	[UN]	[0, 1]
CAM_Specification_ZCAM.C	[UN]	[0, 1]
CAM_Specification_ZCAM.h	[0, 360]	[0, 1]
CAM_Specification_ZCAM.s	[UN]	[0, 1]
CAM_Specification_ZCAM.Q	[UN]	[0, 1]
CAM_Specification_ZCAM.M	[UN]	[0, 1]
CAM_Specification_ZCAM.H	[0, 400]	[0, 1]
CAM_Specification_ZCAM.HC	[UN]	[0, 1]
CAM_Specification_ZCAM.V	[UN]	[0, 1]
CAM_Specification_ZCAM.K	[UN]	[0, 1]
CAM_Specification_ZCAM.H	[UN]	[0, 1]

References

[SHKL18], [SHRL21], [ZL18]

Examples

```
>>> XYZ = np.array([185, 206, 163])
>>> XYZ_w = np.array([256, 264, 202])
>>> L_A = 264
>>> Y_b = 100
>>> surround = VIEWING_CONDITIONS_ZCAM["Average"]
>>> XYZ_to_ZCAM(XYZ, XYZ_w, L_A, Y_b, surround)
...
CAM_Specification_ZCAM(J=92.2504437..., C=3.0216926..., h=196.3245737..., s=19.
↪1319556..., Q=321.3408463..., M=10.5256217..., H=237.6114442..., HC=None, V=34.
↪7006776..., K=25.8835968..., W=91.6821728...)
```

(continues on next page)

(continued from previous page)

colour.ZCAM_to_XYZ

colour.**ZCAM_to_XYZ**(*specification: colour.appearance.zcam.CAM_Specification_ZCAM, XYZ_w: ArrayLike, L_A: ArrayLike, Y_b: ArrayLike, surround: colour.appearance.zcam.InductionFactors_ZCAM = VIEWING_CONDITIONS_ZCAM['Average'], discount_illuminant: bool = False*) → NDArrayFloat

Convert from *ZCAM* specification to *CIE XYZ* tristimulus values.

Parameters

- **specification** (colour.appearance.zcam.CAM_Specification_ZCAM) – *ZCAM* 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** (ArrayLike) – Absolute *CIE XYZ* tristimulus values of the white under reference illuminant.
- **L_A** (ArrayLike) – Test adapting field *luminance* L_A in cd/m^2 such as $L_A = L_w * Y_b/100$ (where L_w is luminance of the reference white and Y_b is the background luminance factor).
- **Y_b** (ArrayLike) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (colour.appearance.zcam.InductionFactors_ZCAM) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool) – Truth value indicating if the illuminant should be discounted.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

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

Warning: The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- *Safdar, Hardeberg and Luo (2021)* does not specify how the chromatic adaptation to *CIE Standard Illuminant D65* in *Step 0* should be performed. A one-step *Von Kries* chromatic adaptation transform is not symetrical or transitive when a degree of adptation is involved. *Safdar, Hardeberg and Luo (2018)* uses *Zhai and Luo (2018)* two-steps chromatic adaptation transform, thus it seems sensible to adopt this transform for the *ZCAM* colour appearance model until more information is available. It is worth noting that a one-step *Von Kries* chromatic adaptation transform with support for degree of adaptation produces values closer to the supplemental document compared to the *Zhai and Luo (2018)* two-steps chromatic adaptation transform but then the *ZCAM* colour appearance model does not round-trip properly.

- *Step 4* of the inverse model uses a rounded exponent of 1.3514 preventing the model to round-trip properly. Given that this implementation takes some liberties with respect to the chromatic adaptation transform to use, it was deemed appropriate to use an exponent value that enables the *ZCAM* colour appearance model to round-trip.
- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
`CAM_Specification_ZCAM.J`	[UN]	[0, 1]
`CAM_Specification_ZCAM.C`	[UN]	[0, 1]
`CAM_Specification_ZCAM.h`	[0, 360]	[0, 1]
`CAM_Specification_ZCAM.s`	[UN]	[0, 1]
`CAM_Specification_ZCAM.Q`	[UN]	[0, 1]
`CAM_Specification_ZCAM.M`	[UN]	[0, 1]
`CAM_Specification_ZCAM.H`	[0, 400]	[0, 1]
`CAM_Specification_ZCAM.HC`	[UN]	[0, 1]
`CAM_Specification_ZCAM.V`	[UN]	[0, 1]
`CAM_Specification_ZCAM.K`	[UN]	[0, 1]
`CAM_Specification_ZCAM.H`	[UN]	[0, 1]

Range	Scale - Reference	Scale - 1
`XYZ`	[UN]	[UN]

References

[SHKL18], [SHRL21], [ZL18]

Examples

```
>>> specification = CAM_Specification_ZCAM(
...     J=92.250443780723629, C=3.0216926733329013, h=196.32457375575581
... )
>>> XYZ_w = np.array([256, 264, 202])
>>> L_A = 264
>>> Y_b = 100
>>> surround = VIEWING_CONDITIONS_ZCAM["Average"]
>>> ZCAM_to_XYZ(specification, XYZ_w, L_A, Y_b, surround)
...
array([ 185.,  206.,  163.])
```

colour.CAM_Specification_ZCAM

class colour.**CAM_Specification_ZCAM**(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>, V: float | NDArrayFloat | None = <factory>, K: float | NDArrayFloat | None = <factory>, W: float | NDArrayFloat | None = <factory>*)

Define the *ZCAM* colour appearance model specification.

Parameters

- **J** (float | NDArrayFloat | None) – *Lightness* J is the "brightness of an area (Q) judged relative to the brightness of a similarly illuminated area that appears to be white or highly transmitting (Q_w)", i.e., $J = (Q/Q_w)$. It is a visual scale with two well defined levels i.e., zero and 100 for a pure black and a reference white, respectively. Note that in HDR visual field, samples could have a higher luminance than that of the reference white, so the lightness could be over 100. Subscripts s and w are used to annotate the sample and the reference white, respectively.
- **C** (float | NDArrayFloat | None) – *Chroma* C is "colourfulness of an area (M) judged as a proportion of the brightness of a similarly illuminated area that appears white or highly transmitting (Q_w)", i.e., $C = (M/Q_w)$. It is an open-end scale with origin as a colour in the neutral axis. It can be estimated as the magnitude of the chromatic difference between the test colour and a neutral colour having the lightness same as the test colour.
- **h** (float | NDArrayFloat | None) – *Hue* angle h is a scale ranged from $0°$ to $360°$ with the hues following rainbow sequence. The same distance between pairs of hues in a constant lightness and chroma shows the same perceived colour difference.
- **s** (float | NDArrayFloat | None) – *Saturation* s is the "colourfulness (M) of an area judged in proportion to its brightness (Q)", i.e., $s = (M/Q)$. It can also be defined as the chroma of an area judged in proportion to its lightness, i.e., $s = (C/J)$. It is an open-end scale with all neutral colours to have saturation of zero. For example, the red bricks in a building would exhibit different colours when illuminated by daylight. Those (directly) under daylight will appear to be bright and colourful, and those under shadow will appear darker and less colourful. However, the two areas have the same saturation.
- **Q** (float | NDArrayFloat | None) – *Brightness* Q is an "attribute of a visual perception according to which an area appears to emit, or reflect, more or less light". It is an open-end scale with origin as pure black or complete darkness. It is an absolute scale according to the illumination condition i.e., an increase of brightness of an object when the illuminance of light is increased. This is a visual phenomenon known as Stevens effect.
- **M** (float | NDArrayFloat | None) – *Colourfulness* M is an "attribute of a visual perception according to which the perceived colour of an area appears to be more or less chromatic". It is an open-end scale with origin as a neutral colour i.e., appearance of no hue. It is an absolute scale according to the illumination condition i.e., an increase of colourfulness of an object when the illuminance of light is increased. This is a visual phenomenon known as Hunt effect.
- **H** (float | NDArrayFloat | None) – *Hue* h quadrature H_C is an "attribute of a visual perception according to which an area appears to be similar to one of the colours: red, yellow, green, and blue, or to a combination of adjacent pairs of these colours considered in a closed ring". It has a 0-400 scale, i.e., hue quadrature of 0, 100, 200, 300, and 400 range from unitary red to, yellow, green, blue, and back to red, respectively. For example, a cyan colour consists of 50% green and 50% blue, corresponding to a hue quadrature of 250.
- **HC** (float | NDArrayFloat | None) – *Hue* h composition H^C used to define the hue appearance of a sample. Note that hue circles formed by the equal hue angle and equal hue composition appear to be quite different.
- **V** (float | NDArrayFloat | None) – *Vividness* V is an "attribute of colour used to indicate the degree of departure of the colour (of stimulus) from a neutral black

colour", i.e., $V = \sqrt{J^2 + C^2}$. It is an open-end scale with origin at pure black. This reflects the visual phenomena of an object illuminated by a light to increase both the lightness and the chroma.

- **K** (float | NDArrayFloat | None) – *Blackness* K is a visual attribute according to which an area appears to contain more or less black content. It is a scale in the Natural Colour System (NCS) and can also be defined in resemblance to a pure black. It is an open-end scale with 100 as pure black (luminance of 0 cd/m^2), i.e., $K = (100 - \sqrt{J^2 + C^2} = (100 - V)$. The visual effect can be illustrated by mixing a black to a colour pigment. The more black pigment is added, the higher blackness will be. A blacker colour will have less lightness and/or chroma than a less black colour.
- **W** (float | NDArrayFloat | None) – *Whiteness* W is a visual attribute according to which an area appears to contain more or less white content. It is a scale of the NCS and can also be defined in resemblance to a pure white. It is an open-end scale with 100 as reference white, i.e., $W = (100 - \sqrt{(100 - J)^2 + C^2} = (100 - D)$. The visual effect can be illustrated by mixing a white to a colour pigment. The more white pigment is added, the higher whiteness will be. A whiter colour will have a lower chroma and higher lightness than the less white colour.

Return type None

References

[SHRL21]

`__init__`(*J: float | NDArrayFloat | None = <factory>, C: float | NDArrayFloat | None = <factory>, h: float | NDArrayFloat | None = <factory>, s: float | NDArrayFloat | None = <factory>, Q: float | NDArrayFloat | None = <factory>, M: float | NDArrayFloat | None = <factory>, H: float | NDArrayFloat | None = <factory>, HC: float | NDArrayFloat | None = <factory>, V: float | NDArrayFloat | None = <factory>, K: float | NDArrayFloat | None = <factory>, W: float | NDArrayFloat | None = <factory>*) → None

Parameters

- **J** (float | NDArrayFloat | None) –
- **C** (float | NDArrayFloat | None) –
- **h** (float | NDArrayFloat | None) –
- **s** (float | NDArrayFloat | None) –
- **Q** (float | NDArrayFloat | None) –
- **M** (float | NDArrayFloat | None) –
- **H** (float | NDArrayFloat | None) –
- **HC** (float | NDArrayFloat | None) –
- **V** (float | NDArrayFloat | None) –
- **K** (float | NDArrayFloat | None) –
- **W** (float | NDArrayFloat | None) –

Return type None

Methods

`__init__`([J, C, h, s, Q, M, H, HC, V, K, W])	
`arithmetical_operation`(a, operation[, in_place])	Perform given arithmetical operation with a operand on the `dataclass`-like class.

Attributes

`fields`	Getter property for the fields of the `dataclass`-like class.
`items`	Getter property for the `dataclass`-like class items, i.e. the field names and values.
`keys`	Getter property for the `dataclass`-like class keys, i.e. the field names.
`values`	Getter property for the `dataclass`-like class values, i.e. the field values.
`J`	
`C`	
`h`	
`s`	
`Q`	
`M`	
`H`	
`HC`	
`V`	
`K`	
`W`	

colour.VIEWING_CONDITIONS_ZCAM

colour.**VIEWING_CONDITIONS_ZCAM = CanonicalMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**

Reference *ZCAM* colour appearance model viewing conditions.

References

[SHRL21]

Ancillary Objects

colour.appearance

InductionFactors_ZCAM(F_s, F, c, N_c)	*ZCAM* colour appearance model induction factors.

colour.appearance.InductionFactors_ZCAM

class colour.appearance.**InductionFactors_ZCAM**(*F_s, F, c, N_c*)

ZCAM colour appearance model induction factors.

Parameters

- **F_s** – Surround impact F_s.
- **F** – Maximum degree of adaptation F.
- **c** – Exponential non-linearity c.
- **N_c** – Chromatic induction factor N_c.

Notes

- The *ZCAM* colour appearance model induction factors are inherited from the *CIECAM02* colour appearance model.

References

[SHRL21]

Create new instance of InductionFactors_ZCAM(F_s, F, c, N_c)

__init__()

Methods

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

Attributes

F	Alias for field number 1
F_s	Alias for field number 0
N_c	Alias for field number 3
c	Alias for field number 2

Helmholtz-Kohlrausch Effect Estimation

colour

HKE_NAYATANI1997_METHODS	*Nayatani (1997) HKE* computation methods, choice between variable achromatic colour ('VAC') and variable chromatic colour ('VCC')
HelmholtzKohlrausch_effect_object_Nayatani1997(uv, ...)	Return the *HKE* value for object colours using *Nayatani (1997)* method.
HelmholtzKohlrausch_effect_luminous_Nayatani1997(uv, ...)	Return the *HKE* factor for luminous colours using *Nayatani (1997)* method.

colour.HKE_NAYATANI1997_METHODS

colour.**HKE_NAYATANI1997_METHODS = CanonicalMapping({'VAC': ..., 'VCC': ...})**

Nayatani (1997) HKE computation methods, choice between variable achromatic colour ('VAC') and variable chromatic colour ('VCC')

References

[Nay97]

colour.HelmholtzKohlrausch_effect_object_Nayatani1997

colour.**HelmholtzKohlrausch_effect_object_Nayatani1997**(*uv: ArrayLike, uv_c: ArrayLike, L_a: ArrayLike, method: Union[Literal['VAC', 'VCC'], str] = 'VCC'*) → NDArrayFloat

Return the *HKE* value for object colours using *Nayatani (1997)* method.

Parameters

- **uv** (ArrayLike) – *CIE uv* chromaticity coordinates of samples.
- **uv_c** (ArrayLike) – *CIE uv* chromaticity coordinates of reference white.
- **L_a** (ArrayLike) – Adapting luminance in cd/m^2.
- **method** (Union[Literal['VAC', 'VCC'], str]) – Which estimation method to use, *VCC* or *VAC*.

Returns Luminance factor (Γ) value(s) computed with Nayatani object colour estimation method.

Return type numpy.ndarray

References

[Nay97]

Examples

```
>>> import colour
>>> white = colour.xy_to_Luv_uv(colour.temperature.CCT_to_xy_CIE_D(6504))
>>> colours = colour.XYZ_to_xy(
...     [colour.wavelength_to_XYZ(430 + i * 50) for i in range(5)]
... )
>>> L_adapting = 65
>>> HelmholtzKohlrausch_effect_object_Nayatani1997(
...     colour.xy_to_Luv_uv(colours), white, L_adapting
... )
array([ 2.2468383...,  1.4619799...,  1.1801658...,  0.9031318...,  1.7999376...])
```

colour.HelmholtzKohlrausch_effect_luminous_Nayatani1997

colour.**HelmholtzKohlrausch_effect_luminous_Nayatani1997**(*uv: ArrayLike, uv_c: ArrayLike, L_a: ArrayLike, method: Union[Literal['VAC', 'VCC'], str] = 'VCC'*) → NDArrayFloat

Return the *HKE* factor for luminous colours using *Nayatani (1997)* method.

Parameters

- **uv** (ArrayLike) – *CIE uv* chromaticity coordinates of samples.
- **uv_c** (ArrayLike) – *CIE uv* chromaticity coordinates of reference white.
- **L_a** (ArrayLike) – Adapting luminance in cd/m^2.
- **method** (Union[Literal['VAC', 'VCC'], str]) – Which estimation method to use, *VCC* or *VAC*.

Returns Luminance factor (Γ) value(s) computed with Nayatani luminous colour estimation method.

Return type numpy.ndarray

References

[Nay97]

Examples

```
>>> import colour
>>> white = colour.xy_to_Luv_uv(colour.temperature.CCT_to_xy_CIE_D(6504))
>>> colours = colour.XYZ_to_xy(
...     [colour.wavelength_to_XYZ(430 + i * 50) for i in range(5)]
... )
>>> L_adapting = 65
>>> HelmholtzKohlrausch_effect_luminous_Nayatani1997(
...     colour.xy_to_Luv_uv(colours), white, L_adapting
... )
array([ 7.4460471...,  2.4767159...,  1.4723422...,  0.7938695...,  4.1828629...])
```

Ancillary Objects

colour.appearance

coefficient_q_Nayatani1997(theta)	Return the $q(\theta)$ coefficient for *Nayatani (1997) HKE* computations.
coefficient_K_Br_Nayatani1997(L_a)	Return the K_{Br} coefficient for *Nayatani (1997) HKE* computations.

colour.appearance.coefficient_q_Nayatani1997

colour.appearance.**coefficient_q_Nayatani1997**(*theta: ArrayLike*) → NDArrayFloat

Return the $q(\theta)$ coefficient for *Nayatani (1997) HKE* computations.

The hue angle θ can be computed as follows:

$$tan^{-1}\frac{v'-v'_c}{u'-u'_c}$$

where u' and v' are the CIE 1976 chromaticity coordinates of the test chromatic light and u'_c and v'_c are the CIE 1976 chromaticity coordinates of the reference white light.

Parameters **theta** (ArrayLike) – Hue angle (θ) in radians.

Returns q coefficient for *Nayatani (1997) HKE* methods.

Return type numpy.ndarray

References

[Nay97]

Examples

This recreates *FIG. A-1*.

```
>>> import matplotlib.pyplot as plt
>>> angles = [(np.pi * 2 / 100 * i) for i in range(100)]
>>> q_values = coefficient_q_Nayatani1997(angles)
>>> plt.plot(np.array(angles), q_values / (np.pi * 2) * 180)
...
[<matplotlib.lines.Line2D object at 0x...>]
>>> plt.show()
```

colour.appearance.coefficient_K_Br_Nayatani1997

colour.appearance.**coefficient_K_Br_Nayatani1997**(*L_a: ArrayLike*) → NDArrayFloat

Return the K_{Br} coefficient for *Nayatani (1997) HKE* computations.

Parameters **L_a** (ArrayLike) – Adapting luminance in cd/m^2.

Returns K_{Br} coefficient for *Nayatani (1997) HKE* methods.

Return type numpy.ndarray

Notes

- The K_{Br} coefficient is normalised to unity around $63.66cd/m^2$.

References

[Nay97]

Examples

```
>>> L_a_values = [10 + i * 20 for i in range(5)]
>>> coefficient_K_Br_Nayatani1997(L_a_values)
array([ 0.7134481...,  0.8781172...,  0.9606248...,  1.0156689...,  1.0567008...])
>>> coefficient_K_Br_Nayatani1997(63.66)
1.0001284...
```

Biochemistry

Michaelis–Menten Kinetics

colour.biochemistry

REACTION_RATE_MICHAELISMENTEN_METHODS	Supported *Michaelis-Menten* kinetics reaction rate equation computation methods.
reaction_rate_MichaelisMenten(S, V_max, K_m)	Describe the rate of enzymatic reactions, by relating reaction rate v to concentration of a substrate S according to given method.
SUBSTRATE_CONCENTRATION_MICHAELISMENTEN_METHODS	Supported *Michaelis-Menten* kinetics substrate concentration equation computation methods.
substrate_concentration_MichaelisMenten(v, ...)	Describe the rate of enzymatic reactions, by relating concentration of a substrate S to reaction rate v according to given method.
reaction_rate_MichaelisMenten_Michaelis1913(S, ...)	Describe the rate of enzymatic reactions, by relating reaction rate v to concentration of a substrate S.
substrate_concentration_MichaelisMenten_Michaelis1913(v, ...)	Describe the rate of enzymatic reactions, by relating concentration of a substrate S to reaction rate v.
reaction_rate_MichaelisMenten_Abebe2017(S, ...)	Describe the rate of enzymatic reactions, by relating reaction rate v to concentration of a substrate S according to the modified *Michaelis-Menten* kinetics equation as given by *Abebe, Pouli, Larabi and Reinhard (2017)*.
substrate_concentration_MichaelisMenten_Abebe2017(v, ...)	Describe the rate of enzymatic reactions, by relating concentration of a substrate S to reaction rate v according to the modified *Michaelis-Menten* kinetics equation as given by *Abebe, Pouli, Larabi and Reinhard (2017)*.

colour.biochemistry.REACTION_RATE_MICHAELISMENTEN_METHODS

`colour.biochemistry.REACTION_RATE_MICHAELISMENTEN_METHODS = CanonicalMapping({'Michaelis 1913': ..., 'Abebe 2017': ...})`

Supported *Michaelis-Menten* kinetics reaction rate equation computation methods.

References

[Wikipedia03d], [APLR17]

colour.biochemistry.reaction_rate_MichaelisMenten

`colour.biochemistry.reaction_rate_MichaelisMenten`(*S: ArrayLike, V_max: ArrayLike, K_m: ArrayLike, method: Union[Literal['Michaelis 1913', 'Abebe 2017'], str] = 'Michaelis 1913', **kwargs: Any*) → NDArrayFloat

Describe the rate of enzymatic reactions, by relating reaction rate v to concentration of a substrate S according to given method.

Parameters

- **S** (`ArrayLike`) – Concentration of a substrate S.
- **V_max** (`ArrayLike`) – Maximum rate V_{max} achieved by the system, at saturating substrate concentration.
- **K_m** (`ArrayLike`) – Substrate concentration K_m at which the reaction rate is half of V_{max}.
- **method** (`Union[Literal['Michaelis 1913', 'Abebe 2017'], str]`) – Computation method.
- **b_m** – {`colour.biochemistry.reaction_rate_MichaelisMenten_Abebe2017()`}, Bias factor b_m.
- **kwargs** (`Any`) –

Returns Reaction rate v.

Return type `numpy.ndarray`

References

[Wikipedia03d], [APLR17]

Examples

```
>>> reaction_rate_MichaelisMenten(0.5, 2.5, 0.8)
0.9615384...
>>> reaction_rate_MichaelisMenten(
...     0.5, 2.5, 0.8, method="Abebe 2017", b_m=0.813
... )
1.0360547...
```

colour.biochemistry.SUBSTRATE_CONCENTRATION_MICHAELISMENTEN_METHODS

colour.biochemistry.**SUBSTRATE_CONCENTRATION_MICHAELISMENTEN_METHODS = CanonicalMapping({'Michaelis 1913': ..., 'Abebe 2017': ...})**

Supported *Michaelis-Menten* kinetics substrate concentration equation computation methods.

References

[Wikipedia03d], [APLR17]

colour.biochemistry.substrate_concentration_MichaelisMenten

colour.biochemistry.**substrate_concentration_MichaelisMenten**(*v: ArrayLike*, *V_max: ArrayLike*, *K_m: ArrayLike*, *method: Union[Literal['Michaelis 1913', 'Abebe 2017'], str] = 'Michaelis 1913'*, ***kwargs: Any*) → NDArrayFloat

Describe the rate of enzymatic reactions, by relating concentration of a substrate S to reaction rate v according to given method.

Parameters

- **v** (ArrayLike) – Reaction rate v.
- **V_max** (ArrayLike) – Maximum rate V_{max} achieved by the system, at saturating substrate concentration.
- **K_m** (ArrayLike) – Substrate concentration K_m at which the reaction rate is half of V_{max}.
- **method** (Union[Literal['Michaelis 1913', 'Abebe 2017'], str]) – Computation method.
- **b_m** – {colour.biochemistry.substrate_concentration_MichaelisMenten_Abebe2017()}, Bias factor b_m.
- **kwargs** (Any) –

Returns Concentration of a substrate S.

Return type numpy.ndarray

References

[Wikipedia03d], [APLR17]

Examples

```
>>> substrate_concentration_MichaelisMenten(0.961538461538461, 2.5, 0.8)
...
0.4999999...
>>> substrate_concentration_MichaelisMenten(
...     1.036054703688355, 2.5, 0.8, method="Abebe 2017", b_m=0.813
... )
...
0.5000000...
```

colour.biochemistry.reaction_rate_MichaelisMenten_Michaelis1913

`colour.biochemistry.reaction_rate_MichaelisMenten_Michaelis1913(S: ArrayLike, V_max: ArrayLike, K_m: ArrayLike) → NDArrayFloat`

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

Parameters

- **S** (ArrayLike) – Concentration of a substrate S.
- **V_max** (ArrayLike) – Maximum rate V_{max} achieved by the system, at saturating substrate concentration.
- **K_m** (ArrayLike) – Substrate concentration K_m at which the reaction rate is half of V_{max}.

Returns Reaction rate v.

Return type numpy.ndarray

References

[Wikipedia03d]

Examples

```
>>> reaction_rate_MichaelisMenten(0.5, 2.5, 0.8)
0.9615384...
```

colour.biochemistry.substrate_concentration_MichaelisMenten_Michaelis1913

`colour.biochemistry.substrate_concentration_MichaelisMenten_Michaelis1913(v: ArrayLike, V_max: ArrayLike, K_m: ArrayLike) → NDArrayFloat`

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

Parameters

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

Returns Concentration of a substrate S.

Return type numpy.ndarray

References

[Wikipedia03d]

Examples

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

colour.biochemistry.reaction_rate_MichaelisMenten_Abebe2017

colour.biochemistry.**reaction_rate_MichaelisMenten_Abebe2017**(*S: ArrayLike*, *V_max: ArrayLike*, *K_m: ArrayLike*, *b_m: ArrayLike*) → NDArrayFloat

Describe the rate of enzymatic reactions, by relating reaction rate v to concentration of a substrate S according to the modified *Michaelis-Menten* kinetics equation as given by *Abebe, Pouli, Larabi and Reinhard (2017)*.

Parameters

- **S** (ArrayLike) – Concentration of a substrate S (or $(\frac{Y}{Y_n})^{\epsilon}$).
- **V_max** (ArrayLike) – Maximum rate V_{max} (or a_m) achieved by the system, at saturating substrate concentration.
- **K_m** (ArrayLike) – Substrate concentration K_m (or c_m) at which the reaction rate is half of V_{max}.
- **b_m** (ArrayLike) – Bias factor b_m.

Returns Reaction rate v.

Return type numpy.ndarray

References

[APLR17]

Examples

```
>>> reaction_rate_MichaelisMenten_Abebe2017(0.5, 1.448, 0.635, 0.813)
...
0.6951512...
```

colour.biochemistry.substrate_concentration_MichaelisMenten_Abebe2017

colour.biochemistry.**substrate_concentration_MichaelisMenten_Abebe2017**(*v: ArrayLike*, *V_max: ArrayLike*, *K_m: ArrayLike*, *b_m: ArrayLike*) → NDArrayFloat

Describe the rate of enzymatic reactions, by relating concentration of a substrate S to reaction rate v according to the modified *Michaelis-Menten* kinetics equation as given by *Abebe, Pouli, Larabi and Reinhard (2017)*.

Parameters

- **v** (ArrayLike) – Reaction rate v.
- **V_max** (ArrayLike) – Maximum rate V_{max} (or a_m) achieved by the system, at saturating substrate concentration.
- **K_m** (ArrayLike) – Substrate concentration K_m (or c_m) at which the reaction rate is half of V_{max}.
- **b_m** (ArrayLike) – Bias factor b_m.

Returns Concentration of a substrate S.

Return type numpy.ndarray

References

[APLR17]

Examples

```
>>> substrate_concentration_MichaelisMenten_Abebe2017(
...     0.695151224195871, 1.448, 0.635, 0.813
... )
0.4999999...
```

Colour Vision Deficiency

Machado, Oliveira and Fernandes (2009)

colour

msds_cmfs_anomalous_trichromacy_Machado2009(...)	Shift given *LMS* cone fundamentals colour matching functions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy using *Machado et al. (2009)* method.
matrix_anomalous_trichromacy_Machado2009(...)	Compute the *Machado et al. (2009) CVD* matrix for given *LMS* cone fundamentals colour matching functions and display primaries tri-spectral distributions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy.
matrix_cvd_Machado2009(deficiency, severity)	Compute *Machado et al. (2009) CVD* matrix for given deficiency and severity using the pre-computed matrices dataset.

colour.msds_cmfs_anomalous_trichromacy_Machado2009

colour.**msds_cmfs_anomalous_trichromacy_Machado2009**(*cmfs: colour.colorimetry.cmfs.LMS_ConeFundamentals, d_LMS: ArrayLike*) → *colour.colorimetry.cmfs.LMS_ConeFundamentals*

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

Parameters

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

Return type *colour.colorimetry.cmfs.LMS_ConeFundamentals*

Notes

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

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

Return type `colour.LMS_ConeFundamentals`

Parameters

- **cmfs** (`colour.colorimetry.cmfs.LMS_ConeFundamentals`) –
- **d_LMS** (`ArrayLike`) –

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

References

[Colblindorb], [Colblindora], [Colblindorc], [MOF09]

Examples

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

colour.matrix_anomalous_trichromacy_Machado2009

colour.**matrix_anomalous_trichromacy_Machado2009**(*cmfs: colour.colorimetry.cmfs.LMS_ConeFundamentals*, *primaries: colour.characterisation.displays.RGB_DisplayPrimaries*, *d_LMS: ArrayLike*) → NDArrayFloat

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

Parameters

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

Return type NDArrayFloat

Notes

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

Returns Anomalous trichromacy matrix.

Return type numpy.ndarray

Parameters

- **cmfs** (colour.colorimetry.cmfs.LMS_ConeFundamentals) –
- **primaries** (colour.characterisation.displays.RGB_DisplayPrimaries) –
- **d_LMS** (ArrayLike) –

References

[Colblindorb], [Colblindora], [Colblindorc], [MOF09]

Examples

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

colour.matrix_cvd_Machado2009

colour.**matrix_cvd_Machado2009**(*deficiency: Union[Literal['Deuteranomaly', 'Protanomaly', 'Tritanomaly'], str], severity: float*) → NDArrayFloat

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

Parameters

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

Returns *CVD* matrix.

Return type numpy.ndarray

References

[Colblindorb], [Colblindora], [Colblindorc], [MOF09]

Examples

```
>>> matrix_cvd_Machado2009("Protanomaly", 0.15)
array([[ 0.7869875...,  0.2694875..., -0.0564735...],
       [ 0.0431695...,  0.933774 ...,  0.023058 ...],
       [-0.004238 ..., -0.0024515...,  1.0066895...]])
```

Dataset

colour

CVD_MATRICES_MACHADO2010	Machado (2010) Simulation matrices Φ_{CVD}.

colour.CVD_MATRICES_MACHADO2010

colour.**CVD_MATRICES_MACHADO2010 = CanonicalMapping({'Protanomaly': ..., 'Deuteranomaly': ..., 'Tritanomaly': ...})**

Machado (2010) Simulation matrices Φ_{CVD}.

Colour Characterisation

ACES Spectral Conversion

colour

sd_to_aces_relative_exposure_values(sd[, ...])	Convert given spectral distribution to *ACES2065-1* colourspace relative exposure values.
sd_to_ACES2065_1(sd[, illuminant, ...])	Convert given spectral distribution to *ACES2065-1* colourspace relative exposure values.

colour.sd_to_aces_relative_exposure_values

colour.**sd_to_aces_relative_exposure_values**(*sd:* colour.colorimetry.spectrum.SpectralDistribution, *illuminant:* colour.colorimetry.spectrum.SpectralDistribution | *None = None, chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02', **kwargs*) → NDArrayFloat

Convert given spectral distribution to *ACES2065-1* colourspace relative exposure values.

Parameters

- **sd** (colour.colorimetry.spectrum.SpectralDistribution) – Spectral distribution.
- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution | None) – *Illuminant* spectral distribution, default to *CIE Standard Illuminant D65*.
- **chromatic_adaptation_transform** (Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]) – *Chromatic adaptation* transform.

Returns *ACES2065-1* colourspace relative exposure values array.

Return type numpy.ndarray

Notes

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

- The chromatic adaptation method implemented here is a bit unusual as it involves building a new colourspace based on *ACES2065-1* colourspace primaries but using the whitepoint of the illuminant that the spectral distribution was measured under.

References

[For18], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

Examples

```
>>> from colour import SDS_COLOURCHECKERS
>>> sd = SDS_COLOURCHECKERS["ColorChecker N Ohta"]["dark skin"]
>>> sd_to_aces_relative_exposure_values(
...     sd, chromatic_adaptation_transform=None
... )
array([ 0.1171814...,  0.0866360...,  0.0589726...])
>>> sd_to_aces_relative_exposure_values(
...     sd, apply_chromatic_adaptation=True
... )
...
array([ 0.1180779...,  0.0869031...,  0.0589125...])
```

colour.sd_to_ACES2065_1

`colour.sd_to_ACES2065_1`(*sd:* colour.colorimetry.spectrum.SpectralDistribution, *illuminant:* colour.colorimetry.spectrum.SpectralDistribution | *None = None*, *chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02', **kwargs*) → NDArrayFloat

Convert given spectral distribution to *ACES2065-1* colourspace relative exposure values.

Parameters

- **sd** (`colour.colorimetry.spectrum.SpectralDistribution`) – Spectral distribution.
- **illuminant** (`colour.colorimetry.spectrum.SpectralDistribution | None`) – *Illuminant* spectral distribution, default to *CIE Standard Illuminant D65*.
- **chromatic_adaptation_transform** (`Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]`) – *Chromatic adaptation* transform.

Returns *ACES2065-1* colourspace relative exposure values array.

Return type `numpy.ndarray`

Notes

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

- The chromatic adaptation method implemented here is a bit unusual as it involves building a new colourspace based on *ACES2065-1* colourspace primaries but using the whitepoint of the illuminant that the spectral distribution was measured under.

References

[For18], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

Examples

```
>>> from colour import SDS_COLOURCHECKERS
>>> sd = SDS_COLOURCHECKERS["ColorChecker N Ohta"]["dark skin"]
>>> sd_to_aces_relative_exposure_values(
...     sd, chromatic_adaptation_transform=None
... )
array([ 0.1171814...,  0.0866360...,  0.0589726...])
>>> sd_to_aces_relative_exposure_values(
...     sd, apply_chromatic_adaptation=True
... )
...
array([ 0.1180779...,  0.0869031...,  0.0589125...])
```

Ancillary Objects

`colour.characterisation`

MSDS_ACES_RICD	Implement support for a camera *RGB* sensitivities.

colour.characterisation.MSDS_ACES_RICD

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

Implement support for a camera *RGB* sensitivities.

Parameters

- **data** – Data to be stored in the multi-spectral distributions.
- **domain** – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths property.
- **labels** – Names to use for the colour.SpectralDistribution class instances.
- **extrapolator** – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **name** – Multi-spectral distributions name.
- **display_labels** – Multi-spectral distributions labels for figures, default to colour.colorimetry.RGB_CameraSensitivities.labels property value.

ACES Input Transform Computation

colour

matrix_idt(sensitivities, illuminant[, ...])	Compute an *Input Device Transform* (IDT) matrix for given camera *RGB* spectral sensitivities, illuminant, training data, standard observer colour matching functions and optimisation settings according to *RAW to ACES* v1 and *P-2013-001* procedures.
camera_RGB_to_ACES2065_1(RGB, B, b[, k, clip])	Convert given camera *RGB* colourspace array to *ACES2065-1* colourspace using the *Input Device Transform* (IDT) matrix B, the white balance multipliers b and the exposure factor k according to *P-2013-001* procedure.

colour.matrix_idt

colour.**matrix_idt**(*sensitivities: RGB_CameraSensitivities, illuminant: SpectralDistribution, training_data: MultiSpectralDistributions | None = None, cmfs: MultiSpectralDistributions | None = None, optimisation_factory: Callable = optimisation_factory_rawtoaces_v1, optimisation_kwargs: dict | None = None, chromatic_adaptation_transform: Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'] | str | None = 'CAT02', additional_data: bool = False*) → Tuple[NDArrayFloat, NDArrayFloat, NDArrayFloat, NDArrayFloat] | Tuple[NDArrayFloat, NDArrayFloat]

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

Parameters

- **sensitivities** (RGB_CameraSensitivities) – Camera *RGB* spectral sensitivities.
- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.
- **training_data** (MultiSpectralDistributions | None) – Training data multi-spectral distributions, defaults to using the *RAW to ACES* v1 190 patches.
- **cmfs** (MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **optimisation_factory** (Callable) – Callable producing the objective function and the *CIE XYZ* to optimisation colour model function.
- **optimisation_kwargs** (dict | None) – Parameters for scipy.optimize.minimize() definition.
- **chromatic_adaptation_transform** (Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'] | str | None) – *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **additional_data** (bool) – If *True*, the *XYZ* and *RGB* tristimulus values are also returned.

Returns Tuple of IDT matrix and white balance multipliers or tuple of IDT matrix, white balance multipliers, *XYZ* and *RGB* tristimulus values.

Return type tuple

References

[DFI+17], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee15]

Examples

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

```
>>> path = os.path.join(
...     ROOT_RESOURCES_RAWTOACES,
...     "CANON_EOS_5DMark_II_RGB_Sensitivities.csv",
... )
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values()
... )
>>> illuminant = SDS_ILLUMINANTS["D55"]
>>> M, RGB_w = matrix_idt(sensitivities, illuminant)
>>> np.around(M, 3)
array([[ 0.865, -0.026,  0.161],
       [ 0.057,  1.123, -0.18 ],
       [ 0.024, -0.203,  1.179]])
>>> RGB_w
array([ 2.3414154...,  1.        ,  1.5163375...])
```

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

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

```
>>> M, RGB_w = matrix_idt(
...     sensitivities,
...     illuminant,
...     optimisation_factory=optimisation_factory_Jzazbz,
... )
>>> np.around(M, 3)
array([[ 0.852, -0.009,  0.158],
       [ 0.054,  1.122, -0.176],
       [ 0.023, -0.224,  1.2  ]])
>>> RGB_w
array([ 2.3414154...,  1.        ,  1.5163375...])
```

```
>>> M, RGB_w = matrix_idt(
...     sensitivities,
...     illuminant,
...     optimisation_factory=optimisation_factory_Oklab_15,
... )
>>> np.around(M, 3)
array([[ 0.645, -0.611,  0.107,  0.736,  0.398, -0.275],
       [-0.159,  0.728, -0.091,  0.651,  0.01 , -0.139],
       [-0.172, -0.403,  1.394,  0.51 , -0.295, -0.034]])
>>> RGB_w
array([ 2.3414154...,  1.        ,  1.5163375...])
```

colour.camera_RGB_to_ACES2065_1

colour.**camera_RGB_to_ACES2065_1**(*RGB: ArrayLike*, *B: ArrayLike*, *b: ArrayLike*, *k: ArrayLike = np.ones(3)*, *clip: bool = False*) → NDArrayFloat

Convert given camera *RGB* colourspace array to *ACES2065-1* colourspace using the *Input Device Transform* (IDT) matrix B, the white balance multipliers b and the exposure factor k according to *P-2013-001* procedure.

Parameters

- **RGB** (ArrayLike) – Camera *RGB* colourspace array.
- **B** (ArrayLike) – *Input Device Transform* (IDT) matrix B.
- **b** (ArrayLike) – White balance multipliers b.
- **k** (ArrayLike) – Exposure factor k that results in a nominally "18% gray" object in the scene producing ACES values [0.18, 0.18, 0.18].
- **clip** (bool) – Whether to clip the white balanced camera *RGB* colourspace array between $-\infty$ and 1. The intent is to keep sensor saturated values achromatic after white balancing.

Returns *ACES2065-1* colourspace relative exposure values array.

Return type numpy.ndarray

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee15]

Examples

```
>>> path = os.path.join(
...     ROOT_RESOURCES_RAWTOACES,
...     "CANON_EOS_5DMark_II_RGB_Sensitivities.csv",
... )
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values()
... )
>>> illuminant = SDS_ILLUMINANTS["D55"]
>>> B, b = matrix_idt(sensitivities, illuminant)
>>> camera_RGB_to_ACES2065_1(np.array([0.1, 0.2, 0.3]), B, b)
...
array([ 0.270644 ...,  0.1561487...,  0.5012965...])
```

Ancillary Objects

colour.characterisation

read_training_data_rawtoaces_v1()	Read the *RAW to ACES* v1 190 patches.
generate_illuminants_rawtoaces_v1()	Generate a series of illuminants according to *RAW to ACES* v1:
white_balance_multipliers(sensitivities, ...)	Compute the *RGB* white balance multipliers for given camera *RGB* spectral sensitivities and illuminant.
best_illuminant(RGB_w, sensitivities, ...)	Select the best illuminant for given *RGB* white balance multipliers, and sensitivities in given series of illuminants.
normalise_illuminant(illuminant, sensitivities)	Normalise given illuminant with given camera *RGB* spectral sensitivities.
training_data_sds_to_RGB(training_data, ...)	Convert given training data to *RGB* tristimulus values using given illuminant and given camera *RGB* spectral sensitivities.
training_data_sds_to_XYZ(training_data, ...)	Convert given training data to *CIE XYZ* tristimulus values using given illuminant and given standard observer colour matching functions.
whitepoint_preserving_matrix(M[, RGB_w])	Normalise given matrix M to preserve given white point RGB_w.
optimisation_factory_rawtoaces_v1()	Produce the objective function and *CIE XYZ* colourspace to optimisation colourspace/colour model function according to *RAW to ACES* v1.
optimisation_factory_Jzazbz()	Produce the objective function and *CIE XYZ* colourspace to optimisation colourspace/colour model function based on the $J_z a_z b_z$ colourspace.
optimisation_factory_Oklab_15()	Produce the objective function and *CIE XYZ* colourspace to optimisation colourspace/colour model function based on the *Oklab* colourspace.

colour.characterisation.read_training_data_rawtoaces_v1

colour.characterisation.**read_training_data_rawtoaces_v1**() → *colour.colorimetry.spectrum.MultiSpectralDistributions*

Read the *RAW to ACES* v1 190 patches.

Returns *RAW to ACES* v1 190 patches.

Return type colour.MultiSpectralDistributions

References

[DFI+17]

Examples

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

colour.characterisation.generate_illuminants_rawtoaces_v1

colour.characterisation.**generate_illuminants_rawtoaces_v1**() → *colour.utilities.data_structures.CanonicalMapping*

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

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

Returns Series of illuminants.

Return type `colour.utilities.CanonicalMapping`

Notes

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

References

[DFI+17]

Examples

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

colour.characterisation.white_balance_multipliers

colour.characterisation.**white_balance_multipliers**(*sensitivities: colour.characterisation.cameras.RGB_CameraSensitivities*, *illuminant: colour.colorimetry.spectrum.SpectralDistribution*) → NDArrayFloat

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

Parameters

- **sensitivities** (`colour.characterisation.cameras.RGB_CameraSensitivities`) – Camera *RGB* spectral sensitivities.
- **illuminant** (`colour.colorimetry.spectrum.SpectralDistribution`) – Illuminant spectral distribution.

Returns *RGB* white balance multipliers.

Return type `numpy.ndarray`

References

[DFI+17]

Examples

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

colour.characterisation.best_illuminant

colour.characterisation.**best_illuminant**(*RGB_w: ArrayLike*, *sensitivities:* colour.characterisation.cameras.RGB_CameraSensitivities, *illuminants:* collections.abc.Mapping) → *colour.colorimetry.spectrum.SpectralDistribution*

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

Parameters

- **RGB_w** (ArrayLike) – *RGB* white balance multipliers.
- **sensitivities** (colour.characterisation.cameras.RGB_CameraSensitivities) – Camera *RGB* spectral sensitivities.
- **illuminants** (collections.abc.Mapping) – Illuminant spectral distributions to choose the best illuminant from.

Returns Best illuminant.

Return type colour.SpectralDistribution

Examples

```
>>> path = os.path.join(
...     ROOT_RESOURCES_RAWTOACES,
...     "CANON_EOS_5DMark_II_RGB_Sensitivities.csv",
... )
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values()
... )
>>> illuminants = generate_illuminants_rawtoaces_v1()
>>> RGB_w = white_balance_multipliers(
...     sensitivities, SDS_ILLUMINANTS["FL2"]
... )
>>> best_illuminant(RGB_w, sensitivities, illuminants).name
'D40'
```

colour.characterisation.normalise_illuminant

colour.characterisation.**normalise_illuminant**(*illuminant:* colour.colorimetry.spectrum.SpectralDistribution, *sensitivities:* colour.characterisation.cameras.RGB_CameraSensitivities) → *colour.colorimetry.spectrum.SpectralDistribution*

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

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

Parameters

- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution) – Illuminant spectral distribution.
- **sensitivities** (colour.characterisation.cameras.RGB_CameraSensitivities) – Camera *RGB* spectral sensitivities.

Returns Normalised illuminant.

Return type colour.SpectralDistribution

Examples

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

colour.characterisation.training_data_sds_to_RGB

colour.characterisation.**training_data_sds_to_RGB**(*training_data:* colour.colorimetry.spectrum.MultiSpectralDistributions, *sensitivities:* colour.characterisation.cameras.RGB_CameraSensitivities, *illuminant:* colour.colorimetry.spectrum.SpectralDistribution) → Tuple[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cecd4a850>, <sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cecd98550>]

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

Parameters

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

Returns Tuple of training data *RGB* tristimulus values and white balance multipliers.

Return type tuple

Examples

```
>>> path = os.path.join(
...     ROOT_RESOURCES_RAWTOACES,
...     "CANON_EOS_5DMark_II_RGB_Sensitivities.csv",
... )
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values()
... )
>>> illuminant = normalise_illuminant(
...     SDS_ILLUMINANTS["D55"], sensitivities
... )
>>> training_data = read_training_data_rawtoaces_v1()
>>> RGB, RGB_w = training_data_sds_to_RGB(
...     training_data, sensitivities, illuminant
... )
>>> RGB[:5]
array([[ 0.0207582...,  0.0196857...,  0.0213935...],
       [ 0.0895775...,  0.0891922...,  0.0891091...],
       [ 0.7810230...,  0.7801938...,  0.7764302...],
       [ 0.1995   ...,  0.1995   ...,  0.1995   ...],
       [ 0.5898478...,  0.5904015...,  0.5851076...]])
>>> RGB_w
array([ 2.3414154...,  1.        ,  1.5163375...])
```

colour.characterisation.training_data_sds_to_XYZ

colour.characterisation.**training_data_sds_to_XYZ**(*training_data: colour.colorimetry.spectrum.MultiSpectralDistributions, cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions, illuminant: colour.colorimetry.spectrum.SpectralDistribution, chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02'*) → NDArrayFloat

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

Parameters

- **training_data** (colour.colorimetry.spectrum.MultiSpectralDistributions) – Training data multi-spectral distributions.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions) – Standard observer colour matching functions.
- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution) – Illuminant spectral distribution.
- **chromatic_adaptation_transform** (Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]) – *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.

Returns Training data *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Examples

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

colour.characterisation.whitepoint_preserving_matrix

colour.characterisation.**whitepoint_preserving_matrix**(*M: ArrayLike*, *RGB_w: ArrayLike = ones(3)*) → NDArrayFloat

Normalise given matrix M to preserve given white point RGB_w.

Parameters

- **M** (ArrayLike) – Matrix M to normalise.
- **RGB_w** (ArrayLike) – White point RGB_w to normalise the matrix M with.

Returns Normalised matrix M.

Return type numpy.ndarray

Examples

```
>>> M = np.arange(9).reshape([3, 3])
>>> whitepoint_preserving_matrix(M)
array([[  0.,   1.,   0.],
       [  3.,   4.,  -6.],
       [  6.,   7., -12.]])
```

colour.characterisation.optimisation_factory_rawtoaces_v1

colour.characterisation.**optimisation_factory_rawtoaces_v1**() → Tuple[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cecde5f10>, typing.Callable, typing.Callable, typing.Callable]

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

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

It implements whitepoint preservation as an optimisation constraint.

Returns x_0 initial values, objective function, *CIE XYZ* colourspace to *CIE L*a*b** colourspace function and finaliser function.

Return type tuple

Examples

```
>>> optimisation_factory_rawtoaces_v1()
(array([ 1.,  0.,  0.,  1.,  0.,  0.]), <function optimisation_factory_rawtoaces_v1.
↪<locals> .objective_function at 0x...>, <function optimisation_factory_rawtoaces_
↪v1.<locals>.XYZ_to_optimization_colour_model at 0x...>, <function optimisation_
↪factory_rawtoaces_v1.<locals>.finaliser_function at 0x...>)
```

colour.characterisation.optimisation_factory_Jzazbz

colour.characterisation.**optimisation_factory_Jzazbz**() → Tuple[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cecbf4110>, typing.Callable, typing.Callable, typing.Callable]

Produce the objective function and *CIE XYZ* colourspace to optimisation colourspace/colour model function based on the $J_z a_z b_z$ colourspace.

The objective function returns the Euclidean distance between the training data *RGB* tristimulus values and the training data *CIE XYZ* tristimulus values** in the $J_z a_z b_z$ colourspace.

It implements whitepoint preservation as a post-optimisation step.

Returns x_0 initial values, objective function, *CIE XYZ* colourspace to $J_z a_z b_z$ colourspace function and finaliser function.

Return type tuple

Examples

```
>>> optimisation_factory_Jzazbz()
(array([ 1.,  0.,  0.,  1.,  0.,  0.]), <function optimisation_factory_Jzazbz.
↪<locals>.objective_function at 0x...>, <function optimisation_factory_Jzazbz.
↪<locals>.XYZ_to_optimization_colour_model at 0x...>, <function optimisation_
↪factory_Jzazbz.<locals>.finaliser_function at 0x...>)
```

colour.characterisation.optimisation_factory_Oklab_15

colour.characterisation.**optimisation_factory_Oklab_15**() → Tuple[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cecde6e10>, typing.Callable, typing.Callable, typing.Callable]

Produce the objective function and *CIE XYZ* colourspace to optimisation colourspace/colour model function based on the *Oklab* colourspace.

The objective function returns the Euclidean distance between the training data *RGB* tristimulus values and the training data *CIE XYZ* tristimulus values** in the *Oklab* colourspace.

It implements support for *Finlayson et al. (2015)* root-polynomials of degree 2 and produces 15 terms.

Returns x_0 initial values, objective function, *CIE XYZ* colourspace to *Oklab* colourspace function and finaliser function.

Return type tuple

References

[FMH15]

Examples

```
>>> optimisation_factory_Oklab_15()
(array([ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0., 0.,  1.]),
↪<function optimisation_factory_Oklab_15.<locals>.objective_function at 0x...>,
↪<function optimisation_factory_Oklab_15.<locals>.XYZ_to_optimization_colour_model␣
↪at 0x...>, <function optimisation_factory_Oklab_15.<locals>.finaliser_function at␣
↪0x...>)
```

Colour Fitting

colour

POLYNOMIAL_EXPANSION_METHODS	Supported polynomial expansion methods.
polynomial_expansion(a[, method])	Perform polynomial expansion of given a array.
MATRIX_COLOUR_CORRECTION_METHODS	Supported colour correction matrix methods.
matrix_colour_correction(M_T, M_R[, method])	Compute a colour correction matrix from given M_T colour array to M_R colour array.
APPLY_MATRIX_COLOUR_CORRECTION_METHODS	Supported methods to apply a colour correction matrix.
apply_matrix_colour_correction(RGB, CCM[, ...])	Apply given colour correction matrix CCM to given *RGB* colourspace array.
COLOUR_CORRECTION_METHODS	Supported colour correction methods.
colour_correction(RGB, M_T, M_R[, method])	Perform colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array.

colour.POLYNOMIAL_EXPANSION_METHODS

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

Supported polynomial expansion methods.

References

[CWCR04], [FMH15], [WR04], [Wikipedia03f]

colour.polynomial_expansion

colour.**polynomial_expansion**(*a: ArrayLike, method: Union[Literal['Cheung 2004', 'Finlayson 2015', 'Vandermonde'], str] = 'Cheung 2004', **kwargs: Any*) → NDArrayFloat

Perform polynomial expansion of given a array.

Parameters

- **a** (ArrayLike) – a array to expand.
- **method** (Union[Literal['Cheung 2004', 'Finlayson 2015', 'Vandermonde'], str]) – Computation method.
- **degree** – {colour.characterisation.polynomial_expansion_Finlayson2015(), colour.characterisation.polynomial_expansion_Vandermonde()}, Expanded polynomial degree, must be one of *[1, 2, 3, 4]* for colour.characterisation.polynomial_expansion_Finlayson2015() definition.
- **root_polynomial_expansion** – {colour.characterisation.polynomial_expansion_Finlayson2015()}, Whether to use the root-polynomials set for the expansion.
- **terms** – {colour.characterisation.matrix_augmented_Cheung2004()}, Number of terms of the expanded polynomial.
- **kwargs** (Any) –

Returns Expanded a array.

Return type numpy.ndarray

References

[CWCR04], [FMH15], [WR04], [Wikipedia03f]

Examples

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

colour.MATRIX_COLOUR_CORRECTION_METHODS

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

Supported colour correction matrix methods.

References

[CWCR04], [FMH15], [WR04], [Wikipedia03f]

colour.matrix_colour_correction

colour.**matrix_colour_correction**(*M_T: ArrayLike, M_R: ArrayLike, method: Union[Literal['Cheung 2004', 'Finlayson 2015', 'Vandermonde'], str] = 'Cheung 2004', **kwargs: Any*) → NDArrayFloat

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

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

Parameters

- **M_T** (ArrayLike) – Test array M_T to fit onto array M_R.
- **M_R** (ArrayLike) – Reference array the array M_T will be colour fitted against.
- **method** (Union[Literal['Cheung 2004', 'Finlayson 2015', 'Vandermonde'], str]) – Computation method.
- **degree** – {colour.characterisation.polynomial_expansion_Finlayson2015(), colour.characterisation.polynomial_expansion_Vandermonde()}, Expanded polynomial degree, must be one of *[1, 2, 3, 4]* for colour.characterisation.polynomial_expansion_Finlayson2015() definition.
- **root_polynomial_expansion** – {colour.characterisation.polynomial_expansion_Finlayson2015()}, Whether to use the root-polynomials set for the expansion.
- **terms** – {colour.characterisation.matrix_augmented_Cheung2004()}, Number of terms of the expanded polynomial.
- **kwargs** (Any) –

Returns Colour correction matrix.

Return type numpy.ndarray

References

[CWCR04], [FMH15], [WR04], [Wikipedia03f]

Examples

```
>>> M_T = np.array(
...     [
...         [0.17224810, 0.09170660, 0.06416938],
...         [0.49189645, 0.27802050, 0.21923399],
...         [0.10999751, 0.18658946, 0.29938611],
...         [0.11666120, 0.14327905, 0.05713804],
...         [0.18988879, 0.18227649, 0.36056247],
...         [0.12501329, 0.42223442, 0.37027445],
...         [0.64785606, 0.22396782, 0.03365194],
...         [0.06761093, 0.11076896, 0.39779139],
...         [0.49101797, 0.09448929, 0.11623839],
...         [0.11622386, 0.04425753, 0.14469986],
...         [0.36867946, 0.44545230, 0.06028681],
...         [0.61632937, 0.32323906, 0.02437089],
...         [0.03016472, 0.06153243, 0.29014596],
...         [0.11103655, 0.30553067, 0.08149137],
...         [0.41162190, 0.05816656, 0.04845934],
...         [0.73339206, 0.53075188, 0.02475212],
...         [0.47347718, 0.08834792, 0.30310315],
...         [0.00000000, 0.25187016, 0.35062450],
...         [0.76809639, 0.78486240, 0.77808297],
...         [0.53822392, 0.54307997, 0.54710883],
...         [0.35458526, 0.35318419, 0.35524431],
...         [0.17976704, 0.18000531, 0.17991488],
...         [0.09351417, 0.09510603, 0.09675027],
...         [0.03405071, 0.03295077, 0.03702047],
...     ]
... )
>>> M_R = np.array(
...     [
...         [0.15579559, 0.09715755, 0.07514556],
...         [0.39113140, 0.25943419, 0.21266708],
...         [0.12824821, 0.18463570, 0.31508023],
...         [0.12028974, 0.13455659, 0.07408400],
...         [0.19368988, 0.21158946, 0.37955964],
...         [0.19957425, 0.36085439, 0.40678123],
...         [0.48896605, 0.20691688, 0.05816533],
...         [0.09775522, 0.16710693, 0.47147724],
...         [0.39358649, 0.12233400, 0.10526425],
...         [0.10780332, 0.07258529, 0.16151473],
...         [0.27502671, 0.34705454, 0.09728099],
...         [0.43980441, 0.26880559, 0.05430533],
...         [0.05887212, 0.11126272, 0.38552469],
...         [0.12705825, 0.25787860, 0.13566464],
...         [0.35612929, 0.07933258, 0.05118732],
...         [0.48131976, 0.42082843, 0.07120612],
...         [0.34665585, 0.15170714, 0.24969804],
...         [0.08261116, 0.24588716, 0.48707733],
...         [0.66054904, 0.65941137, 0.66376412],
...         [0.48051509, 0.47870296, 0.48230082],
```

(continues on next page)

(continued from previous page)

```
...         [0.33045354, 0.32904184, 0.33228886],
...         [0.18001305, 0.17978567, 0.18004416],
...         [0.10283975, 0.10424680, 0.10384975],
...         [0.04742204, 0.04772203, 0.04914226],
...     ]
... )
>>> matrix_colour_correction(M_T, M_R)
array([[ 0.6982266...,  0.0307162...,  0.1621042...],
       [ 0.0689349...,  0.6757961...,  0.1643038...],
       [-0.0631495...,  0.0921247...,  0.9713415...]])
```

colour.APPLY_MATRIX_COLOUR_CORRECTION_METHODS

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

Supported methods to apply a colour correction matrix.

References

[CWCR04], [FMH15], [WR04], [Wikipedia03f]

colour.apply_matrix_colour_correction

colour.**apply_matrix_colour_correction**(*RGB: ArrayLike, CCM: ArrayLike, method: Union[Literal['Cheung 2004', 'Finlayson 2015', 'Vandermonde'], str] = 'Cheung 2004', **kwargs: Any*) → NDArrayFloat

Apply given colour correction matrix CCM to given *RGB* colourspace array.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to apply the colour correction matrix CCM to.
- **CCM** (ArrayLike) – Colour correction matrix CCM.
- **method** (Union[Literal['Cheung 2004', 'Finlayson 2015', 'Vandermonde'], str]) – Computation method.
- **degree** – {colour.characterisation.polynomial_expansion_Finlayson2015(), colour.characterisation.polynomial_expansion_Vandermonde()}, Expanded polynomial degree, must be one of *[1, 2, 3, 4]* for colour.characterisation.polynomial_expansion_Finlayson2015() definition.
- **root_polynomial_expansion** – {colour.characterisation.polynomial_expansion_Finlayson2015()}, Whether to use the root-polynomials set for the expansion.
- **terms** – {colour.characterisation.matrix_augmented_Cheung2004()}, Number of terms of the expanded polynomial.
- **kwargs** (Any) –

Returns Colour corrected *RGB* colourspace array.

Return type numpy.ndarray

References

[CWCR04], [FMH15], [WR04], [Wikipedia03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> CCM = np.array(
...     [
...         [1.05263767, 0.13780789, -0.22763399],
...         [0.07395843, 1.02939945, -0.1060115],
...         [0.05725508, -0.20526336, 1.10151945],
...     ]
... )
>>> apply_matrix_colour_correction(RGB, CCM)
array([ 0.1793456...,  0.1003392...,  0.0617218...])
```

colour.COLOUR_CORRECTION_METHODS

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

Supported colour correction methods.

References

[CWCR04], [FMH15], [WR04], [Wikipedia03f]

colour.colour_correction

colour.**colour_correction**(*RGB: ArrayLike, M_T: ArrayLike, M_R: ArrayLike, method: Union[Literal['Cheung 2004', 'Finlayson 2015', 'Vandermonde'], str] = 'Cheung 2004', **kwargs: Any*) → NDArrayFloat

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

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to colour correct.
- **M_T** (ArrayLike) – Test array M_T to fit onto array M_R.
- **M_R** (ArrayLike) – Reference array the array M_T will be colour fitted against.
- **method** (Union[Literal['Cheung 2004', 'Finlayson 2015', 'Vandermonde'], str]) – Computation method.
- **degree** – {colour.characterisation.polynomial_expansion_Finlayson2015(), colour.characterisation.polynomial_expansion_Vandermonde()}, Expanded polynomial degree, must be one of *[1, 2, 3, 4]* for colour.characterisation.polynomial_expansion_Finlayson2015() definition.
- **root_polynomial_expansion** – {colour.characterisation.polynomial_expansion_Finlayson2015()}, Whether to use the root-polynomials set for the expansion.
- **terms** – {colour.characterisation.matrix_augmented_Cheung2004()}, Number of terms of the expanded polynomial.

- **kwargs** (Any) –

Returns Colour corrected *RGB* colourspace array.

Return type numpy.ndarray

References

[CWCR04], [FMH15], [WR04], [Wikipedia03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> M_T = np.array(
...     [
...         [0.17224810, 0.09170660, 0.06416938],
...         [0.49189645, 0.27802050, 0.21923399],
...         [0.10999751, 0.18658946, 0.29938611],
...         [0.11666120, 0.14327905, 0.05713804],
...         [0.18988879, 0.18227649, 0.36056247],
...         [0.12501329, 0.42223442, 0.37027445],
...         [0.64785606, 0.22396782, 0.03365194],
...         [0.06761093, 0.11076896, 0.39779139],
...         [0.49101797, 0.09448929, 0.11623839],
...         [0.11622386, 0.04425753, 0.14469986],
...         [0.36867946, 0.44545230, 0.06028681],
...         [0.61632937, 0.32323906, 0.02437089],
...         [0.03016472, 0.06153243, 0.29014596],
...         [0.11103655, 0.30553067, 0.08149137],
...         [0.41162190, 0.05816656, 0.04845934],
...         [0.73339206, 0.53075188, 0.02475212],
...         [0.47347718, 0.08834792, 0.30310315],
...         [0.00000000, 0.25187016, 0.35062450],
...         [0.76809639, 0.78486240, 0.77808297],
...         [0.53822392, 0.54307997, 0.54710883],
...         [0.35458526, 0.35318419, 0.35524431],
...         [0.17976704, 0.18000531, 0.17991488],
...         [0.09351417, 0.09510603, 0.09675027],
...         [0.03405071, 0.03295077, 0.03702047],
...     ]
... )
>>> M_R = np.array(
...     [
...         [0.15579559, 0.09715755, 0.07514556],
...         [0.39113140, 0.25943419, 0.21266708],
...         [0.12824821, 0.18463570, 0.31508023],
...         [0.12028974, 0.13455659, 0.07408400],
...         [0.19368988, 0.21158946, 0.37955964],
...         [0.19957425, 0.36085439, 0.40678123],
...         [0.48896605, 0.20691688, 0.05816533],
...         [0.09775522, 0.16710693, 0.47147724],
...         [0.39358649, 0.12233400, 0.10526425],
...         [0.10780332, 0.07258529, 0.16151473],
...         [0.27502671, 0.34705454, 0.09728099],
...         [0.43980441, 0.26880559, 0.05430533],
...         [0.05887212, 0.11126272, 0.38552469],
...         [0.12705825, 0.25787860, 0.13566464],
```

(continues on next page)

(continued from previous page)

```
...         [0.35612929, 0.07933258, 0.05118732],
...         [0.48131976, 0.42082843, 0.07120612],
...         [0.34665585, 0.15170714, 0.24969804],
...         [0.08261116, 0.24588716, 0.48707733],
...         [0.66054904, 0.65941137, 0.66376412],
...         [0.48051509, 0.47870296, 0.48230082],
...         [0.33045354, 0.32904184, 0.33228886],
...         [0.18001305, 0.17978567, 0.18004416],
...         [0.10283975, 0.10424680, 0.10384975],
...         [0.04742204, 0.04772203, 0.04914226],
...     ]
... )
>>> colour_correction(RGB, M_T, M_R)
array([ 0.1334872...,  0.0843921...,  0.0599014...])
```

Ancillary Objects

colour.characterisation

matrix_augmented_Cheung2004(RGB[, terms])	Perform polynomial expansion of given *RGB* colourspace array using *Cheung et al. (2004)* method.
polynomial_expansion_Finlayson2015(RGB[, ...])	Perform polynomial expansion of given *RGB* colourspace array using *Finlayson et al. (2015)* method.
polynomial_expansion_Vandermonde(a[, degree])	Perform polynomial expansion of given a array using *Vandermonde* method.
matrix_colour_correction_Cheung2004(M_T, M_R)	Compute a colour correction matrix from given M_T colour array to M_R colour array using *Cheung et al. (2004)* method.
matrix_colour_correction_Finlayson2015(M_T, M_R)	Compute a colour correction matrix from given M_T colour array to M_R colour array using *Finlayson et al. (2015)* method.
matrix_colour_correction_Vandermonde(M_T, M_R)	Compute a colour correction matrix from given M_T colour array to M_R colour array using *Vandermonde* method.
apply_matrix_colour_correction_Cheung2004(...)	Apply given colour correction matrix CCM computed using *Cheung et al. (2004)* method to given *RGB* colourspace array.
apply_matrix_colour_correction_Finlayson2015(...)	Apply given colour correction matrix CCM computed using *Finlayson et al. (2015)* method to given *RGB* colourspace array.
apply_matrix_colour_correction_Vandermonde(...)	Apply given colour correction matrix CCM computed using *Vandermonde* method to given *RGB* colourspace array.
colour_correction_Cheung2004(RGB, M_T, M_R)	Perform colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Cheung et al. (2004)* method.
colour_correction_Finlayson2015(RGB, M_T, M_R)	Perform colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Finlayson et al. (2015)* method.
colour_correction_Vandermonde(RGB, M_T, M_R)	Perform colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Vandermonde* method.

colour.characterisation.matrix_augmented_Cheung2004

colour.characterisation.**matrix_augmented_Cheung2004**(*RGB: ArrayLike, terms: Literal[3, 4, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22, 35] = 3*) → NDArrayFloat

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

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to expand.
- **terms** (Literal[3, 4, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22, 35]) – Number of terms of the expanded polynomial.

Returns Expanded *RGB* colourspace array.

Return type numpy.ndarray

Notes

- This definition combines the augmented matrices given in [CWCR04] and [WR04].

References

[CWCR04], [WR04]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> matrix_augmented_Cheung2004(RGB, terms=5)
array([ 0.1722481...,  0.0917066...,  0.0641693...,  0.0010136...,  1...])
```

colour.characterisation.polynomial_expansion_Finlayson2015

colour.characterisation.**polynomial_expansion_Finlayson2015**(*RGB: ArrayLike*, *degree: Literal[1, 2, 3, 4] = 1*, *root_polynomial_expansion: bool = True*) → NDArrayFloat

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

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to expand.
- **degree** (Literal[1, 2, 3, 4]) – Expanded polynomial degree.
- **root_polynomial_expansion** (bool) – Whether to use the root-polynomials set for the expansion.

Returns Expanded *RGB* colourspace array.

Return type numpy.ndarray

References

[FMH15]

Examples

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

colour.characterisation.polynomial_expansion_Vandermonde

colour.characterisation.**polynomial_expansion_Vandermonde**(*a: ArrayLike*, *degree: int = 1*) → NDArrayFloat

Perform polynomial expansion of given a array using *Vandermonde* method.

Parameters

- **a** (ArrayLike) – a array to expand.
- **degree** (int) – Expanded polynomial degree.

Returns Expanded a array.

Return type numpy.ndarray

References

[Wikipedia03f]

Examples

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

colour.characterisation.matrix_colour_correction_Cheung2004

colour.characterisation.**matrix_colour_correction_Cheung2004**(*M_T: ArrayLike*, *M_R: ArrayLike*, *terms: Literal[3, 4, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22, 35] = 3*) → NDArrayFloat

Compute a colour correction matrix from given M_T colour array to M_R colour array using *Cheung et al. (2004)* method.

Parameters

- **M_T** (ArrayLike) – Test array M_T to fit onto array M_R.
- **M_R** (ArrayLike) – Reference array the array M_T will be colour fitted against.
- **terms** (Literal[3, 4, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22, 35]) – Number of terms of the expanded polynomial.

Returns Colour correction matrix.

Return type numpy.ndarray

References

[CWCR04], [WR04]

Examples

```
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> matrix_colour_correction_Cheung2004(M_T, M_R)
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.matrix_colour_correction_Finlayson2015

colour.characterisation.**matrix_colour_correction_Finlayson2015**(*M_T: ArrayLike, M_R: ArrayLike, degree: Literal[1, 2, 3, 4] = 1, root_polynomial_expansion: bool = True*) → NDArrayFloat

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

Parameters

- **M_T** (ArrayLike) – Test array M_T to fit onto array M_R.
- **M_R** (ArrayLike) – Reference array the array M_T will be colour fitted against.
- **degree** (Literal[1, 2, 3, 4]) – Expanded polynomial degree.
- **root_polynomial_expansion** (bool) – Whether to use the root-polynomials set for the expansion.

Returns Colour correction matrix.

Return type numpy.ndarray

References

[FMH15]

Examples

```
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> matrix_colour_correction_Finlayson2015(M_T, M_R)
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.matrix_colour_correction_Vandermonde

colour.characterisation.**matrix_colour_correction_Vandermonde**(*M_T: ArrayLike*, *M_R: ArrayLike*, *degree: int = 1*) → NDArrayFloat

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

Parameters

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

Returns Colour correction matrix.

Return type numpy.ndarray

References

[Wikipedia03f]

Examples

```
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> matrix_colour_correction_Vandermonde(M_T, M_R)
array([[ 1.0300256...,  0.1141770..., -0.2621816...,  0.0418022...],
       [ 0.0670209...,  1.0221494..., -0.1166108...,  0.0128250...],
       [ 0.0744612..., -0.1872819...,  1.1278078..., -0.0318085...]])
```

colour.characterisation.apply_matrix_colour_correction_Cheung2004

colour.characterisation.**apply_matrix_colour_correction_Cheung2004**(*RGB: ArrayLike*, *CCM: ArrayLike*, *terms: Literal[3, 4, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22, 35] = 3*) → NDArrayFloat

Apply given colour correction matrix CCM computed using *Cheung et al. (2004)* method to given *RGB* colourspace array.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to apply the colour correction matrix CCM to.
- **CCM** (ArrayLike) – Colour correction matrix CCM.
- **terms** (Literal[3, 4, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22, 35]) – Number of terms of the expanded polynomial.

Returns Colour corrected *RGB* colourspace array.

Return type numpy.ndarray

References

[CWCR04], [WR04]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> CCM = np.array(
...     [
...         [1.05263767, 0.13780789, -0.22763399],
...         [0.07395843, 1.02939945, -0.1060115],
...         [0.05725508, -0.20526336, 1.10151945],
...     ]
... )
>>> apply_matrix_colour_correction_Cheung2004(
...     RGB, CCM
... )
array([ 0.1793456...,  0.1003392...,  0.0617218...])
```

colour.characterisation.apply_matrix_colour_correction_Finlayson2015

colour.characterisation.**apply_matrix_colour_correction_Finlayson2015**(*RGB: ArrayLike, CCM: ArrayLike, degree: Literal[1, 2, 3, 4] = 1, root_polynomial_expansion: bool = True*) → NDArrayFloat

Apply given colour correction matrix CCM computed using *Finlayson et al. (2015)* method to given *RGB* colourspace array.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to apply the colour correction matrix CCM to.
- **CCM** (ArrayLike) – Colour correction matrix CCM.
- **degree** (Literal[1, 2, 3, 4]) – Expanded polynomial degree.
- **root_polynomial_expansion** (bool) – Whether to use the root-polynomials set for the expansion.

Returns Colour corrected *RGB* colourspace array.

Return type numpy.ndarray

References

[FMH15]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> CCM = np.array(
...     [
...         [1.05263767, 0.13780789, -0.22763399],
...         [0.07395843, 1.02939945, -0.1060115],
...         [0.05725508, -0.20526336, 1.10151945],
...     ]
... )
>>> apply_matrix_colour_correction_Finlayson2015(
...     RGB, CCM
... )
array([ 0.1793456...,  0.1003392...,  0.0617218...])
```

colour.characterisation.apply_matrix_colour_correction_Vandermonde

colour.characterisation.**apply_matrix_colour_correction_Vandermonde**(*RGB: ArrayLike*, *CCM: ArrayLike*, *degree: int = 1*) → NDArrayFloat

Apply given colour correction matrix CCM computed using *Vandermonde* method to given *RGB* colourspace array.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to apply the colour correction matrix CCM to.
- **CCM** (ArrayLike) – Colour correction matrix CCM.
- **degree** (int) – Expanded polynomial degree.

Returns Colour corrected *RGB* colourspace array.

Return type numpy.ndarray

References

[Wikipedia03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> CCM = np.array(
...     [
...         [1.0300256, 0.11417701, -0.26218168, 0.04180222],
...         [0.06702098, 1.02214943, -0.11661082, 0.01282503],
...         [0.07446128, -0.18728192, 1.12780782, -0.03180856],
...     ]
... )
>>> apply_matrix_colour_correction_Vandermonde(
...     RGB, CCM
... )
array([ 0.2128689...,  0.1106242...,  0.0362129...])
```

colour.characterisation.colour_correction_Cheung2004

colour.characterisation.**colour_correction_Cheung2004**(*RGB: ArrayLike*, *M_T: ArrayLike*, *M_R: ArrayLike*, *terms: Literal[3, 4, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22, 35] = 3*) → NDArrayFloat

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

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to colour correct.
- **M_T** (ArrayLike) – Test array M_T to fit onto array M_R.
- **M_R** (ArrayLike) – Reference array the array M_T will be colour fitted against.
- **terms** (Literal[3, 4, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22, 35]) – Number of terms of the expanded polynomial.

Returns Colour corrected *RGB* colourspace array.

Return type numpy.ndarray

References

[CWCR04], [WR04]

Examples

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

colour.characterisation.colour_correction_Finlayson2015

colour.characterisation.**colour_correction_Finlayson2015**(*RGB: ArrayLike*, *M_T: ArrayLike*, *M_R: ArrayLike*, *degree: Literal[1, 2, 3, 4] = 1*, *root_polynomial_expansion: bool = True*) → NDArrayFloat

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

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to colour correct.
- **M_T** (ArrayLike) – Test array M_T to fit onto array M_R.
- **M_R** (ArrayLike) – Reference array the array M_T will be colour fitted against.
- **degree** (Literal[1, 2, 3, 4]) – Expanded polynomial degree.
- **root_polynomial_expansion** (bool) – Whether to use the root-polynomials set for the expansion.

Returns Colour corrected *RGB* colourspace array.

Return type numpy.ndarray

References

[FMH15]

Examples

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

colour.characterisation.colour_correction_Vandermonde

`colour.characterisation.colour_correction_Vandermonde`(*RGB: ArrayLike*, *M_T: ArrayLike*, *M_R: ArrayLike*, *degree: int = 1*) → NDArrayFloat

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

Parameters

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

Returns Colour corrected *RGB* colourspace array.

Return type `numpy.ndarray`

References

[Wikipedia03f]

Examples

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

Colour Rendition Charts

Dataset

colour

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

colour.CCS_COLOURCHECKERS

colour.**CCS_COLOURCHECKERS = CanonicalMapping({'ColorChecker 1976': ..., 'ColorChecker 2005': ..., 'BabelColor Average': ..., 'ColorChecker24 - Before November 2014': ..., 'ColorChecker24 - After November 2014': ..., 'TE226 V2': ..., 'babel_average': ..., 'cc2005': ..., 'ccb2014': ..., 'cca2014': ...})**

Chromaticity coordinates of the colour checkers.

References

[BabelColor12a], [BabelColor12b], [ImageEngineering17], [XRite16]

Aliases:

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

colour.SDS_COLOURCHECKERS

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

Spectral distributions of the colour checkers.

References

[Oht97], [BabelColor12a], [BabelColor12b], [MunsellCSciencea], [InternationalOfStandardization12]

Notes

- Data from [InternationalOfStandardization12] and [Oht97] has been verified to be the same.

Aliases:

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

Ancillary Objects

`colour.characterisation`

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

colour.characterisation.ColourChecker

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

Colour Checker data.

Parameters

- **name** – *Colour Checker* name.
- **data** – Chromaticity coordinates in *CIE xyY* colourspace.
- **illuminant** – *Colour Checker* illuminant chromaticity coordinates.
- **rows** – *Colour Checker* row count.
- **columns** – *Colour Checker* column count.

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

__init__()

Methods

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

Attributes

columns	Alias for field number 4
data	Alias for field number 1
illuminant	Alias for field number 2
name	Alias for field number 0
rows	Alias for field number 3

Cameras

`colour.characterisation`

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

colour.characterisation.RGB_CameraSensitivities

class colour.characterisation.**RGB_CameraSensitivities**(*data: ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*)

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implement support for a camera *RGB* sensitivities.

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) – Data to be stored in the multi-spectral distributions.
- **domain** (ArrayLike | SpectralShape | None) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths property.
- **labels** (Sequence | None) – Names to use for the colour.SpectralDistribution class instances.
- **extrapolator** – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **name** – Multi-spectral distributions name.
- **display_labels** – Multi-spectral distributions labels for figures, default to colour.colorimetry.RGB_CameraSensitivities.labels property value.
- **kwargs** (Any) –

Return type None

__init__(*data: ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*) → None

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) –
- **domain** (ArrayLike | SpectralShape | None) –
- **labels** (Sequence | None) –
- **kwargs** (Any) –

Return type None

Dataset

colour

MSDS_CAMERA_SENSITIVITIES	Multi-spectral distributions of camera sensitivities.

colour.MSDS_CAMERA_SENSITIVITIES

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

Multi-spectral distributions of camera sensitivities.

References

[DFGM15]

Displays

colour.characterisation

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

colour.characterisation.RGB_DisplayPrimaries

class colour.characterisation.**RGB_DisplayPrimaries**(*data: ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*)

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

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

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) – Data to be stored in the multi-spectral distributions.
- **domain** (ArrayLike | SpectralShape | None) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths property.
- **labels** (Sequence | None) – Names to use for the colour.SpectralDistribution class instances.

- **extrapolator** – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the `colour.SpectralDistribution` class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **name** – Multi-spectral distributions name.
- **display_labels** – Multi-spectral distributions labels for figures, default to `colour.colorimetry.RGB_DisplayPrimaries.labels` property value.
- **kwargs** (Any) –

Return type None

__init__(*data: ArrayLike* | *DataFrame* | *dict* | MultiSignals | MultiSpectralDistributions | Sequence | *Series* | Signal | SpectralDistribution | *None* = *None*, *domain: ArrayLike* | SpectralShape | *None* = *None*, *labels:* Sequence | *None* = *None*, ***kwargs:* Any) → None

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) –
- **domain** (ArrayLike | SpectralShape | None) –
- **labels** (Sequence | None) –
- **kwargs** (Any) –

Return type None

Dataset

colour

MSDS_DISPLAY_PRIMARIES	Primaries multi-spectral distributions of displays.

colour.MSDS_DISPLAY_PRIMARIES

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

Primaries multi-spectral distributions of displays.

References

[FW98], [Mac10]

Filters

Dataset

colour

SDS_FILTERS	Spectral distributions of filters.

colour.SDS_FILTERS

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

Spectral distributions of filters.

References

[InternationalOfStandardization02]

Lenses

Dataset

colour

SDS_LENSES	Spectral distributions of lenses.

colour.SDS_LENSES

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

Spectral distributions of lenses.

References

[InternationalOfStandardization02]

Colorimetry

Spectral Data Structure

colour

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

colour.SpectralShape

class colour.**SpectralShape**(*start: Real*, *end: Real*, *interval: Real*)

Bases: object

Define the base object for spectral distribution shape.

Parameters

- **start** (Real) – Wavelength λ_i range start in nm.
- **end** (Real) – Wavelength λ_i range end in nm.
- **interval** (Real) – Wavelength λ_i range interval.

Return type None

Attributes

- start
- end
- interval
- boundaries
- wavelengths

Methods

- __init__()
- __str__()
- __repr__()
- __hash__()
- __iter__()
- __contains__()
- __len__()
- __eq__()
- __ne__()
- range()

Examples

```
>>> SpectralShape(360, 830, 1)
SpectralShape(360, 830, 1)
```

__init__(*start: Real*, *end: Real*, *interval: Real*) → None

Parameters

- **start** (Real) –
- **end** (Real) –
- **interval** (Real) –

Return type None

property start: Real

Getter and setter property for the spectral shape start.

Parameters **value** – Value to set the spectral shape start with.

Returns Spectral shape start.

Return type Real

property end: Real

Getter and setter property for the spectral shape end.

Parameters **value** – Value to set the spectral shape end with.

Returns Spectral shape end.

Return type Real

property interval: Real

Getter and setter property for the spectral shape interval.

Parameters **value** – Value to set the spectral shape interval with.

Returns Spectral shape interval.

Return type Real

property boundaries: tuple

Getter and setter property for the spectral shape boundaries.

Parameters **value** – Value to set the spectral shape boundaries with.

Returns Spectral shape boundaries.

Return type `tuple`

property wavelengths: NDArrayFloat

Getter property for the spectral shape wavelengths.

Returns Spectral shape wavelengths.

Return type `numpy.ndarray`

__str__() → str

Return a formatted string representation of the spectral shape.

Returns Formatted string representation.

Return type `str`

__repr__() → str

Return an evaluable string representation of the spectral shape.

Returns Evaluable string representation.

Return type `str`

__hash__() → int

Return the spectral shape hash.

Returns Object hash.

Return type `int`

__iter__() → *collections.abc.Generator*

Return a generator for the spectral shape data.

Yields *Generator* – Spectral shape data generator.

Return type *collections.abc.Generator*

Examples

```
>>> shape = SpectralShape(0, 10, 1)
>>> for wavelength in shape:
...     print(wavelength)
...
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
```

`__contains__`(*wavelength: ArrayLike*) → bool

Return if the spectral shape contains given wavelength λ.

Parameters **`wavelength`** (`ArrayLike`) – Wavelength λ.

Returns Whether wavelength λ is contained in the spectral shape.

Return type `bool`

Examples

```
>>> 0.5 in SpectralShape(0, 10, 0.1)
True
>>> 0.6 in SpectralShape(0, 10, 0.1)
True
>>> 0.51 in SpectralShape(0, 10, 0.1)
False
>>> np.array([0.5, 0.6]) in SpectralShape(0, 10, 0.1)
True
>>> np.array([0.51, 0.6]) in SpectralShape(0, 10, 0.1)
False
```

`__len__`() → int

Return the spectral shape wavelength λ_n count.

Returns Spectral shape wavelength λ_n count.

Return type `int`

Examples

```
>>> len(SpectralShape(0, 10, 0.1))
101
```

`__eq__`(*other: Any*) → bool

Return whether the spectral shape is equal to given other object.

Parameters **`other`** (`Any`) – Object to test whether it is equal to the spectral shape.

Returns Whether given object is equal to the spectral shape.

Return type bool

Examples

```
>>> SpectralShape(0, 10, 0.1) == SpectralShape(0, 10, 0.1)
True
>>> SpectralShape(0, 10, 0.1) == SpectralShape(0, 10, 1)
False
```

__ne__(*other: Any*) → bool

Return whether the spectral shape is not equal to given other object.

Parameters **other** (Any) – Object to test whether it is not equal to the spectral shape.

Returns Whether given object is not equal to the spectral shape.

Return type bool

Examples

```
>>> SpectralShape(0, 10, 0.1) != SpectralShape(0, 10, 0.1)
False
>>> SpectralShape(0, 10, 0.1) != SpectralShape(0, 10, 1)
True
```

range(*dtype: Type[DTypeFloat] | None = None*) → NDArrayFloat

Return an iterable range for the spectral shape.

Parameters **dtype** (Type[DTypeFloat] | None) – Data type used to generate the range.

Returns Iterable range for the spectral distribution shape

Return type numpy.ndarray

Examples

```
>>> SpectralShape(0, 10, 0.1).wavelengths
array([ 0. ,   0.1,   0.2,   0.3,   0.4,   0.5,   0.6,   0.7,   0.8,
        0.9,   1. ,   1.1,   1.2,   1.3,   1.4,   1.5,   1.6,   1.7,
        1.8,   1.9,   2. ,   2.1,   2.2,   2.3,   2.4,   2.5,   2.6,
        2.7,   2.8,   2.9,   3. ,   3.1,   3.2,   3.3,   3.4,   3.5,
        3.6,   3.7,   3.8,   3.9,   4. ,   4.1,   4.2,   4.3,   4.4,
        4.5,   4.6,   4.7,   4.8,   4.9,   5. ,   5.1,   5.2,   5.3,
        5.4,   5.5,   5.6,   5.7,   5.8,   5.9,   6. ,   6.1,   6.2,
        6.3,   6.4,   6.5,   6.6,   6.7,   6.8,   6.9,   7. ,   7.1,
        7.2,   7.3,   7.4,   7.5,   7.6,   7.7,   7.8,   7.9,   8. ,
        8.1,   8.2,   8.3,   8.4,   8.5,   8.6,   8.7,   8.8,   8.9,
        9. ,   9.1,   9.2,   9.3,   9.4,   9.5,   9.6,   9.7,   9.8,
        9.9,  10. ])
```

__weakref__

list of weak references to the object (if defined)

colour.SpectralDistribution

class colour.**SpectralDistribution**(*data: ArrayLike | dict | Series | Signal | None = None, domain: ArrayLike | SpectralShape | None = None, **kwargs: Any*)

Bases: `colour.continuous.signal.Signal`

Define the spectral distribution: the base object for spectral computations.

The spectral distribution will be initialised according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable. Extrapolation is performed according to *CIE 167:2005* recommendation.

Important: Specific documentation about getting, setting, indexing and slicing the spectral power distribution values is available in the *Spectral Representation and Continuous Signal* section.

Parameters

- **data** (ArrayLike | dict | Series | Signal | None) – Data to be stored in the spectral distribution.
- **domain** (ArrayLike | SpectralShape | None) – Values to initialise the `colour.SpectralDistribution.wavelength` property with. If both `data` and `domain` arguments are defined, the latter will be used to initialise the `colour.SpectralDistribution.wavelength` property.
- **extrapolator** – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function.
- **interpolator** – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function.
- **name** – Spectral distribution name.
- **display_name** – Spectral distribution name for figures, default to `colour.SpectralDistribution.name` property value.
- **kwargs** (Any) –

Return type None

Warning: The *Cubic Spline* method might produce unexpected results with exceptionally noisy or non-uniformly spaced data.

Attributes

- `display_name`
- `wavelengths`
- `values`
- `shape`

Methods

- `__init__()`
- `interpolate()`
- `extrapolate()`
- `align()`
- `trim()`
- `normalise()`

References

[CIET13805a], [CIET13805d], [CIET14804c]

Examples

Instantiating a spectral distribution with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> with numpy_print_options(suppress=True):
...     SpectralDistribution(data)
...
SpectralDistribution([[ 500.    ,    0.0651],
                      [ 520.    ,    0.0705],
                      [ 540.    ,    0.0772],
                      [ 560.    ,    0.087 ],
                      [ 580.    ,    0.1128],
                      [ 600.    ,    0.136 ]],
                     SpragueInterpolator,
                     {},
                     Extrapolator,
                     {'method': 'Constant', 'left': None, 'right': None})
```

Instantiating a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[510] = 0.31416
>>> with numpy_print_options(suppress=True):
...     SpectralDistribution(data)
...
SpectralDistribution([[ 500.     ,    0.0651 ],
                      [ 510.     ,    0.31416],
                      [ 520.     ,    0.0705 ],
                      [ 540.     ,    0.0772 ],
                      [ 560.     ,    0.087  ],
                      [ 580.     ,    0.1128 ],
                      [ 600.     ,    0.136  ]],
                     CubicSplineInterpolator,
```

(continues on next page)

(continued from previous page)

```
                    {},
                    Extrapolator,
                    {'method': 'Constant', 'left': None, 'right': None})
```

Instantiation with a *Pandas* pandas.Series:

```
>>> from colour.utilities import is_pandas_installed
>>> if is_pandas_installed():
...     from pandas import Series
...
...     print(SpectralDistribution(Series(data)))
...
[[  5.0000000...e+02   6.5100000...e-02]
 [  5.2000000...e+02   7.0500000...e-02]
 [  5.4000000...e+02   7.7200000...e-02]
 [  5.6000000...e+02   8.7000000...e-02]
 [  5.8000000...e+02   1.1280000...e-01]
 [  6.0000000...e+02   1.3600000...e-01]
 [  5.1000000...e+02   3.1416000...e-01]]
```

__init__(*data: ArrayLike | dict | Series | Signal | None = None, domain: ArrayLike | SpectralShape | None = None, **kwargs: Any*) → None

Parameters

- **data** (ArrayLike | dict | Series | Signal | None) –
- **domain** (ArrayLike | SpectralShape | None) –
- **kwargs** (Any) –

Return type None

property display_name: str

Getter and setter property for the spectral distribution display name.

Parameters **value** – Value to set the spectral distribution display name with.

Returns Spectral distribution display name.

Return type str

property wavelengths: NDArrayFloat

Getter and setter property for the spectral distribution wavelengths λ_n.

Parameters **value** – Value to set the spectral distribution wavelengths λ_n with.

Returns Spectral distribution wavelengths λ_n.

Return type numpy.ndarray

property values: NDArrayFloat

Getter and setter property for the spectral distribution values.

Parameters **value** – Value to set the spectral distribution wavelengths values with.

Returns Spectral distribution values.

Return type numpy.ndarray

property shape: colour.colorimetry.spectrum.SpectralShape

Getter property for the spectral distribution shape.

Returns Spectral distribution shape.

Return type colour.SpectralShape

Notes

- A spectral distribution with a non-uniformly spaced independent variable have multiple intervals, in that case `colour.SpectralDistribution.shape` property returns the *minimum* interval size.

Examples

Shape of a spectral distribution with a uniformly spaced independent variable:

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> SpectralDistribution(data).shape
SpectralShape(500.0, 600.0, 20.0)
```

Shape of a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[510] = 0.31416
>>> SpectralDistribution(data).shape
SpectralShape(500.0, 600.0, 10.0)
```

interpolate(*shape: colour.colorimetry.spectrum.SpectralShape*, *interpolator: Optional[Type[colour.hints.ProtocolInterpolator]] = None*, *interpolator_kwargs: dict | None = None*) → Self

Interpolate the spectral distribution in-place according to *CIE 167:2005* recommendation (if the interpolator has not been changed at instantiation time) or given interpolation arguments.

The logic for choosing the interpolator class when `interpolator` is not given is as follows:

```
if self.interpolator not in (
    SpragueInterpolator,
    CubicSplineInterpolator,
):
    interpolator = self.interpolator
elif self.is_uniform():
    interpolator = SpragueInterpolator
else:
    interpolator = CubicSplineInterpolator
```

The logic for choosing the interpolator keyword arguments when `interpolator_kwargs` is not given is as follows:

```
if self.interpolator not in (
    SpragueInterpolator,
    CubicSplineInterpolator,
):
    interpolator_kwargs = self.interpolator_kwargs
else:
    interpolator_kwargs = {}
```

Parameters

- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used for interpolation.
- **interpolator** (Optional[Type[colour.hints.ProtocolInterpolator]]) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (dict | None) – Arguments to use when instantiating the interpolating function.

Returns Interpolated spectral distribution.

Return type colour.SpectralDistribution

Notes

- Interpolation will be performed over boundaries range, if you need to extend the range of the spectral distribution use the colour.SpectralDistribution.extrapolate() or colour.SpectralDistribution.align() methods.

Warning:

- *Cubic Spline* interpolator requires at least 3 wavelengths λ_n for interpolation.
- *Sprague (1880)* interpolator requires at least 6 wavelengths λ_n for interpolation.

References

[CIET13805a]

Examples

Spectral distribution with a uniformly spaced independent variable uses *Sprague (1880)* interpolation:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> sd = SpectralDistribution(data)
>>> with numpy_print_options(suppress=True):
...     print(sd.interpolate(SpectralShape(500, 600, 1)))
...
...
[[ 500.            0.0651   ...]
 [ 501.            0.0653522...]
 [ 502.            0.0656105...]
 [ 503.            0.0658715...]
 [ 504.            0.0661328...]
 [ 505.            0.0663929...]
 [ 506.            0.0666509...]
 [ 507.            0.0669069...]
```

(continues on next page)

(continued from previous page)

```
[ 508.          0.0671613...]
[ 509.          0.0674150...]
[ 510.          0.0676692...]
[ 511.          0.0679253...]
[ 512.          0.0681848...]
[ 513.          0.0684491...]
[ 514.          0.0687197...]
[ 515.          0.0689975...]
[ 516.          0.0692832...]
[ 517.          0.0695771...]
[ 518.          0.0698787...]
[ 519.          0.0701870...]
[ 520.          0.0705   ...]
[ 521.          0.0708155...]
[ 522.          0.0711336...]
[ 523.          0.0714547...]
[ 524.          0.0717789...]
[ 525.          0.0721063...]
[ 526.          0.0724367...]
[ 527.          0.0727698...]
[ 528.          0.0731051...]
[ 529.          0.0734423...]
[ 530.          0.0737808...]
[ 531.          0.0741203...]
[ 532.          0.0744603...]
[ 533.          0.0748006...]
[ 534.          0.0751409...]
[ 535.          0.0754813...]
[ 536.          0.0758220...]
[ 537.          0.0761633...]
[ 538.          0.0765060...]
[ 539.          0.0768511...]
[ 540.          0.0772   ...]
[ 541.          0.0775527...]
[ 542.          0.0779042...]
[ 543.          0.0782507...]
[ 544.          0.0785908...]
[ 545.          0.0789255...]
[ 546.          0.0792576...]
[ 547.          0.0795917...]
[ 548.          0.0799334...]
[ 549.          0.0802895...]
[ 550.          0.0806671...]
[ 551.          0.0810740...]
[ 552.          0.0815176...]
[ 553.          0.0820049...]
[ 554.          0.0825423...]
[ 555.          0.0831351...]
[ 556.          0.0837873...]
[ 557.          0.0845010...]
[ 558.          0.0852763...]
[ 559.          0.0861110...]
[ 560.          0.087    ...]
[ 561.          0.0879383...]
[ 562.          0.0889300...]
[ 563.          0.0899793...]
```

(continues on next page)

(continued from previous page)

```
 [ 564.              0.0910876...]
 [ 565.              0.0922541...]
 [ 566.              0.0934760...]
 [ 567.              0.0947487...]
 [ 568.              0.0960663...]
 [ 569.              0.0974220...]
 [ 570.              0.0988081...]
 [ 571.              0.1002166...]
 [ 572.              0.1016394...]
 [ 573.              0.1030687...]
 [ 574.              0.1044972...]
 [ 575.              0.1059186...]
 [ 576.              0.1073277...]
 [ 577.              0.1087210...]
 [ 578.              0.1100968...]
 [ 579.              0.1114554...]
 [ 580.              0.1128   ...]
 [ 581.              0.1141333...]
 [ 582.              0.1154495...]
 [ 583.              0.1167424...]
 [ 584.              0.1180082...]
 [ 585.              0.1192452...]
 [ 586.              0.1204536...]
 [ 587.              0.1216348...]
 [ 588.              0.1227915...]
 [ 589.              0.1239274...]
 [ 590.              0.1250465...]
 [ 591.              0.1261531...]
 [ 592.              0.1272517...]
 [ 593.              0.1283460...]
 [ 594.              0.1294393...]
 [ 595.              0.1305340...]
 [ 596.              0.1316310...]
 [ 597.              0.1327297...]
 [ 598.              0.1338277...]
 [ 599.              0.1349201...]
 [ 600.              0.136    ...]]
```

Spectral distribution with a non-uniformly spaced independent variable uses *Cubic Spline* interpolation:

```
>>> sd = SpectralDistribution(data)
>>> sd[510] = np.pi / 10
>>> with numpy_print_options(suppress=True):
...     print(sd.interpolate(SpectralShape(500, 600, 1)))
...
...
[[ 500.              0.0651   ...]
 [ 501.              0.1365202...]
 [ 502.              0.1953263...]
 [ 503.              0.2423724...]
 [ 504.              0.2785126...]
 [ 505.              0.3046010...]
 [ 506.              0.3214916...]
 [ 507.              0.3300387...]
 [ 508.              0.3310962...]
```

(continues on next page)

(continued from previous page)

```
[ 509.          0.3255184...]
[ 510.          0.3141592...]
[ 511.          0.2978729...]
[ 512.          0.2775135...]
[ 513.          0.2539351...]
[ 514.          0.2279918...]
[ 515.          0.2005378...]
[ 516.          0.1724271...]
[ 517.          0.1445139...]
[ 518.          0.1176522...]
[ 519.          0.0926962...]
[ 520.          0.0705   ...]
[ 521.          0.0517370...]
[ 522.          0.0363589...]
[ 523.          0.0241365...]
[ 524.          0.0148407...]
[ 525.          0.0082424...]
[ 526.          0.0041126...]
[ 527.          0.0022222...]
[ 528.          0.0023421...]
[ 529.          0.0042433...]
[ 530.          0.0076966...]
[ 531.          0.0124729...]
[ 532.          0.0183432...]
[ 533.          0.0250785...]
[ 534.          0.0324496...]
[ 535.          0.0402274...]
[ 536.          0.0481829...]
[ 537.          0.0560870...]
[ 538.          0.0637106...]
[ 539.          0.0708246...]
[ 540.          0.0772   ...]
[ 541.          0.0826564...]
[ 542.          0.0872086...]
[ 543.          0.0909203...]
[ 544.          0.0938549...]
[ 545.          0.0960760...]
[ 546.          0.0976472...]
[ 547.          0.0986321...]
[ 548.          0.0990942...]
[ 549.          0.0990971...]
[ 550.          0.0987043...]
[ 551.          0.0979794...]
[ 552.          0.0969861...]
[ 553.          0.0957877...]
[ 554.          0.0944480...]
[ 555.          0.0930304...]
[ 556.          0.0915986...]
[ 557.          0.0902161...]
[ 558.          0.0889464...]
[ 559.          0.0878532...]
[ 560.          0.087    ...]
[ 561.          0.0864371...]
[ 562.          0.0861623...]
[ 563.          0.0861600...]
[ 564.          0.0864148...]
```

(continues on next page)

(continued from previous page)

```
[ 565.          0.0869112...]
[ 566.          0.0876336...]
[ 567.          0.0885665...]
[ 568.          0.0896945...]
[ 569.          0.0910020...]
[ 570.          0.0924735...]
[ 571.          0.0940936...]
[ 572.          0.0958467...]
[ 573.          0.0977173...]
[ 574.          0.0996899...]
[ 575.          0.1017491...]
[ 576.          0.1038792...]
[ 577.          0.1060649...]
[ 578.          0.1082906...]
[ 579.          0.1105408...]
[ 580.          0.1128   ...]
[ 581.          0.1150526...]
[ 582.          0.1172833...]
[ 583.          0.1194765...]
[ 584.          0.1216167...]
[ 585.          0.1236884...]
[ 586.          0.1256760...]
[ 587.          0.1275641...]
[ 588.          0.1293373...]
[ 589.          0.1309798...]
[ 590.          0.1324764...]
[ 591.          0.1338114...]
[ 592.          0.1349694...]
[ 593.          0.1359349...]
[ 594.          0.1366923...]
[ 595.          0.1372262...]
[ 596.          0.1375211...]
[ 597.          0.1375614...]
[ 598.          0.1373316...]
[ 599.          0.1368163...]
[ 600.          0.136    ...]]
```

extrapolate(*shape: colour.colorimetry.spectrum.SpectralShape, extrapolator: Optional[Type[colour.hints.ProtocolExtrapolator]] = None, extrapolator_kwargs: dict | None = None*) → Self

Extrapolate the spectral distribution in-place according to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.

Parameters

- **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape used for extrapolation.
- **extrapolator** (`Optional[Type[colour.hints.ProtocolExtrapolator]]`) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (`dict | None`) – Arguments to use when instantiating the extrapolating function.

Returns Extrapolated spectral distribution.

Return type `colour.SpectralDistribution`

References

[CIET13805d], [CIET14804c]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> sd = SpectralDistribution(data)
>>> sd.extrapolate(SpectralShape(400, 700, 20)).shape
SpectralShape(400.0, 700.0, 20.0)
>>> with numpy_print_options(suppress=True):
...     print(sd)
...
[[ 400.        0.0651]
 [ 420.        0.0651]
 [ 440.        0.0651]
 [ 460.        0.0651]
 [ 480.        0.0651]
 [ 500.        0.0651]
 [ 520.        0.0705]
 [ 540.        0.0772]
 [ 560.        0.087 ]
 [ 580.        0.1128]
 [ 600.        0.136 ]
 [ 620.        0.136 ]
 [ 640.        0.136 ]
 [ 660.        0.136 ]
 [ 680.        0.136 ]
 [ 700.        0.136 ]]
```

align(*shape: colour.colorimetry.spectrum.SpectralShape, interpolator: Optional[Type[colour.hints.ProtocolInterpolator]] = None, interpolator_kwargs: dict | None = None, extrapolator: Optional[Type[colour.hints.ProtocolExtrapolator]] = None, extrapolator_kwargs: dict | None = None*) → Self

Align the spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.

Interpolation is performed according to *CIE 167:2005* recommendation (if the interpolator has not been changed at instantiation time) or given interpolation arguments.

The logic for choosing the interpolator class when `interpolator` is not given is as follows:

```
if self.interpolator not in (
    SpragueInterpolator,
    CubicSplineInterpolator,
):
    interpolator = self.interpolator
elif self.is_uniform():
    interpolator = SpragueInterpolator
```

(continues on next page)

(continued from previous page)

```
else:
    interpolator = CubicSplineInterpolator
```

The logic for choosing the interpolator keyword arguments when `interpolator_kwargs` is not given is as follows:

```
if self.interpolator not in (
    SpragueInterpolator,
    CubicSplineInterpolator,
):
    interpolator_kwargs = self.interpolator_kwargs
else:
    interpolator_kwargs = {}
```

Parameters

- **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape used for alignment.
- **interpolator** (`Optional[Type[colour.hints.ProtocolInterpolator]]`) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (`dict | None`) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (`Optional[Type[colour.hints.ProtocolExtrapolator]]`) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (`dict | None`) – Arguments to use when instantiating the extrapolating function.

Returns Aligned spectral distribution.

Return type `colour.SpectralDistribution`

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> sd = SpectralDistribution(data)
>>> with numpy_print_options(suppress=True):
...     print(sd.align(SpectralShape(505, 565, 1)))
...
...
[[ 505.            0.0663929...]
 [ 506.            0.0666509...]
 [ 507.            0.0669069...]
 [ 508.            0.0671613...]
 [ 509.            0.0674150...]
 [ 510.            0.0676692...]
 [ 511.            0.0679253...]
```

(continues on next page)

(continued from previous page)

```
[ 512.            0.0681848...]
[ 513.            0.0684491...]
[ 514.            0.0687197...]
[ 515.            0.0689975...]
[ 516.            0.0692832...]
[ 517.            0.0695771...]
[ 518.            0.0698787...]
[ 519.            0.0701870...]
[ 520.            0.0705   ...]
[ 521.            0.0708155...]
[ 522.            0.0711336...]
[ 523.            0.0714547...]
[ 524.            0.0717789...]
[ 525.            0.0721063...]
[ 526.            0.0724367...]
[ 527.            0.0727698...]
[ 528.            0.0731051...]
[ 529.            0.0734423...]
[ 530.            0.0737808...]
[ 531.            0.0741203...]
[ 532.            0.0744603...]
[ 533.            0.0748006...]
[ 534.            0.0751409...]
[ 535.            0.0754813...]
[ 536.            0.0758220...]
[ 537.            0.0761633...]
[ 538.            0.0765060...]
[ 539.            0.0768511...]
[ 540.            0.0772   ...]
[ 541.            0.0775527...]
[ 542.            0.0779042...]
[ 543.            0.0782507...]
[ 544.            0.0785908...]
[ 545.            0.0789255...]
[ 546.            0.0792576...]
[ 547.            0.0795917...]
[ 548.            0.0799334...]
[ 549.            0.0802895...]
[ 550.            0.0806671...]
[ 551.            0.0810740...]
[ 552.            0.0815176...]
[ 553.            0.0820049...]
[ 554.            0.0825423...]
[ 555.            0.0831351...]
[ 556.            0.0837873...]
[ 557.            0.0845010...]
[ 558.            0.0852763...]
[ 559.            0.0861110...]
[ 560.            0.087    ...]
[ 561.            0.0879383...]
[ 562.            0.0889300...]
[ 563.            0.0899793...]
[ 564.            0.0910876...]
[ 565.            0.0922541...]]
```

trim(*shape: colour.colorimetry.spectrum.SpectralShape*) → Self

Trim the spectral distribution wavelengths to given spectral shape.

Parameters **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used for trimming.

Returns Trimmed spectral distribution.

Return type colour.SpectralDistribution

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> sd = SpectralDistribution(data)
>>> sd = sd.interpolate(SpectralShape(500, 600, 1))
>>> with numpy_print_options(suppress=True):
...     print(sd.trim(SpectralShape(520, 580, 5)))
...
...
[[ 520.            0.0705    ...]
 [ 521.            0.0708155...]
 [ 522.            0.0711336...]
 [ 523.            0.0714547...]
 [ 524.            0.0717789...]
 [ 525.            0.0721063...]
 [ 526.            0.0724367...]
 [ 527.            0.0727698...]
 [ 528.            0.0731051...]
 [ 529.            0.0734423...]
 [ 530.            0.0737808...]
 [ 531.            0.0741203...]
 [ 532.            0.0744603...]
 [ 533.            0.0748006...]
 [ 534.            0.0751409...]
 [ 535.            0.0754813...]
 [ 536.            0.0758220...]
 [ 537.            0.0761633...]
 [ 538.            0.0765060...]
 [ 539.            0.0768511...]
 [ 540.            0.0772    ...]
 [ 541.            0.0775527...]
 [ 542.            0.0779042...]
 [ 543.            0.0782507...]
 [ 544.            0.0785908...]
 [ 545.            0.0789255...]
 [ 546.            0.0792576...]
 [ 547.            0.0795917...]
 [ 548.            0.0799334...]
 [ 549.            0.0802895...]
 [ 550.            0.0806671...]
 [ 551.            0.0810740...]
 [ 552.            0.0815176...]
```

(continues on next page)

(continued from previous page)

```
 [ 553.            0.0820049...]
 [ 554.            0.0825423...]
 [ 555.            0.0831351...]
 [ 556.            0.0837873...]
 [ 557.            0.0845010...]
 [ 558.            0.0852763...]
 [ 559.            0.0861110...]
 [ 560.            0.087    ...]
 [ 561.            0.0879383...]
 [ 562.            0.0889300...]
 [ 563.            0.0899793...]
 [ 564.            0.0910876...]
 [ 565.            0.0922541...]
 [ 566.            0.0934760...]
 [ 567.            0.0947487...]
 [ 568.            0.0960663...]
 [ 569.            0.0974220...]
 [ 570.            0.0988081...]
 [ 571.            0.1002166...]
 [ 572.            0.1016394...]
 [ 573.            0.1030687...]
 [ 574.            0.1044972...]
 [ 575.            0.1059186...]
 [ 576.            0.1073277...]
 [ 577.            0.1087210...]
 [ 578.            0.1100968...]
 [ 579.            0.1114554...]
 [ 580.            0.1128    ...]]
```

normalise(*factor: Real = 1*) → Self

Normalise the spectral distribution using given normalization factor.

Parameters **factor** (Real) – Normalization factor.

Returns Normalised spectral distribution.

Return type colour.SpectralDistribution

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> sd = SpectralDistribution(data)
>>> with numpy_print_options(suppress=True):
...     print(sd.normalise())
...
[[ 500.            0.4786764...]
 [ 520.            0.5183823...]
 [ 540.            0.5676470...]
```

(continues on next page)

(continued from previous page)

```
[ 560.         0.6397058...]
[ 580.         0.8294117...]
[ 600.         1.       ...]]
```

colour.MultiSpectralDistributions

class colour.**MultiSpectralDistributions**(*data: ArrayLike | DataFrame | dict | MultiSignals | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*)

Bases: `colour.continuous.multi_signals.MultiSignals`

Define the multi-spectral distributions: the base object for multi spectral computations. It is used to model colour matching functions, display primaries, camera sensitivities, etc...

The multi-spectral distributions will be initialised according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable. Extrapolation is performed according to *CIE 167:2005* recommendation.

Important: Specific documentation about getting, setting, indexing and slicing the multi-spectral power distributions values is available in the *Spectral Representation and Continuous Signal* section.

Parameters

- **data** (`ArrayLike | DataFrame | dict | MultiSignals | Sequence | Series | Signal | SpectralDistribution | None`) – Data to be stored in the multi-spectral distributions.
- **domain** (`ArrayLike | SpectralShape | None`) – Values to initialise the multiple `colour.SpectralDistribution` class instances `colour.continuous.Signal.wavelengths` attribute with. If both `data` and `domain` arguments are defined, the latter will be used to initialise the `colour.continuous.Signal.wavelengths` property.
- **labels** (`Sequence | None`) – Names to use for the `colour.SpectralDistribution` class instances.
- **extrapolator** – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the `colour.SpectralDistribution` class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **name** – Multi-spectral distributions name.
- **display_labels** – Multi-spectral distributions labels for figures, default to `colour.MultiSpectralDistributions.labels` property value.
- **kwargs** (`Any`) –

Return type None

> **Warning:** The *Cubic Spline* method might produce unexpected results with exceptionally noisy or non-uniformly spaced data.

Attributes

- display_name
- display_labels
- wavelengths
- values
- shape

Methods

- __init__()
- interpolate()
- extrapolate()
- align()
- trim()
- normalise()
- to_sds()

References

[CIET13805a], [CIET13805d], [CIET14804c]

Examples

Instantiating the multi-spectral distributions with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> labels = ("x_bar", "y_bar", "z_bar")
>>> with numpy_print_options(suppress=True):
...     MultiSpectralDistributions(data, labels=labels)
...
...
MultiSpectral...([[ 500.       ,    0.0049 ,    0.323  ,    0.272  ],
             ...  [ 510.       ,    0.0093 ,    0.503  ,    0.1582 ],
             ...  [ 520.       ,    0.06327,    0.71   ,    0.07825],
             ...  [ 530.       ,    0.1655 ,    0.862  ,    0.04216],
```

(continues on next page)

(continued from previous page)

```
             ...  [ 540.      ,    0.2904 ,    0.954  ,    0.0203 ],
             ...  [ 550.      ,    0.43345,    0.99495,    0.00875],
             ...  [ 560.      ,    0.5945 ,    0.995  ,    0.0039 ]],
             ... [...'x_bar', ...'y_bar', ...'z_bar'],
             ... SpragueInterpolator,
             ... {},
             ... Extrapolator,
             ... {'method': 'Constant', 'left': None, 'right': None})
```

Instantiating a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> with numpy_print_options(suppress=True):
...     MultiSpectralDistributions(data, labels=labels)
...
...
MultiSpectral...([[ 500.      ,    0.0049 ,    0.323  ,    0.272  ],
             ...  [ 510.      ,    0.0093 ,    0.503  ,    0.1582 ],
             ...  [ 511.      ,    0.00314,    0.31416,    0.03142],
             ...  [ 520.      ,    0.06327,    0.71   ,    0.07825],
             ...  [ 530.      ,    0.1655 ,    0.862  ,    0.04216],
             ...  [ 540.      ,    0.2904 ,    0.954  ,    0.0203 ],
             ...  [ 550.      ,    0.43345,    0.99495,    0.00875],
             ...  [ 560.      ,    0.5945 ,    0.995  ,    0.0039 ]],
             ... [...'x_bar', ...'y_bar', ...'z_bar'],
             ... CubicSplineInterpolator,
             ... {},
             ... Extrapolator,
             ... {'method': 'Constant', 'left': None, 'right': None})
```

Instantiation with a *Pandas DataFrame*:

```
>>> from colour.utilities import is_pandas_installed
>>> if is_pandas_installed():
...     from pandas import DataFrame
...
...     x_bar = [data[key][0] for key in sorted(data.keys())]
...     y_bar = [data[key][1] for key in sorted(data.keys())]
...     z_bar = [data[key][2] for key in sorted(data.keys())]
...     print(
...         MultiSignals(
...             DataFrame(
...                 dict(zip(labels, [x_bar, y_bar, z_bar])), data.keys()
...             )
...         )
...     )
...
[[  5.0000000...e+02   4.9000000...e-03   3.2300000...e-01   2.7200000...e-01]
 [  5.1000000...e+02   9.3000000...e-03   5.0300000...e-01   1.5820000...e-01]
 [  5.2000000...e+02   3.1400000...e-03   3.1416000...e-01   3.1420000...e-02]
 [  5.3000000...e+02   6.3270000...e-02   7.1000000...e-01   7.8250000...e-02]
 [  5.4000000...e+02   1.6550000...e-01   8.6200000...e-01   4.2160000...e-02]
 [  5.5000000...e+02   2.9040000...e-01   9.5400000...e-01   2.0300000...e-02]
 [  5.6000000...e+02   4.3345000...e-01   9.9495000...e-01   8.7500000...e-03]
 [  5.1100000...e+02   5.9450000...e-01   9.9500000...e-01   3.9000000...e-03]]
```

__init__(*data: ArrayLike | DataFrame | dict | MultiSignals | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*) → None

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | Sequence | Series | Signal | SpectralDistribution | None) –
- **domain** (ArrayLike | SpectralShape | None) –
- **labels** (Sequence | None) –
- **kwargs** (Any) –

Return type None

property display_name: str

Getter and setter property for the multi-spectral distributions display name.

Parameters value – Value to set the multi-spectral distributions display name with.

Returns Multi-spectral distributions display name.

Return type str

property display_labels: List[str]

Getter and setter property for the multi-spectral distributions display labels.

Parameters value – Value to set the multi-spectral distributions display labels with.

Returns Multi-spectral distributions display labels.

Return type list

property wavelengths: NDArrayFloat

Getter and setter property for the multi-spectral distributions wavelengths λ_n.

Parameters value – Value to set the multi-spectral distributions wavelengths λ_n with.

Returns Multi-spectral distributions wavelengths λ_n.

Return type numpy.ndarray

property values: NDArrayFloat

Getter and setter property for the multi-spectral distributions values.

Parameters value – Value to set the multi-spectral distributions wavelengths values with.

Returns Multi-spectral distributions values.

Return type numpy.ndarray

property shape: colour.colorimetry.spectrum.SpectralShape

Getter property for the multi-spectral distributions shape.

Returns Multi-spectral distributions shape.

Return type colour.SpectralShape

Notes

- Multi-spectral distributions with a non-uniformly spaced independent variable have multiple intervals, in that case `colour.MultiSpectralDistributions.shape` property returns the *minimum* interval size.

Examples

Shape of the multi-spectral distributions with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> MultiSpectralDistributions(data).shape
SpectralShape(500.0, 560.0, 10.0)
```

Shape of the multi-spectral distributions with a non-uniformly spaced independent variable:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> MultiSpectralDistributions(data).shape
SpectralShape(500.0, 560.0, 1.0)
```

interpolate(*shape: colour.colorimetry.spectrum.SpectralShape*, *interpolator: Optional[Type[colour.hints.ProtocolInterpolator]] = None*, *interpolator_kwargs: dict | None = None*) → Self

Interpolate the multi-spectral distributions in-place according to *CIE 167:2005* recommendation (if the interpolator has not been changed at instantiation time) or given interpolation arguments.

The logic for choosing the interpolator class when `interpolator` is not given is as follows:

```
if self.interpolator not in (
    SpragueInterpolator,
    CubicSplineInterpolator,
):
    interpolator = self.interpolator
elif self.is_uniform():
    interpolator = SpragueInterpolator
else:
    interpolator = CubicSplineInterpolator
```

The logic for choosing the interpolator keyword arguments when `interpolator_kwargs` is not given is as follows:

```
if self.interpolator not in (
    SpragueInterpolator,
    CubicSplineInterpolator,
):
    interpolator_kwargs = self.interpolator_kwargs
else:
    interpolator_kwargs = {}
```

Parameters

- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used for interpolation.
- **interpolator** (Optional[Type[colour.hints.ProtocolInterpolator]]) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (dict | None) – Arguments to use when instantiating the interpolating function.

Returns Interpolated multi-spectral distributions.

Return type colour.MultiSpectralDistributions

Notes

- See colour.SpectralDistribution.interpolate() method notes section.

Warning: See colour.SpectralDistribution.interpolate() method warning section.

References

[CIET13805a]

Examples

Multi-spectral distributions with a uniformly spaced independent variable uses *Sprague (1880)* interpolation:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.interpolate(SpectralShape(500, 560, 1)))
...
...
[[ 500.            0.0049    ...     0.323    ...     0.272    ...]
 [ 501.            0.0043252...     0.3400642...     0.2599848...]
 [ 502.            0.0037950...     0.3572165...     0.2479849...]
 [ 503.            0.0033761...     0.3744030...     0.2360688...]
 [ 504.            0.0031397...     0.3916650...     0.2242878...]
 [ 505.            0.0031582...     0.4091067...     0.2126801...]
 [ 506.            0.0035019...     0.4268629...     0.2012748...]
 [ 507.            0.0042365...     0.4450668...     0.1900968...]
 [ 508.            0.0054192...     0.4638181...     0.1791709...]
 [ 509.            0.0070965...     0.4831505...     0.1685260...]
 [ 510.            0.0093    ...     0.503    ...     0.1582   ...]
```

(continues on next page)

(continued from previous page)

```
 [ 511.           0.0120562...   0.5232543...   0.1482365...]
 [ 512.           0.0154137...   0.5439717...   0.1386625...]
 [ 513.           0.0193991...   0.565139 ...   0.1294993...]
 [ 514.           0.0240112...   0.5866255...   0.1207676...]
 [ 515.           0.0292289...   0.6082226...   0.1124864...]
 [ 516.           0.0350192...   0.6296821...   0.1046717...]
 [ 517.           0.0413448...   0.6507558...   0.0973361...]
 [ 518.           0.0481727...   0.6712346...   0.0904871...]
 [ 519.           0.0554816...   0.6909873...   0.0841267...]
 [ 520.           0.06327  ...   0.71     ...   0.07825  ...]
 [ 521.           0.0715642...   0.7283456...   0.0728614...]
 [ 522.           0.0803970...   0.7459679...   0.0680051...]
 [ 523.           0.0897629...   0.7628184...   0.0636823...]
 [ 524.           0.0996227...   0.7789004...   0.0598449...]
 [ 525.           0.1099142...   0.7942533...   0.0564111...]
 [ 526.           0.1205637...   0.8089368...   0.0532822...]
 [ 527.           0.1314973...   0.8230153...   0.0503588...]
 [ 528.           0.1426523...   0.8365417...   0.0475571...]
 [ 529.           0.1539887...   0.8495422...   0.0448253...]
 [ 530.           0.1655   ...   0.862    ...   0.04216  ...]
 [ 531.           0.1772055...   0.8738585...   0.0395936...]
 [ 532.           0.1890877...   0.8850940...   0.0371046...]
 [ 533.           0.2011304...   0.8957073...   0.0346733...]
 [ 534.           0.2133310...   0.9057092...   0.0323006...]
 [ 535.           0.2256968...   0.9151181...   0.0300011...]
 [ 536.           0.2382403...   0.9239560...   0.0277974...]
 [ 537.           0.2509754...   0.9322459...   0.0257131...]
 [ 538.           0.2639130...   0.9400080...   0.0237668...]
 [ 539.           0.2770569...   0.9472574...   0.0219659...]
 [ 540.           0.2904   ...   0.954    ...   0.0203   ...]
 [ 541.           0.3039194...   0.9602409...   0.0187414...]
 [ 542.           0.3175893...   0.9660106...   0.0172748...]
 [ 543.           0.3314022...   0.9713260...   0.0158947...]
 [ 544.           0.3453666...   0.9761850...   0.0146001...]
 [ 545.           0.3595019...   0.9805731...   0.0133933...]
 [ 546.           0.3738324...   0.9844703...   0.0122777...]
 [ 547.           0.3883818...   0.9878583...   0.0112562...]
 [ 548.           0.4031674...   0.9907270...   0.0103302...]
 [ 549.           0.4181943...   0.9930817...   0.0094972...]
 [ 550.           0.43345  ...   0.99495  ...   0.00875  ...]
 [ 551.           0.4489082...   0.9963738...   0.0080748...]
 [ 552.           0.4645599...   0.9973682...   0.0074580...]
 [ 553.           0.4803950...   0.9979568...   0.0068902...]
 [ 554.           0.4963962...   0.9981802...   0.0063660...]
 [ 555.           0.5125410...   0.9980910...   0.0058818...]
 [ 556.           0.5288034...   0.9977488...   0.0054349...]
 [ 557.           0.5451560...   0.9972150...   0.0050216...]
 [ 558.           0.5615719...   0.9965479...   0.0046357...]
 [ 559.           0.5780267...   0.9957974...   0.0042671...]
 [ 560.           0.5945   ...   0.995    ...   0.0039   ...]]
```

Multi-spectral distributions with a non-uniformly spaced independent variable uses *Cubic Spline* interpolation:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> msds = MultiSpectralDistributions(data)
```

(continues on next page)

(continued from previous page)

```
>>> with numpy_print_options(suppress=True):
...     print(msds.interpolate(SpectralShape(500, 560, 1)))
...
...
[[ 500.           0.0049    ...   0.323     ...   0.272    ...]
 [ 501.           0.0300110...   0.9455153...   0.5985102...]
 [ 502.           0.0462136...   1.3563103...   0.8066498...]
 [ 503.           0.0547925...   1.5844039...   0.9126502...]
 [ 504.           0.0570325...   1.6588148...   0.9327429...]
 [ 505.           0.0542183...   1.6085619...   0.8831594...]
 [ 506.           0.0476346...   1.4626640...   0.7801312...]
 [ 507.           0.0385662...   1.2501401...   0.6398896...]
 [ 508.           0.0282978...   1.0000089...   0.4786663...]
 [ 509.           0.0181142...   0.7412892...   0.3126925...]
 [ 510.           0.0093    ...   0.503     ...   0.1582    ...]
 [ 511.           0.00314   ...   0.31416   ...   0.03142   ...]
 [ 512.           0.0006228...   0.1970419...  -0.0551709...]
 [ 513.           0.0015528...   0.1469341...  -0.1041165...]
 [ 514.           0.0054381...   0.1523785...  -0.1217152...]
 [ 515.           0.0117869...   0.2019173...  -0.1142659...]
 [ 516.           0.0201073...   0.2840925...  -0.0880670...]
 [ 517.           0.0299077...   0.3874463...  -0.0494174...]
 [ 518.           0.0406961...   0.5005208...  -0.0046156...]
 [ 519.           0.0519808...   0.6118579...   0.0400397...]
 [ 520.           0.06327   ...   0.71      ...   0.07825   ...]
 [ 521.           0.0741690...   0.7859059...   0.1050384...]
 [ 522.           0.0846726...   0.8402033...   0.1207164...]
 [ 523.           0.0948728...   0.8759363...   0.1269173...]
 [ 524.           0.1048614...   0.8961496...   0.1252743...]
 [ 525.           0.1147305...   0.9038874...   0.1174207...]
 [ 526.           0.1245719...   0.9021942...   0.1049899...]
 [ 527.           0.1344776...   0.8941145...   0.0896151...]
 [ 528.           0.1445395...   0.8826926...   0.0729296...]
 [ 529.           0.1548497...   0.8709729...   0.0565668...]
 [ 530.           0.1655    ...   0.862     ...   0.04216   ...]
 [ 531.           0.1765618...   0.858179  ...   0.0309976...]
 [ 532.           0.1880244...   0.8593588...   0.0229897...]
 [ 533.           0.1998566...   0.8647493...   0.0177013...]
 [ 534.           0.2120269...   0.8735601...   0.0146975...]
 [ 535.           0.2245042...   0.8850011...   0.0135435...]
 [ 536.           0.2372572...   0.8982820...   0.0138044...]
 [ 537.           0.2502546...   0.9126126...   0.0150454...]
 [ 538.           0.2634650...   0.9272026...   0.0168315...]
 [ 539.           0.2768572...   0.9412618...   0.0187280...]
 [ 540.           0.2904    ...   0.954     ...   0.0203    ...]
 [ 541.           0.3040682...   0.9647869...   0.0211987...]
 [ 542.           0.3178617...   0.9736329...   0.0214207...]
 [ 543.           0.3317865...   0.9807080...   0.0210486...]
 [ 544.           0.3458489...   0.9861825...   0.0201650...]
 [ 545.           0.3600548...   0.9902267...   0.0188525...]
 [ 546.           0.3744103...   0.9930107...   0.0171939...]
 [ 547.           0.3889215...   0.9947048...   0.0152716...]
 [ 548.           0.4035944...   0.9954792...   0.0131685...]
 [ 549.           0.4184352...   0.9955042...   0.0109670...]
 [ 550.           0.43345   ...   0.99495   ...   0.00875   ...]
 [ 551.           0.4486447...   0.9939867...   0.0065999...]
```

(continues on next page)

(continued from previous page)

```
[ 552.           0.4640255...    0.9927847...    0.0045994...]
[ 553.           0.4795984...    0.9915141...    0.0028313...]
[ 554.           0.4953696...    0.9903452...    0.0013781...]
[ 555.           0.5113451...    0.9894483...    0.0003224...]
[ 556.           0.5275310...    0.9889934...   -0.0002530...]
[ 557.           0.5439334...    0.9891509...   -0.0002656...]
[ 558.           0.5605583...    0.9900910...    0.0003672...]
[ 559.           0.5774118...    0.9919840...    0.0017282...]
[ 560.           0.5945    ...   0.995     ...   0.0039    ...]]
```

extrapolate(*shape: colour.colorimetry.spectrum.SpectralShape*, *extrapolator: Optional[Type[colour.hints.ProtocolExtrapolator]] = None*, *extrapolator_kwargs: dict | None = None*) → Self

Extrapolate the multi-spectral distributions in-place according to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.

Parameters

- **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape used for extrapolation.
- **extrapolator** (`Optional[Type[colour.hints.ProtocolExtrapolator]]`) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (`dict | None`) – Arguments to use when instantiating the extrapolating function.

Returns Extrapolated multi-spectral distributions.

Return type `colour.MultiSpectralDistributions`

References

[CIET13805d], [CIET14804c]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> msds = MultiSpectralDistributions(data)
>>> msds.extrapolate(SpectralShape(400, 700, 10)).shape
SpectralShape(400.0, 700.0, 10.0)
>>> with numpy_print_options(suppress=True):
...     print(msds)
...
[[ 400.         0.0049     0.323      0.272  ]
 [ 410.         0.0049     0.323      0.272  ]
 [ 420.         0.0049     0.323      0.272  ]
 [ 430.         0.0049     0.323      0.272  ]
```

(continues on next page)

(continued from previous page)

```
[ 440.        0.0049     0.323      0.272   ]
[ 450.        0.0049     0.323      0.272   ]
[ 460.        0.0049     0.323      0.272   ]
[ 470.        0.0049     0.323      0.272   ]
[ 480.        0.0049     0.323      0.272   ]
[ 490.        0.0049     0.323      0.272   ]
[ 500.        0.0049     0.323      0.272   ]
[ 510.        0.0093     0.503      0.1582 ]
[ 520.        0.06327    0.71       0.07825]
[ 530.        0.1655     0.862      0.04216]
[ 540.        0.2904     0.954      0.0203 ]
[ 550.        0.43345    0.99495    0.00875]
[ 560.        0.5945     0.995      0.0039 ]
[ 570.        0.5945     0.995      0.0039 ]
[ 580.        0.5945     0.995      0.0039 ]
[ 590.        0.5945     0.995      0.0039 ]
[ 600.        0.5945     0.995      0.0039 ]
[ 610.        0.5945     0.995      0.0039 ]
[ 620.        0.5945     0.995      0.0039 ]
[ 630.        0.5945     0.995      0.0039 ]
[ 640.        0.5945     0.995      0.0039 ]
[ 650.        0.5945     0.995      0.0039 ]
[ 660.        0.5945     0.995      0.0039 ]
[ 670.        0.5945     0.995      0.0039 ]
[ 680.        0.5945     0.995      0.0039 ]
[ 690.        0.5945     0.995      0.0039 ]
[ 700.        0.5945     0.995      0.0039 ]]
```

align(*shape: colour.colorimetry.spectrum.SpectralShape*, *interpolator: Optional[Type[colour.hints.ProtocolInterpolator]] = None*, *interpolator_kwargs: dict | None = None*, *extrapolator: Optional[Type[colour.hints.ProtocolExtrapolator]] = None*, *extrapolator_kwargs: dict | None = None*) → Self

Align the multi-spectral distributions in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.

Interpolation is performed according to *CIE 167:2005* recommendation (if the interpolator has not been changed at instantiation time) or given interpolation arguments.

The logic for choosing the interpolator class when `interpolator` is not given is as follows:

```
if self.interpolator not in (
    SpragueInterpolator,
    CubicSplineInterpolator,
):
    interpolator = self.interpolator
elif self.is_uniform():
    interpolator = SpragueInterpolator
else:
    interpolator = CubicSplineInterpolator
```

The logic for choosing the interpolator keyword arguments when `interpolator_kwargs` is not given is as follows:

```
if self.interpolator not in (
    SpragueInterpolator,
    CubicSplineInterpolator,
):
```

(continues on next page)

(continued from previous page)

```
    interpolator_kwargs = self.interpolator_kwargs
else:
    interpolator_kwargs = {}
```

Parameters

- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used for alignment.
- **interpolator** (Optional[Type[colour.hints.ProtocolInterpolator]]) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (dict | None) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (Optional[Type[colour.hints.ProtocolExtrapolator]]) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (dict | None) – Arguments to use when instantiating the extrapolating function.

Returns Aligned multi-spectral distributions.

Return type colour.MultiSpectralDistributions

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.align(SpectralShape(505, 565, 1)))
...
...
[[ 505.          0.0031582...   0.4091067...   0.2126801...]
 [ 506.          0.0035019...   0.4268629...   0.2012748...]
 [ 507.          0.0042365...   0.4450668...   0.1900968...]
 [ 508.          0.0054192...   0.4638181...   0.1791709...]
 [ 509.          0.0070965...   0.4831505...   0.1685260...]
 [ 510.          0.0093    ...   0.503    ...   0.1582    ...]
 [ 511.          0.0120562...   0.5232543...   0.1482365...]
 [ 512.          0.0154137...   0.5439717...   0.1386625...]
 [ 513.          0.0193991...   0.565139 ...   0.1294993...]
 [ 514.          0.0240112...   0.5866255...   0.1207676...]
 [ 515.          0.0292289...   0.6082226...   0.1124864...]
 [ 516.          0.0350192...   0.6296821...   0.1046717...]
 [ 517.          0.0413448...   0.6507558...   0.0973361...]
 [ 518.          0.0481727...   0.6712346...   0.0904871...]
 [ 519.          0.0554816...   0.6909873...   0.0841267...]
 [ 520.          0.06327   ...   0.71      ...   0.07825   ...]
```

(continues on next page)

(continued from previous page)

```
[ 521.          0.0715642...   0.7283456...   0.0728614...]
[ 522.          0.0803970...   0.7459679...   0.0680051...]
[ 523.          0.0897629...   0.7628184...   0.0636823...]
[ 524.          0.0996227...   0.7789004...   0.0598449...]
[ 525.          0.1099142...   0.7942533...   0.0564111...]
[ 526.          0.1205637...   0.8089368...   0.0532822...]
[ 527.          0.1314973...   0.8230153...   0.0503588...]
[ 528.          0.1426523...   0.8365417...   0.0475571...]
[ 529.          0.1539887...   0.8495422...   0.0448253...]
[ 530.          0.1655    ...   0.862     ...   0.04216   ...]
[ 531.          0.1772055...   0.8738585...   0.0395936...]
[ 532.          0.1890877...   0.8850940...   0.0371046...]
[ 533.          0.2011304...   0.8957073...   0.0346733...]
[ 534.          0.2133310...   0.9057092...   0.0323006...]
[ 535.          0.2256968...   0.9151181...   0.0300011...]
[ 536.          0.2382403...   0.9239560...   0.0277974...]
[ 537.          0.2509754...   0.9322459...   0.0257131...]
[ 538.          0.2639130...   0.9400080...   0.0237668...]
[ 539.          0.2770569...   0.9472574...   0.0219659...]
[ 540.          0.2904    ...   0.954     ...   0.0203    ...]
[ 541.          0.3039194...   0.9602409...   0.0187414...]
[ 542.          0.3175893...   0.9660106...   0.0172748...]
[ 543.          0.3314022...   0.9713260...   0.0158947...]
[ 544.          0.3453666...   0.9761850...   0.0146001...]
[ 545.          0.3595019...   0.9805731...   0.0133933...]
[ 546.          0.3738324...   0.9844703...   0.0122777...]
[ 547.          0.3883818...   0.9878583...   0.0112562...]
[ 548.          0.4031674...   0.9907270...   0.0103302...]
[ 549.          0.4181943...   0.9930817...   0.0094972...]
[ 550.          0.43345   ...   0.99495   ...   0.00875   ...]
[ 551.          0.4489082...   0.9963738...   0.0080748...]
[ 552.          0.4645599...   0.9973682...   0.0074580...]
[ 553.          0.4803950...   0.9979568...   0.0068902...]
[ 554.          0.4963962...   0.9981802...   0.0063660...]
[ 555.          0.5125410...   0.9980910...   0.0058818...]
[ 556.          0.5288034...   0.9977488...   0.0054349...]
[ 557.          0.5451560...   0.9972150...   0.0050216...]
[ 558.          0.5615719...   0.9965479...   0.0046357...]
[ 559.          0.5780267...   0.9957974...   0.0042671...]
[ 560.          0.5945    ...   0.995     ...   0.0039    ...]
[ 561.          0.5945    ...   0.995     ...   0.0039    ...]
[ 562.          0.5945    ...   0.995     ...   0.0039    ...]
[ 563.          0.5945    ...   0.995     ...   0.0039    ...]
[ 564.          0.5945    ...   0.995     ...   0.0039    ...]
[ 565.          0.5945    ...   0.995     ...   0.0039    ...]]
```

trim(*shape: colour.colorimetry.spectrum.SpectralShape*) → Self

Trim the multi-spectral distributions wavelengths to given shape.

Parameters **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape used for trimming.

Returns Trimmed multi-spectral distributions.

Return type `colour.MultiSpectralDistributions`

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> msds = MultiSpectralDistributions(data)
>>> msds = msds.interpolate(SpectralShape(500, 560, 1))
>>> with numpy_print_options(suppress=True):
...     print(msds.trim(SpectralShape(520, 580, 5)))
...
...
[[ 520.          0.06327  ...    0.71      ...    0.07825  ...]
 [ 521.          0.0715642...    0.7283456...    0.0728614...]
 [ 522.          0.0803970...    0.7459679...    0.0680051...]
 [ 523.          0.0897629...    0.7628184...    0.0636823...]
 [ 524.          0.0996227...    0.7789004...    0.0598449...]
 [ 525.          0.1099142...    0.7942533...    0.0564111...]
 [ 526.          0.1205637...    0.8089368...    0.0532822...]
 [ 527.          0.1314973...    0.8230153...    0.0503588...]
 [ 528.          0.1426523...    0.8365417...    0.0475571...]
 [ 529.          0.1539887...    0.8495422...    0.0448253...]
 [ 530.          0.1655   ...    0.862    ...    0.04216  ...]
 [ 531.          0.1772055...    0.8738585...    0.0395936...]
 [ 532.          0.1890877...    0.8850940...    0.0371046...]
 [ 533.          0.2011304...    0.8957073...    0.0346733...]
 [ 534.          0.2133310...    0.9057092...    0.0323006...]
 [ 535.          0.2256968...    0.9151181...    0.0300011...]
 [ 536.          0.2382403...    0.9239560...    0.0277974...]
 [ 537.          0.2509754...    0.9322459...    0.0257131...]
 [ 538.          0.2639130...    0.9400080...    0.0237668...]
 [ 539.          0.2770569...    0.9472574...    0.0219659...]
 [ 540.          0.2904   ...    0.954    ...    0.0203   ...]
 [ 541.          0.3039194...    0.9602409...    0.0187414...]
 [ 542.          0.3175893...    0.9660106...    0.0172748...]
 [ 543.          0.3314022...    0.9713260...    0.0158947...]
 [ 544.          0.3453666...    0.9761850...    0.0146001...]
 [ 545.          0.3595019...    0.9805731...    0.0133933...]
 [ 546.          0.3738324...    0.9844703...    0.0122777...]
 [ 547.          0.3883818...    0.9878583...    0.0112562...]
 [ 548.          0.4031674...    0.9907270...    0.0103302...]
 [ 549.          0.4181943...    0.9930817...    0.0094972...]
 [ 550.          0.43345  ...    0.99495  ...    0.00875  ...]
 [ 551.          0.4489082...    0.9963738...    0.0080748...]
 [ 552.          0.4645599...    0.9973682...    0.0074580...]
 [ 553.          0.4803950...    0.9979568...    0.0068902...]
 [ 554.          0.4963962...    0.9981802...    0.0063660...]
 [ 555.          0.5125410...    0.9980910...    0.0058818...]
 [ 556.          0.5288034...    0.9977488...    0.0054349...]
 [ 557.          0.5451560...    0.9972150...    0.0050216...]
 [ 558.          0.5615719...    0.9965479...    0.0046357...]
```

(continues on next page)

(continued from previous page)

```
[ 559.         0.5780267...   0.9957974...   0.0042671...]
[ 560.         0.5945    ...   0.995    ...   0.0039    ...]]
```

normalise(*factor: Real = 1*) → Self

Normalise the multi-spectral distributions with given normalization factor.

Parameters **factor** (Real) – Normalization factor.

Returns Normalised multi- spectral distribution.

Return type colour.MultiSpectralDistributions

Notes

- The implementation uses the maximum value for each colour.SpectralDistribution class instances.

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.normalise())
...
[[ 500.         0.0082422...   0.3246231...   1.         ...]
 [ 510.         0.0156434...   0.5055276...   0.5816176...]
 [ 520.         0.1064255...   0.7135678...   0.2876838...]
 [ 530.         0.2783852...   0.8663316...   0.155     ...]
 [ 540.         0.4884777...   0.9587939...   0.0746323...]
 [ 550.         0.7291000...   0.9999497...   0.0321691...]
 [ 560.         1.         ...   1.         ...   0.0143382...]]
```

to_sds() → List[*colour.colorimetry.spectrum.SpectralDistribution*]

Convert the multi-spectral distributions to a list of spectral distributions.

Returns List of spectral distributions.

Return type list

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     for sd in msds.to_sds():
...         print(sd)
...
[[ 500.         0.0049 ...]
 [ 510.         0.0093 ...]
 [ 520.         0.06327...]
 [ 530.         0.1655 ...]
 [ 540.         0.2904 ...]
 [ 550.         0.43345...]
 [ 560.         0.5945 ...]]
[[ 500.         0.323  ...]
 [ 510.         0.503  ...]
 [ 520.         0.71   ...]
 [ 530.         0.862  ...]
 [ 540.         0.954  ...]
 [ 550.         0.99495...]
 [ 560.         0.995  ...]]
[[ 500.         0.272  ...]
 [ 510.         0.1582 ...]
 [ 520.         0.07825...]
 [ 530.         0.04216...]
 [ 540.         0.0203 ...]
 [ 550.         0.00875...]
 [ 560.         0.0039 ...]]
```

SPECTRAL_SHAPE_ASTME308	(360, 780, 1).
SPECTRAL_SHAPE_DEFAULT	(360, 780, 1).

colour.SPECTRAL_SHAPE_ASTME308

colour.**SPECTRAL_SHAPE_ASTME308 = SpectralShape(360, 780, 1)**

(360, 780, 1).

References

[ASTMInternational15a]

Type Shape for *ASTM E308-15* practise

colour.SPECTRAL_SHAPE_DEFAULT

`colour.SPECTRAL_SHAPE_DEFAULT = SpectralShape(360, 780, 1)`

(360, 780, 1).

References

[ASTMInternational15a]

Type Shape for *ASTM E308-15* practise

Ancillary Objects

`colour.colorimetry`

reshape_sd(sd[, shape, method, copy])	Reshape given spectral distribution with given spectral shape.
reshape_msds(msds[, shape, method, copy])	Reshape given multi-spectral distributions with given spectral shape.
sds_and_msds_to_sds(sds)	Convert given spectral and multi-spectral distributions to a list of spectral distributions.
sds_and_msds_to_msds(sds)	Convert given spectral and multi-spectral distributions to multi-spectral distributions.

colour.colorimetry.reshape_sd

`colour.colorimetry.reshape_sd`(*sd: colour.colorimetry.spectrum.TypeSpectralDistribution, shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT, method: Union[Literal['Align', 'Extrapolate', 'Interpolate', 'Trim'], str] = 'Align', copy: bool = True, **kwargs: Any*) → colour.colorimetry.spectrum.TypeSpectralDistribution

Reshape given spectral distribution with given spectral shape.

The reshaped object is cached, thus another call to the definition with the same arguments will yield the cached object immediately.

Parameters

- **sd** (`colour.colorimetry.spectrum.TypeSpectralDistribution`) – Spectral distribution to reshape.
- **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape to reshape the spectral distribution with.
- **method** (`Union[Literal['Align', 'Extrapolate', 'Interpolate', 'Trim'], str]`) – Reshape method.
- **copy** (`bool`) – Whether to return a copy of the cached spectral distribution. Default is *True*.

- **kwargs** (Any) – {colour.SpectralDistribution.align(), colour.SpectralDistribution.extrapolate(), colour.SpectralDistribution.interpolate(), colour.SpectralDistribution.trim()}, See the documentation of the previously listed methods.

Return type colour.SpectralDistribution

> **Warning:** Contrary to *Numpy*, reshaping a spectral distribution alters its data!

colour.colorimetry.reshape_msds

colour.colorimetry.**reshape_msds**(*msds: colour.colorimetry.spectrum.TypeMultiSpectralDistributions*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, *method: Union[Literal['Align', 'Extrapolate', 'Interpolate', 'Trim'], str] = 'Align'*, *copy: bool = True*, ***kwargs: Any*) → colour.colorimetry.spectrum.TypeMultiSpectralDistributions

Reshape given multi-spectral distributions with given spectral shape.

The reshaped object is cached, thus another call to the definition with the same arguments will yield the cached object immediately.

Parameters

- **msds** (colour.colorimetry.spectrum.TypeMultiSpectralDistributions) – Spectral distribution to reshape.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape to reshape the multi-spectral distributions with.
- **method** (Union[Literal['Align', 'Extrapolate', 'Interpolate', 'Trim'], str]) – Reshape method.
- **copy** (bool) – Whether to return a copy of the cached multi-spectral distributions. Default is *True*.
- **kwargs** (Any) – {colour.MultiSpectralDistributions.align(), colour.MultiSpectralDistributions.extrapolate(), colour.MultiSpectralDistributions.interpolate(), colour.MultiSpectralDistributions.trim()}, See the documentation of the previously listed methods.

Return type colour.MultiSpectralDistributions

> **Warning:** Contrary to *Numpy*, reshaping a multi-spectral distributions alters its data!

colour.colorimetry.sds_and_msds_to_sds

colour.colorimetry.**sds_and_msds_to_sds**(*sds: collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions*) → List[*colour.colorimetry.spectrum.SpectralDistribution*]

Convert given spectral and multi-spectral distributions to a list of spectral distributions.

Parameters **sds** (collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions) – Spectral and multi-spectral distributions to convert to a list of spectral distributions.

Returns List of spectral distributions.

Return type list

Examples

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> sd_1 = SpectralDistribution(data)
>>> sd_2 = SpectralDistribution(data)
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> multi_sds_1 = MultiSpectralDistributions(data)
>>> multi_sds_2 = MultiSpectralDistributions(data)
>>> len(sds_and_msds_to_sds([sd_1, sd_2, multi_sds_1, multi_sds_2]))
8
```

colour.colorimetry.sds_and_msds_to_msds

colour.colorimetry.**sds_and_msds_to_msds**(*sds: collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions*) → *colour.colorimetry.spectrum.MultiSpectralDistributions*

Convert given spectral and multi-spectral distributions to multi-spectral distributions.

The spectral and multi-spectral distributions will be aligned to the intersection of their spectral shapes.

Parameters **sds** (collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions) – Spectral and multi-spectral distributions to convert to multi-spectral distributions.

Returns Multi-spectral distributions.

Return type colour.MultiSpectralDistributions

Examples

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> sd_1 = SpectralDistribution(data)
>>> sd_2 = SpectralDistribution(data)
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> multi_sds_1 = MultiSpectralDistributions(data)
>>> multi_sds_2 = MultiSpectralDistributions(data)
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True, linewidth=160):
...     sds_and_msds_to_msds(
...         [sd_1, sd_2, multi_sds_1, multi_sds_2]
...     )
...
MultiSpectralDistributions([[ 500.        ,    0.0651   ...,0.0651   ...,    0.0049 ␣
↪ ...,    0.323    ...,    0.272    ...,0.0049   ...,    0.323    ...,    0.272    .
↪..],
                            [ 510.        ,    0.0676692...,0.0676692...,    0.0093 ␣
↪ ...,    0.503    ...,    0.1582   ...,0.0093   ...,    0.503    ...,    0.1582   .
↪..],
                            [ 520.        ,    0.0705   ...,0.0705   ...,    0.06327␣
↪ ...,    0.71     ...,    0.07825  ...,0.06327  ...,    0.71     ...,    0.07825  .
↪..],
                            [ 530.        ,    0.0737808...,0.0737808...,    0.1655 ␣
↪ ...,    0.862    ...,    0.04216  ...,0.1655   ...,    0.862    ...,    0.04216  .
↪..],
                            [ 540.        ,    0.0772   ...,0.0772   ...,    0.2904 ␣
↪ ...,    0.954    ...,    0.0203   ...,0.2904   ...,    0.954    ...,    0.0203   .
↪..],
                            [ 550.        ,    0.0806671...,0.0806671...,    0.43345␣
↪ ...,    0.99495  ...,    0.00875  ...,0.43345  ...,    0.99495  ...,    0.00875  .
↪..],
                            [ 560.        ,    0.087    ...,0.087    ...,    0.5945 ␣
↪ ...,    0.995    ...,    0.0039   ...,0.5945   ...,    0.995    ...,    0.0039   .
↪..]],
                           labels=['SpectralDistribution (...)',
↪'SpectralDistribution (...)', '0 - SpectralDistribution (...)', '1 -␣
↪SpectralDistribution (...)', '2 - SpectralDistribution (...)', '0 -␣
↪SpectralDistribution (...)', '1 - SpectralDistribution (...)', '2 -␣
↪SpectralDistribution (...)'],
```

(continues on next page)

(continued from previous page)

```
                    interpolator=SpragueInterpolator,
                    interpolator_kwargs={},
                    extrapolator=Extrapolator,
                    extrapolator_kwargs={...})
```

Spectral Data Generation

colour

sd_CIE_standard_illuminant_A([shape])	*CIE Standard Illuminant A* is intended to represent typical, domestic, tungsten-filament lighting.
sd_CIE_illuminant_D_series(xy[, ...])	Return the spectral distribution of given *CIE Illuminant D Series* using given *CIE xy* chromaticity coordinates.
sd_blackbody(temperature[, shape, c1, c2, n])	Return the spectral distribution of the planckian radiator for given temperature $T[K]$ with values in *watts per steradian per square metre per nanometer* ($W/sr/m^2/nm$).
sd_rayleigh_jeans(temperature[, shape])	Return the spectral distribution of the planckian radiator for given temperature $T[K]$ with values in *watts per steradian per square metre per nanometer* ($W/sr/m^2/nm$) according to *Rayleigh-Jeans* law.
sd_constant(k[, shape])	Return a spectral distribution of given spectral shape filled with constant k values.
sd_ones([shape])	Return a spectral distribution of given spectral shape filled with ones.
sd_zeros([shape])	Return a spectral distribution of given spectral shape filled with zeros.
msds_constant(k, labels[, shape])	Return the multi-spectral distributions with given labels and given spectral shape filled with constant k values.
msds_ones(labels[, shape])	Return the multi-spectral distributionss with given labels and given spectral shape filled with ones.
msds_zeros(labels[, shape])	Return the multi-spectral distributionss with given labels and given spectral shape filled with zeros.
SD_GAUSSIAN_METHODS	Supported gaussian spectral distribution computation methods.
sd_gaussian(mu_peak_wavelength, sigma_fwhm)	Return a gaussian spectral distribution of given spectral shape using given method.
SD_SINGLE_LED_METHODS	Supported single *LED* spectral distribution computation methods.
sd_single_led(peak_wavelength[, shape, method])	Return a single *LED* spectral distribution of given spectral shape at given peak wavelength according to given method.
SD_MULTI_LEDS_METHODS	Supported multi *LED* spectral distribution computation methods.
sd_multi_leds(peak_wavelengths[, shape, method])	Return a multi *LED* spectral distribution of given spectral shape at given peak wavelengths.

colour.sd_CIE_standard_illuminant_A

colour.**sd_CIE_standard_illuminant_A**(*shape*: colour.colorimetry.spectrum.SpectralShape = *SPECTRAL_SHAPE_DEFAULT*) → *colour.colorimetry.spectrum.SpectralDistribution*

CIE Standard Illuminant A is intended to represent typical, domestic, tungsten-filament lighting.

Its spectral distribution is that of a Planckian radiator at a temperature of approximately 2856 K. *CIE Standard Illuminant A* should be used in all applications of colorimetry involving the use of incandescent lighting, unless there are specific reasons for using a different illuminant.

Parameters **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape used to create the spectral distribution of the *CIE Standard Illuminant A*.

Returns *CIE Standard Illuminant A*. spectral distribution.

Return type `colour.SpectralDistribution`

References

[CIET14804a]

Examples

```
>>> from colour import SpectralShape
>>> sd_CIE_standard_illuminant_A(SpectralShape(400, 700, 10))
...
SpectralDistribution([[ 400.        ,   14.7080384...],
                      [ 410.        ,   17.6752521...],
                      [ 420.        ,   20.9949572...],
                      [ 430.        ,   24.6709226...],
                      [ 440.        ,   28.7027304...],
                      [ 450.        ,   33.0858929...],
                      [ 460.        ,   37.8120566...],
                      [ 470.        ,   42.8692762...],
                      [ 480.        ,   48.2423431...],
                      [ 490.        ,   53.9131532...],
                      [ 500.        ,   59.8610989...],
                      [ 510.        ,   66.0634727...],
                      [ 520.        ,   72.4958719...],
                      [ 530.        ,   79.1325945...],
                      [ 540.        ,   85.9470183...],
                      [ 550.        ,   92.9119589...],
                      [ 560.        ,  100.        ...],
                      [ 570.        ,  107.1837952...],
                      [ 580.        ,  114.4363383...],
                      [ 590.        ,  121.7312009...],
                      [ 600.        ,  129.0427389...],
                      [ 610.        ,  136.3462674...],
                      [ 620.        ,  143.6182057...],
                      [ 630.        ,  150.8361944...],
                      [ 640.        ,  157.9791857...],
                      [ 650.        ,  165.0275098...],
                      [ 660.        ,  171.9629200...],
                      [ 670.        ,  178.7686175...],
                      [ 680.        ,  185.4292591...],
                      [ 690.        ,  191.9309499...],
```

(continues on next page)

(continued from previous page)

```
                        [ 700.        ,  198.2612232...]],
                       SpragueInterpolator,
                       {},
                       Extrapolator,
                       {'method': 'Constant', 'left': None, 'right': None})
```

colour.sd_CIE_illuminant_D_series

colour.**sd_CIE_illuminant_D_series**(*xy: ArrayLike*, *M1_M2_rounding: bool = True*, *shape: colour.colorimetry.spectrum.SpectralShape | None = None*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return the spectral distribution of given *CIE Illuminant D Series* using given *CIE xy* chromaticity coordinates.

Parameters

- **xy** (ArrayLike) – *CIE xy* chromaticity coordinates.
- **M1_M2_rounding** (bool) – Whether to round $M1$ and $M2$ variables to 3 decimal places in order to yield the internationally agreed values.
- **shape** (colour.colorimetry.spectrum.SpectralShape | None) – Specifies the shape of the returned SpectralDistribution. Optional, default None.

Returns *CIE Illuminant D Series* spectral distribution.

Return type colour.SpectralDistribution

Notes

- The nominal *CIE xy* chromaticity coordinates which have been computed with colour.temperature.CCT_to_xy_CIE_D() must be given according to *CIE 015:2004* recommendation and thus multiplied by 1.4388 / 1.4380.
- $M1$ **and** $M2$ **variables are rounded to 3 decimal places** according to *CIE 015:2004* recommendation.

References

[CIET14804i], [WS00k]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> from colour.temperature import CCT_to_xy_CIE_D
>>> CCT_D65 = 6500 * 1.4388 / 1.4380
>>> xy = CCT_to_xy_CIE_D(CCT_D65)
>>> with numpy_print_options(suppress=True):
...     sd_CIE_illuminant_D_series(xy)
...
SpectralDistribution([[ 300.      ,    0.0341...],
                      [ 305.      ,    1.6643...],
                      [ 310.      ,    3.2945...],
                      [ 315.      ,   11.7652...],
                      [ 320.      ,   20.236 ...],
```

(continues on next page)

(continued from previous page)

```
                    [ 325.     ,   28.6447...],
                    [ 330.     ,   37.0535...],
                    [ 335.     ,   38.5011...],
                    [ 340.     ,   39.9488...],
                    [ 345.     ,   42.4302...],
                    [ 350.     ,   44.9117...],
                    [ 355.     ,   45.775 ...],
                    [ 360.     ,   46.6383...],
                    [ 365.     ,   49.3637...],
                    [ 370.     ,   52.0891...],
                    [ 375.     ,   51.0323...],
                    [ 380.     ,   49.9755...],
                    [ 385.     ,   52.3118...],
                    [ 390.     ,   54.6482...],
                    [ 395.     ,   68.7015...],
                    [ 400.     ,   82.7549...],
                    [ 405.     ,   87.1204...],
                    [ 410.     ,   91.486 ...],
                    [ 415.     ,   92.4589...],
                    [ 420.     ,   93.4318...],
                    [ 425.     ,   90.0570...],
                    [ 430.     ,   86.6823...],
                    [ 435.     ,   95.7736...],
                    [ 440.     ,  104.8649...],
                    [ 445.     ,  110.9362...],
                    [ 450.     ,  117.0076...],
                    [ 455.     ,  117.4099...],
                    [ 460.     ,  117.8122...],
                    [ 465.     ,  116.3365...],
                    [ 470.     ,  114.8609...],
                    [ 475.     ,  115.3919...],
                    [ 480.     ,  115.9229...],
                    [ 485.     ,  112.3668...],
                    [ 490.     ,  108.8107...],
                    [ 495.     ,  109.0826...],
                    [ 500.     ,  109.3545...],
                    [ 505.     ,  108.5781...],
                    [ 510.     ,  107.8017...],
                    [ 515.     ,  106.2957...],
                    [ 520.     ,  104.7898...],
                    [ 525.     ,  106.2396...],
                    [ 530.     ,  107.6895...],
                    [ 535.     ,  106.0475...],
                    [ 540.     ,  104.4055...],
                    [ 545.     ,  104.2258...],
                    [ 550.     ,  104.0462...],
                    [ 555.     ,  102.0231...],
                    [ 560.     ,  100.     ...],
                    [ 565.     ,   98.1671...],
                    [ 570.     ,   96.3342...],
                    [ 575.     ,   96.0611...],
                    [ 580.     ,   95.788 ...],
                    [ 585.     ,   92.2368...],
                    [ 590.     ,   88.6856...],
                    [ 595.     ,   89.3459...],
                    [ 600.     ,   90.0062...],
```

(continues on next page)

(continued from previous page)

```
                      [ 605.     ,   89.8026...],
                      [ 610.     ,   89.5991...],
                      [ 615.     ,   88.6489...],
                      [ 620.     ,   87.6987...],
                      [ 625.     ,   85.4936...],
                      [ 630.     ,   83.2886...],
                      [ 635.     ,   83.4939...],
                      [ 640.     ,   83.6992...],
                      [ 645.     ,   81.863 ...],
                      [ 650.     ,   80.0268...],
                      [ 655.     ,   80.1207...],
                      [ 660.     ,   80.2146...],
                      [ 665.     ,   81.2462...],
                      [ 670.     ,   82.2778...],
                      [ 675.     ,   80.281 ...],
                      [ 680.     ,   78.2842...],
                      [ 685.     ,   74.0027...],
                      [ 690.     ,   69.7213...],
                      [ 695.     ,   70.6652...],
                      [ 700.     ,   71.6091...],
                      [ 705.     ,   72.9790...],
                      [ 710.     ,   74.349 ...],
                      [ 715.     ,   67.9765...],
                      [ 720.     ,   61.604 ...],
                      [ 725.     ,   65.7448...],
                      [ 730.     ,   69.8856...],
                      [ 735.     ,   72.4863...],
                      [ 740.     ,   75.087 ...],
                      [ 745.     ,   69.3398...],
                      [ 750.     ,   63.5927...],
                      [ 755.     ,   55.0054...],
                      [ 760.     ,   46.4182...],
                      [ 765.     ,   56.6118...],
                      [ 770.     ,   66.8054...],
                      [ 775.     ,   65.0941...],
                      [ 780.     ,   63.3828...],
                      [ 785.     ,   63.8434...],
                      [ 790.     ,   64.304 ...],
                      [ 795.     ,   61.8779...],
                      [ 800.     ,   59.4519...],
                      [ 805.     ,   55.7054...],
                      [ 810.     ,   51.959 ...],
                      [ 815.     ,   54.6998...],
                      [ 820.     ,   57.4406...],
                      [ 825.     ,   58.8765...],
                      [ 830.     ,   60.3125...]],
                     LinearInterpolator,
                     {},
                     Extrapolator,
                     {'method': 'Constant', 'left': None, 'right': None})
```

colour.sd_blackbody

colour.**sd_blackbody**(*temperature: float*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, *c1: float = CONSTANT_C1*, *c2: float = CONSTANT_C2*, *n: float = CONSTANT_N*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return the spectral distribution of the planckian radiator for given temperature $T[K]$ with values in *watts per steradian per square metre per nanometer* $(W/sr/m^2/nm)$.

Parameters

- **temperature** (float) – Temperature $T[K]$ in kelvin degrees.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution of the planckian radiator.
- **c1** (float) – 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** (float) – 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 al., 1991)*, namely $c2 = 1,4388x10.2\ m/K$.
- **n** (float) – Medium index of refraction. For dry air at 15C and 101 325 Pa, containing 0,03 percent by volume of carbon dioxide, it is approximately 1,00028 throughout the visible region although *CIE 15:2004* recommends using $n = 1$.

Returns Blackbody spectral distribution with values in *watts per steradian per square metre per nanometer* $(W/sr/m^2/nm)$.

Return type colour.SpectralDistribution

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_blackbody(5000, shape=SpectralShape(400, 700, 20))
...
...
SpectralDistribution([[   400.        ,   8742.5713329...],
                      [   420.        ,   9651.6810212...],
                      [   440.        ,  10447.3423137...],
                      [   460.        ,  11121.8597759...],
                      [   480.        ,  11673.7121534...],
                      [   500.        ,  12106.0645344...],
                      [   520.        ,  12425.4166118...],
                      [   540.        ,  12640.4550541...],
                      [   560.        ,  12761.1284859...],
                      [   580.        ,  12797.9345572...],
                      [   600.        ,  12761.3938171...],
                      [   620.        ,  12661.6795247...],
                      [   640.        ,  12508.3723863...],
                      [   660.        ,  12310.3119640...],
                      [   680.        ,  12075.5205176...],
                      [   700.        ,  11811.1793602...]],
                     SpragueInterpolator,
                     {},
```

(continues on next page)

(continued from previous page)

```
                              Extrapolator,
                              {'method': 'Constant', 'left': None, 'right': None})
```

colour.sd_rayleigh_jeans

colour.**sd_rayleigh_jeans**(*temperature: float*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return the spectral distribution of the planckian radiator for given temperature $T[K]$ with values in *watts per steradian per square metre per nanometer* ($W/sr/m^2/nm$) according to *Rayleigh-Jeans* law.

Parameters

- **temperature** (float) – Temperature $T[K]$ in kelvin degrees.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution of the planckian radiator.

Returns Blackbody spectral distribution with values in *watts per steradian per square metre per nanometer* ($W/sr/m^2/nm$).

Return type colour.SpectralDistribution

Notes

- The *Rayleigh-Jeans* law agrees with experimental results at large wavelengths (low frequencies) but strongly disagrees at short wavelengths (high frequencies). This inconsistency between observations and the predictions of classical physics is commonly known as the *ultraviolet catastrophe*.

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_rayleigh_jeans(5000, shape=SpectralShape(400, 700, 20))
...
...
SpectralDistribution([[     400.        ,  1616829.9106941...],
                      [     420.        ,  1330169.9688456...],
                      [     440.        ,  1104316.5840408...],
                      [     460.        ,   924427.7490112...],
                      [     480.        ,   779721.2146480...],
                      [     500.        ,   662253.5314203...],
                      [     520.        ,   566097.0941823...],
                      [     540.        ,   486776.1157138...],
                      [     560.        ,   420874.0917050...],
                      [     580.        ,   365756.7299433...],
                      [     600.        ,   319373.8095198...],
                      [     620.        ,   280115.7588306...],
                      [     640.        ,   246708.6655722...],
                      [     660.        ,   218136.6091932...],
                      [     680.        ,   193583.6389284...],
                      [     700.        ,   172390.0279623...]],
                     SpragueInterpolator,
```

(continues on next page)

(continued from previous page)

```
                      {},
                      Extrapolator,
                      {'method': 'Constant', 'left': None, 'right': None})
```

colour.sd_constant

colour.**sd_constant**(*k: float*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, ***kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a spectral distribution of given spectral shape filled with constant k values.

Parameters

- **k** (float) – Constant k to fill the spectral distribution with.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution.
- **kwargs** (Any) – {colour.SpectralDistribution}, See the documentation of the previously listed class.

Returns Constant k filled spectral distribution.

Return type colour.SpectralDistribution

Notes

- By default, the spectral distribution will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.
- The interpolator is set to colour.LinearInterpolator class.

Examples

```
>>> sd = sd_constant(100)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
100.0
```

colour.sd_ones

colour.**sd_ones**(*shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, ***kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a spectral distribution of given spectral shape filled with ones.

Parameters

- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution.
- **kwargs** (Any) – {colour.sd_constant()}, See the documentation of the previously listed definition.

Returns Ones filled spectral distribution.

Return type colour.SpectralDistribution

Notes

- By default, the spectral distribution will use the shape given by `colour.SPECTRAL_SHAPE_DEFAULT` attribute.
- The interpolator is set to `colour.LinearInterpolator` class.

Examples

```
>>> sd = sd_ones()
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
1.0
```

colour.sd_zeros

`colour.sd_zeros`(*shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, ****kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a spectral distribution of given spectral shape filled with zeros.

Parameters

- **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape used to create the spectral distribution.
- **kwargs** (`Any`) – {`colour.sd_constant()`}, See the documentation of the previously listed definition.

Returns Zeros filled spectral distribution.

Return type `colour.SpectralDistribution`

Notes

- By default, the spectral distribution will use the shape given by `colour.SPECTRAL_SHAPE_DEFAULT` attribute.
- The interpolator is set to `colour.LinearInterpolator` class.

Examples

```
>>> sd = sd_zeros()
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
0.0
```

colour.msds_constant

colour.**msds_constant**(*k: float*, *labels: collections.abc.Sequence*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, ***kwargs: Any*) → *colour.colorimetry.spectrum.MultiSpectralDistributions*

Return the multi-spectral distributions with given labels and given spectral shape filled with constant k values.

Parameters

- **k** (`float`) – Constant k to fill the multi-spectral distributions with.
- **labels** (`collections.abc.Sequence`) – Names to use for the `colour.SpectralDistribution` class instances.
- **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape used to create the multi-spectral distributions.
- **kwargs** (`Any`) – {`colour.MultiSpectralDistributions`}, See the documentation of the previously listed class.

Returns Constant k filled multi-spectral distributions.

Return type `colour.MultiSpectralDistributions`

Notes

- By default, the multi-spectral distributions will use the shape given by `colour.SPECTRAL_SHAPE_DEFAULT` attribute.
- The interpolator is set to `colour.LinearInterpolator` class.

Examples

```
>>> msds = msds_constant(100, labels=["a", "b", "c"])
>>> msds.shape
SpectralShape(360.0, 780.0, 1.0)
>>> msds[400]
array([ 100.,  100.,  100.])
>>> msds.labels
['a', 'b', 'c']
```

colour.msds_ones

colour.**msds_ones**(*labels: collections.abc.Sequence*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, ***kwargs: Any*) → *colour.colorimetry.spectrum.MultiSpectralDistributions*

Return the multi-spectral distributionss with given labels and given spectral shape filled with ones.

Parameters

- **labels** (`collections.abc.Sequence`) – Names to use for the `colour.SpectralDistribution` class instances.
- **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape used to create the multi-spectral distributions.
- **kwargs** (`Any`) – {`colour.msds_constant()`}, See the documentation of the previously listed definition.

Returns Ones filled multi-spectral distributions.

Return type colour.MultiSpectralDistributions

Notes

- By default, the multi-spectral distributions will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.
- The interpolator is set to colour.LinearInterpolator class.

Examples

```
>>> msds = msds_ones(labels=["a", "b", "c"])
>>> msds.shape
SpectralShape(360.0, 780.0, 1.0)
>>> msds[400]
array([ 1.,  1.,  1.])
>>> msds.labels
['a', 'b', 'c']
```

colour.msds_zeros

colour.**msds_zeros**(*labels: collections.abc.Sequence*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, ***kwargs: Any*) → *colour.colorimetry.spectrum.MultiSpectralDistributions*

Return the multi-spectral distributionss with given labels and given spectral shape filled with zeros.

Parameters

- **labels** (collections.abc.Sequence) – Names to use for the colour.SpectralDistribution class instances.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the multi-spectral distributions.
- **kwargs** (Any) – {colour.msds_constant()}, See the documentation of the previously listed definition.

Returns Zeros filled multi-spectral distributions.

Return type colour.MultiSpectralDistributions

Notes

- By default, the multi-spectral distributions will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.
- The interpolator is set to colour.LinearInterpolator class.

Examples

```
>>> msds = msds_zeros(labels=["a", "b", "c"])
>>> msds.shape
SpectralShape(360.0, 780.0, 1.0)
>>> msds[400]
array([ 0.,  0.,  0.])
>>> msds.labels
['a', 'b', 'c']
```

colour.SD_GAUSSIAN_METHODS

colour.**SD_GAUSSIAN_METHODS = CanonicalMapping({'Normal': ..., 'FWHM': ...})**

Supported gaussian spectral distribution computation methods.

colour.sd_gaussian

colour.**sd_gaussian**(*mu_peak_wavelength: float, sigma_fwhm: float, shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT, method: Union[Literal['Normal', 'FWHM'], str] = 'Normal', **kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a gaussian spectral distribution of given spectral shape using given method.

Parameters

- **mu_peak_wavelength** (float) – Mean wavelength μ the gaussian spectral distribution will peak at.
- **sigma_fwhm** (float) – Standard deviation $sigma$ of the gaussian spectral distribution or full width at half maximum, i.e. width of the gaussian spectral distribution measured between those points on the y axis which are half the maximum amplitude.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution.
- **method** (Union[Literal['Normal', 'FWHM'], str]) – Computation method.
- **kwargs** (Any) – {colour.colorimetry.sd_gaussian_normal(), colour.colorimetry.sd_gaussian_fwhm()}, See the documentation of the previously listed definitions.

Returns Gaussian spectral distribution.

Return type colour.SpectralDistribution

Notes

- By default, the spectral distribution will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.

Examples

```
>>> sd = sd_gaussian(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530]
0.6065306...
>>> sd = sd_gaussian(555, 25, method="FWHM")
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530]
0.0625
```

colour.SD_SINGLE_LED_METHODS

colour.**SD_SINGLE_LED_METHODS = CanonicalMapping({'Ohno 2005': ...})**

Supported single *LED* spectral distribution computation methods.

colour.sd_single_led

colour.**sd_single_led**(*peak_wavelength: float, shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT, method: Union[Literal['Ohno 2005'], str] = 'Ohno 2005', **kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a single *LED* spectral distribution of given spectral shape at given peak wavelength according to given method.

Parameters

- **peak_wavelength** (float) – Wavelength the single *LED* spectral distribution will peak at.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution.
- **method** (Union[Literal['Ohno 2005'], str]) – Computation method.
- **kwargs** (Any) – {colour.colorimetry.sd_single_led_Ohno2005()}, See the documentation of the previously listed definition.

Returns Single *LED* spectral distribution.

Return type colour.SpectralDistribution

Notes

- By default, the spectral distribution will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_single_led(555, half_spectral_width=25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1...
```

colour.SD_MULTI_LEDS_METHODS

colour.**SD_MULTI_LEDS_METHODS = CanonicalMapping({'Ohno 2005': ...})**
Supported multi *LED* spectral distribution computation methods.

colour.sd_multi_leds

colour.**sd_multi_leds**(*peak_wavelengths: ArrayLike, shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT, method: Union[Literal['Ohno 2005'], str] = 'Ohno 2005', **kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a multi *LED* spectral distribution of given spectral shape at given peak wavelengths.

Parameters

- **peak_wavelengths** (ArrayLike) – Wavelengths the multi *LED* spectral distribution will peak at, i.e. the peaks for each generated single *LED* spectral distributions.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution.
- **method** (Union[Literal['Ohno 2005'], str]) – Computation method.
- **kwargs** (Any) – {colour.colorimetry.sd_multi_leds_Ohno2005()}, See the documentation of the previously listed definition.

Returns Multi *LED* spectral distribution.

Return type colour.SpectralDistribution

Notes

- By default, the spectral distribution will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_multi_leds(
...     np.array([457, 530, 615]),
...     half_spectral_widths=np.array([20, 30, 20]),
...     peak_power_ratios=np.array([0.731, 1.000, 1.660]),
... )
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[500]
0.1295132...
```

`colour.colorimetry`

`blackbody_spectral_radiance`(wavelength, ...)	Return the spectral radiance of a blackbody as a function of wavelength at thermodynamic temperature $T[K]$ in a medium having index of refraction n.
`planck_law`(wavelength, temperature[, c1, c2, n])	Return the spectral radiance of a blackbody as a function of wavelength at thermodynamic temperature $T[K]$ in a medium having index of refraction n.
`rayleigh_jeans_law`(wavelength, temperature)	Return the approximation of the spectral radiance of a blackbody as a function of wavelength at thermodynamic temperature $T[K]$ according to *Rayleigh-Jeans* law.
`daylight_locus_function`(x_D)	Return the daylight locus as *CIE xy* chromaticity coordinates.
`sd_gaussian_normal`(mu, sigma[, shape])	Return a gaussian spectral distribution of given spectral shape at given mean wavelength μ and standard deviation $sigma$.
`sd_gaussian_fwhm`(peak_wavelength, fwhm[, shape])	Return a gaussian spectral distribution of given spectral shape at given peak wavelength and full width at half maximum.
`sd_single_led_Ohno2005`(peak_wavelength, ...)	Return a single *LED* spectral distribution of given spectral shape at given peak wavelength and half spectral width $\Delta\lambda_{0.5}$ according to *Ohno (2005)* method.
`sd_multi_leds_Ohno2005`(peak_wavelengths, ...)	Return a multi *LED* spectral distribution of given spectral shape at given peak wavelengths, half spectral widths $\Delta\lambda_{0.5}$ and peak power ratios according to *Ohno (2005)* method.

colour.colorimetry.blackbody_spectral_radiance

colour.colorimetry.**blackbody_spectral_radiance**(*wavelength: ArrayLike*, *temperature: ArrayLike*, *c1: float = CONSTANT_C1*, *c2: float = CONSTANT_C2*, *n: float = CONSTANT_N*) → NDArrayFloat

Return the spectral radiance of a blackbody as a function of wavelength at thermodynamic temperature $T[K]$ in a medium having index of refraction n.

Parameters

- **wavelength** (ArrayLike) – Wavelength in meters.
- **temperature** (ArrayLike) – Temperature $T[K]$ in kelvin degrees.
- **c1** (float) – 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** (float) – 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 al., 1991)*, namely $c2 = 1,4388x10.2\ m/K$.
- **n** (float) – Medium index of refraction. For dry air at 15C and 101 325 Pa, containing 0,03 percent by volume of carbon dioxide, it is approximately 1,00028 throughout the visible region although *CIE 15:2004* recommends using $n = 1$.

Returns Radiance in *watts per steradian per square metre* ($W/sr/m^2$).

Return type numpy.ndarray

> **Warning:** The colour.colorimetry.planck_law() definition behaviour with n-dimensional arrays is unusual: The wavelength and temperature parameters are first raveled using numpy.ravel(). Then, they are *broadcasted* together by transposing the temperature parameter. Finally, and for convenience, the return value is squeezed using numpy.squeeze().

Notes

- The following implementation is expressed in terms of wavelength.
- The SI unit of radiance is *watts per steradian per square metre* ($W/sr/m^2$).

References

[CIET14804e]

Examples

```
>>> planck_law(500 * 1e-9, 5500)
20472701909806.5...
>>> planck_law(500 * 1e-9, [5000, 5500, 6000])
array([  1.2106064...e+13,   2.0472701...e+13,   3.1754431...e+13])
```

colour.colorimetry.planck_law

`colour.colorimetry.planck_law`(*wavelength: ArrayLike, temperature: ArrayLike, c1: float = CONSTANT_C1, c2: float = CONSTANT_C2, n: float = CONSTANT_N*) → NDArrayFloat

Return the spectral radiance of a blackbody as a function of wavelength at thermodynamic temperature $T[K]$ in a medium having index of refraction n.

Parameters

- **wavelength** (ArrayLike) – Wavelength in meters.
- **temperature** (ArrayLike) – Temperature $T[K]$ in kelvin degrees.
- **c1** (float) – 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** (float) – 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 al., 1991)*, namely $c2 = 1,4388x10.2\ m/K$.
- **n** (float) – Medium index of refraction. For dry air at 15C and 101 325 Pa, containing 0,03 percent by volume of carbon dioxide, it is approximately 1,00028 throughout the visible region although *CIE 15:2004* recommends using $n = 1$.

Returns Radiance in *watts per steradian per square metre* ($W/sr/m^2$).

Return type numpy.ndarray

> **Warning:** The `colour.colorimetry.planck_law()` definition behaviour with n-dimensional arrays is unusual: The `wavelength` and `temperature` parameters are first raveled using `numpy.ravel()`. Then, they are *broadcasted* together by transposing the `temperature` parameter. Finally, and for convenience, the return value is squeezed using `numpy.squeeze()`.

Notes

- The following implementation is expressed in terms of wavelength.
- The SI unit of radiance is *watts per steradian per square metre* ($W/sr/m^2$).

References

[CIET14804e]

Examples

```
>>> planck_law(500 * 1e-9, 5500)
20472701909806.5...
>>> planck_law(500 * 1e-9, [5000, 5500, 6000])
array([  1.2106064...e+13,   2.0472701...e+13,   3.1754431...e+13])
```

colour.colorimetry.rayleigh_jeans_law

`colour.colorimetry.rayleigh_jeans_law(wavelength: ArrayLike, temperature: ArrayLike) → NDArrayFloat`

Return the approximation of the spectral radiance of a blackbody as a function of wavelength at thermodynamic temperature $T[K]$ according to *Rayleigh-Jeans* law.

Parameters

- **wavelength** (ArrayLike) – Wavelength in meters.
- **temperature** (ArrayLike) – Temperature $T[K]$ in kelvin degrees.

Returns Radiance in *watts per steradian per square metre* ($W/sr/m^2$).

Return type numpy.ndarray

Warning: The `colour.colorimetry.rayleigh_jeans_law()` definition behaviour with n-dimensional arrays is unusual: The `wavelength` and `temperature` parameters are first raveled using `numpy.ravel()`. Then, they are *broadcasted* together by transposing the `temperature` parameter. Finally, and for convenience, the return value is squeezed using `numpy.squeeze()`.

Notes

- The *Rayleigh-Jeans* law agrees with experimental results at large wavelengths (low frequencies) but strongly disagrees at short wavelengths (high frequencies). This inconsistency between observations and the predictions of classical physics is commonly known as the *ultraviolet catastrophe*.
- The following implementation is expressed in terms of wavelength.
- The SI unit of radiance is *watts per steradian per square metre* ($W/sr/m^2$).

References

[Wikipedia03e]

Examples

```
>>> rayleigh_jeans_law(500 * 1e-9, 5500)
728478884562351.5...
>>> rayleigh_jeans_law(500 * 1e-9, [5000, 5500, 6000])
...
array([  6.6225353...e+14,   7.2847888...e+14,   7.9470423...e+14])
```

colour.colorimetry.daylight_locus_function

`colour.colorimetry.daylight_locus_function(x_D: ArrayLike) → NDArrayFloat`

Return the daylight locus as *CIE xy* chromaticity coordinates.

Parameters **x_D** (ArrayLike) – Chromaticity coordinate x_D.

Returns Daylight locus as *CIE xy* chromaticity coordinates.

Return type numpy.ndarray

References

[WS00a]

Examples

```
>>> daylight_locus_function(0.31270)
0.3291051...
```

colour.colorimetry.sd_gaussian_normal

colour.colorimetry.**sd_gaussian_normal**(*mu: float, sigma: float, shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT, **kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a gaussian spectral distribution of given spectral shape at given mean wavelength μ and standard deviation $sigma$.

Parameters

- **mu** (float) – Mean wavelength μ the gaussian spectral distribution will peak at.
- **sigma** (float) – Standard deviation $sigma$ of the gaussian spectral distribution.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution.
- **kwargs** (Any) – {colour.SpectralDistribution}, See the documentation of the previously listed class.

Returns Gaussian spectral distribution.

Return type colour.SpectralDistribution

Notes

- By default, the spectral distribution will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.

Examples

```
>>> sd = sd_gaussian_normal(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530]
0.6065306...
```

colour.colorimetry.sd_gaussian_fwhm

colour.colorimetry.**sd_gaussian_fwhm**(*peak_wavelength: float*, *fwhm: float*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, ***kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a gaussian spectral distribution of given spectral shape at given peak wavelength and full width at half maximum.

Parameters

- **peak_wavelength** (float) – Wavelength the gaussian spectral distribution will peak at.
- **fwhm** (float) – Full width at half maximum, i.e. width of the gaussian spectral distribution measured between those points on the y axis which are half the maximum amplitude.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution.
- **kwargs** (Any) – {colour.SpectralDistribution}, See the documentation of the previously listed class.

Returns Gaussian spectral distribution.

Return type colour.SpectralDistribution

Notes

- By default, the spectral distribution will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.

Examples

```
>>> sd = sd_gaussian_fwhm(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530]
0.0625
```

colour.colorimetry.sd_single_led_Ohno2005

colour.colorimetry.**sd_single_led_Ohno2005**(*peak_wavelength: float*, *half_spectral_width: float*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT*, ***kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a single *LED* spectral distribution of given spectral shape at given peak wavelength and half spectral width $\Delta\lambda_{0.5}$ according to *Ohno (2005)* method.

Parameters

- **peak_wavelength** (float) – Wavelength the single *LED* spectral distribution will peak at.
- **half_spectral_width** (float) – Half spectral width $\Delta\lambda_{0.5}$.

- **shape** (colour.colorimetry.spectrum.SpectralShape) – Spectral shape used to create the spectral distribution.
- **kwargs** (Any) – {colour.SpectralDistribution}, See the documentation of the previously listed class.

Returns Single *LED* spectral distribution.

Return type colour.SpectralDistribution

Notes

- By default, the spectral distribution will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_single_led_Ohno2005(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1...
```

colour.colorimetry.sd_multi_leds_Ohno2005

colour.colorimetry.**sd_multi_leds_Ohno2005**(*peak_wavelengths: ArrayLike, half_spectral_widths: ArrayLike, peak_power_ratios: ArrayLike | None = None, shape: SpectralShape = SPECTRAL_SHAPE_DEFAULT, **kwargs: Any*) → *SpectralDistribution*

Return a multi *LED* spectral distribution of given spectral shape at given peak wavelengths, half spectral widths $\Delta\lambda_{0.5}$ and peak power ratios according to *Ohno (2005)* method.

The multi *LED* spectral distribution is generated using many single *LED* spectral distributions generated with colour.sd_single_led_Ohno2005() definition.

Parameters

- **peak_wavelengths** (ArrayLike) – Wavelengths the multi *LED* spectral distribution will peak at, i.e. the peaks for each generated single *LED* spectral distributions.
- **half_spectral_widths** (ArrayLike) – Half spectral widths $\Delta\lambda_{0.5}$.
- **peak_power_ratios** (ArrayLike | None) – Peak power ratios for each generated single *LED* spectral distributions.
- **shape** (SpectralShape) – Spectral shape used to create the spectral distribution.
- **kwargs** (Any) – {colour.colorimetry.sd_single_led_Ohno2005()}, See the documentation of the previously listed definition.

Returns Multi *LED* spectral distribution.

Return type colour.SpectralDistribution

Notes

- By default, the spectral distribution will use the shape given by `colour.SPECTRAL_SHAPE_DEFAULT` attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_multi_leds_Ohno2005(
...     np.array([457, 530, 615]),
...     np.array([20, 30, 20]),
...     np.array([0.731, 1.000, 1.660]),
... )
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[500]
0.1295132...
```

Aliases

`colour.colorimetry`

`planck_law`(wavelength, temperature[, c1, c2, n])	Return the spectral radiance of a blackbody as a function of wavelength at thermodynamic temperature $T[K]$ in a medium having index of refraction n.

Conversion to Tristimulus Values

`colour`

`sd_to_XYZ`(sd[, cmfs, illuminant, k, method])	Convert given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions, illuminant and method.
`SD_TO_XYZ_METHODS`	Supported spectral distribution to *CIE XYZ* tristimulus values conversion methods.
`msds_to_XYZ`(msds[, cmfs, illuminant, k, method])	Convert given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant.
`MSDS_TO_XYZ_METHODS`	Supported multi-spectral array to *CIE XYZ* tristimulus values conversion methods.
`wavelength_to_XYZ`(wavelength[, cmfs])	Convert given wavelength λ to *CIE XYZ* tristimulus values using given colour matching functions.

colour.sd_to_XYZ

colour.**sd_to_XYZ**(*sd: ArrayLike* | SpectralDistribution | MultiSpectralDistributions, *cmfs:* MultiSpectralDistributions | *None* = *None*, *illuminant:* SpectralDistribution | *None* = *None*, *k: Real* | *None* = *None*, *method: Literal['ASTM E308', 'Integration']* | *str* = *'ASTM E308'*, ***kwargs:* Any) → NDArrayFloat

Convert given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions, illuminant and method.

If method is *Integration*, the spectral distribution can be either a colour.SpectralDistribution class instance or an *ArrayLike* in which case the shape must be passed.

Parameters

- **sd** (ArrayLike | SpectralDistribution | MultiSpectralDistributions) – Spectral distribution, if an *ArrayLike* and method is *Integration* the wavelengths are expected to be in the last axis, e.g. for a spectral array with 77 bins, sd shape could be (77,) or (1, 77).
- **cmfs** (MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **k** (Real | None) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.
- **method** (Literal['ASTM E308', 'Integration'] | str) – Computation method.
- **mi_5nm_omission_method** – {colour.colorimetry.sd_to_XYZ_ASTME308()}, 5 nm measurement intervals spectral distribution conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** – {colour.colorimetry.sd_to_XYZ_ASTME308()}, 20 nm measurement intervals spectral distribution conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **shape** – Spectral shape of the spectral distribution, cmfs and illuminant will be aligned to it if sd is an *ArrayLike*.
- **use_practice_range** – {colour.colorimetry.sd_to_XYZ_ASTME308()}, Practise *ASTM E308-15* working wavelengths range is [360, 780], if *True* this argument will trim the colour matching functions appropriately.
- **kwargs** (Any) –

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- When k is set to a value other than *None*, the computed *CIE XYZ* tristimulus values are assumed to be absolute and are thus converted from percentages by a final division by 100.
- The code path using the *ArrayLike* spectral distribution produces results different to the code path using a `colour.SpectralDistribution` class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the spectral distribution to the colour matching functions.

References

[ASTMInternational11], [ASTMInternational15a], [WS00c]

Examples

```
>>> import numpy as np
>>> from colour import (
...     MSDS_CMFS,
...     SDS_ILLUMINANTS,
...     SpectralDistribution,
...     SpectralShape,
... )
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> illuminant = SDS_ILLUMINANTS["D65"]
>>> shape = SpectralShape(400, 700, 20)
>>> data = np.array(
...     [
...         0.0641,
...         0.0645,
...         0.0562,
...         0.0537,
...         0.0559,
...         0.0651,
...         0.0705,
...         0.0772,
...         0.0870,
...         0.1128,
...         0.1360,
...         0.1511,
...         0.1688,
...         0.1996,
...         0.2397,
...         0.2852,
...     ]
... )
>>> sd = SpectralDistribution(data, shape)
>>> sd_to_XYZ(sd, cmfs, illuminant)
...
array([ 10.8401953...,   9.6841740...,   6.2158913...])
>>> sd_to_XYZ(sd, cmfs, illuminant, use_practice_range=False)
```

(continues on next page)

(continued from previous page)

```
...
array([ 10.8402774...,   9.6841967...,   6.2158838...])
>>> sd_to_XYZ(sd, cmfs, illuminant, method="Integration")
...
array([ 10.8404805...,   9.6838697...,   6.2115722...])
>>> sd_to_XYZ(data, cmfs, illuminant, method="Integration", shape=shape)
...
array([ 10.8993917...,   9.6986145...,   6.2540301...])
```

The default CMFS are the *CIE 1931 2 Degree Standard Observer*, and the default illuminant is *CIE Illuminant E*:

```
>>> sd_to_XYZ(sd)
...
array([ 11.7781589...,   9.9585580...,   5.7408602...])
```

colour.SD_TO_XYZ_METHODS

colour.**SD_TO_XYZ_METHODS = CanonicalMapping({'ASTM E308': ..., 'Integration': ..., 'astm2015': ...})**

Supported spectral distribution to *CIE XYZ* tristimulus values conversion methods.

References

[ASTMInternational11], [ASTMInternational15a], [WS00c]

Aliases:

- 'astm2015': 'ASTM E308'

colour.msds_to_XYZ

colour.**msds_to_XYZ**(*msds: ArrayLike* | SpectralDistribution | MultiSpectralDistributions, *cmfs:* MultiSpectralDistributions | *None* = *None*, *illuminant:* SpectralDistribution | *None* = *None*, *k: Real* | *None* = *None*, *method: Literal['ASTM E308', 'Integration']* | *str* = *'ASTM E308'*, ***kwargs:* Any) → NDArrayFloat

Convert given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant. For the *Integration* method, the multi-spectral distributions can be either a `colour.MultiSpectralDistributions` class instance or an *ArrayLike* in which case the shape must be passed.

Parameters

- **msds** (ArrayLike | SpectralDistribution | MultiSpectralDistributions) – Multi-spectral distributions, if an *ArrayLike* the wavelengths are expected to be in the last axis, e.g. for a 512x384 multi-spectral image with 77 bins, `msds` shape should be (384, 512, 77).
- **cmfs** (MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **k** (Real | None) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the

object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

- **method** (`Literal['ASTM E308', 'Integration'] | str`) – Computation method.
- **mi_5nm_omission_method** – {`colour.colorimetry.msds_to_XYZ_ASTME308()`}, 5 nm measurement intervals multi-spectral distributions conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** – {`colour.colorimetry.msds_to_XYZ_ASTME308()`}, 20 nm measurement intervals multi-spectral distributions conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **shape** – {`colour.colorimetry.msds_to_XYZ_integration()`}, Spectral shape of the multi-spectral distributions array $msds$, `cmfs` and `illuminant` will be aligned to it.
- **use_practice_range** – {`colour.colorimetry.msds_to_XYZ_ASTME308()`}, Practise *ASTM E308-15* working wavelengths range is [360, 780], if *True* this argument will trim the colour matching functions appropriately.
- **kwargs** (`Any`) –

Returns *CIE XYZ* tristimulus values, for a 512x384 multi-spectral image with 77 wavelengths, the output shape will be (384, 512, 3).

Return type `numpy.ndarray`

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- When k is set to a value other than *None*, the computed *CIE XYZ* tristimulus values are assumed to be absolute and are thus converted from percentages by a final division by 100.
- The code path using the *ArrayLike* multi-spectral distributions produces results different to the code path using a `colour.MultiSpectralDistributions` class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the multi-spectral distributions to the colour matching functions.
- If precision is required, it is possible to interpolate the multi-spectral distributions with `scipy.interpolate.interp1d` class on the last / tail axis as follows:

```
interpolator = scipy.interpolate.interp1d(
    wavelengths,
    values,
    axis=-1,
    kind="linear",
    fill_value="extrapolate",
)
values_i = interpolator(wavelengths_i)
```

References

[ASTMInternational11], [ASTMInternational15a], [WS00c]

Examples

```
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS, SpectralDistribution
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> illuminant = SDS_ILLUMINANTS["D65"]
>>> shape = SpectralShape(400, 700, 60)
>>> data = np.array(
...     [
...         [
...             0.0137,
...             0.0159,
...             0.0096,
...             0.0111,
...             0.0179,
...             0.1057,
...             0.0433,
...             0.0258,
...             0.0248,
...             0.0186,
...             0.0310,
...             0.0473,
...         ],
...         [
...             0.0913,
...             0.3145,
...             0.2582,
...             0.0709,
...             0.2971,
...             0.4620,
...             0.2683,
...             0.0831,
...             0.1203,
...             0.1292,
...             0.1682,
...             0.3221,
...         ],
...         [
...             0.0152,
...             0.0842,
...             0.4139,
...             0.0220,
...             0.5630,
...             0.1918,
...             0.2373,
...             0.0430,
...             0.0054,
...             0.0079,
...             0.3719,
...             0.2268,
...         ],
...         [
...             0.0281,
```

(continues on next page)

(continued from previous page)

```
...             0.0907,
...             0.2228,
...             0.1249,
...             0.2375,
...             0.5625,
...             0.0518,
...             0.3230,
...             0.0065,
...             0.4006,
...             0.0861,
...             0.3161,
...         ],
...         [
...             0.1918,
...             0.7103,
...             0.0041,
...             0.1817,
...             0.0024,
...             0.4209,
...             0.0118,
...             0.2302,
...             0.1860,
...             0.9404,
...             0.0041,
...             0.1124,
...         ],
...         [
...             0.0430,
...             0.0437,
...             0.3744,
...             0.0020,
...             0.5819,
...             0.0027,
...             0.0823,
...             0.0081,
...             0.3625,
...             0.3213,
...             0.7849,
...             0.0024,
...         ],
...     ]
... )
>>> msds = MultiSpectralDistributions(data, shape)
>>> msds_to_XYZ(msds, cmfs, illuminant, method="Integration")
...
array([[  7.5029704...,   3.9487844...,   8.4034669...],
       [ 26.9259681...,  15.0724609...,  28.7057807...],
       [ 16.7032188...,  28.2172346...,  25.6455984...],
       [ 11.5767013...,   8.6400993...,   6.5768406...],
       [ 18.7314793...,  35.0750364...,  30.1457266...],
       [ 45.1656756...,  39.6136917...,  43.6783499...],
       [  8.1755696...,  13.0934177...,  25.9420944...],
       [ 22.4676286...,  19.3099080...,   7.9637549...],
       [  6.5781241...,   2.5255349...,  11.0930768...],
       [ 43.9147364...,  27.9803924...,  11.7292655...],
       [  8.5365923...,  19.7030166...,  17.7050933...],
```

(continues on next page)

(continued from previous page)

```
       [ 23.9088250...,  26.2129529...,  30.6763148...]])
>>> data = np.reshape(data, (2, 6, 6))
>>> msds_to_XYZ(data, cmfs, illuminant, method="Integration", shape=shape)
...
array([[[  1.3104332...,   1.1377026...,   1.8267926...],
        [  2.1875548...,   2.2510619...,   3.0721540...],
        [ 16.8714661...,  17.7063715...,  35.8709902...],
        [ 12.1648722...,  12.7222194...,  10.4880888...],
        [ 16.0419431...,  23.0985768...,  11.1479902...],
        [  9.2391014...,   3.8301575...,   5.4703803...]],

       [[ 13.8734231...,  17.3942194...,  11.0364103...],
        [ 27.7096381...,  20.8626722...,  35.5581690...],
        [ 22.7886687...,  11.4769218...,  78.3300659...],
        [ 51.1284864...,  52.2463568...,  26.1483754...],
        [ 14.4749229...,  20.5011495...,   6.6228107...],
        [ 33.6001365...,  36.3242617...,   2.8254217...]]])
```

The default CMFS are the *CIE 1931 2 Degree Standard Observer*, and the default illuminant is *CIE Illuminant E*:

```
>>> msds_to_XYZ(msds, method="Integration")
...
array([[  8.2415862...,   4.2543993...,   7.6100842...],
       [ 29.6144619...,  16.1158465...,  25.9015472...],
       [ 16.6799560...,  27.2350547...,  22.9413337...],
       [ 12.5597688...,   9.0667136...,   5.9670327...],
       [ 18.5804689...,  33.6618109...,  26.9249733...],
       [ 47.7113308...,  40.4573249...,  39.6439145...],
       [  7.830207 ...,  12.3689624...,  23.3742655...],
       [ 24.1695370...,  20.0629815...,   7.2718670...],
       [  7.2333751...,   2.7982097...,  10.0688374...],
       [ 48.7358074...,  30.2417164...,  10.6753233...],
       [  8.3231013...,  18.6791507...,  15.8228184...],
       [ 24.6452277...,  26.0809382...,  27.7106399...]])
```

colour.MSDS_TO_XYZ_METHODS

colour.**MSDS_TO_XYZ_METHODS = CanonicalMapping({'ASTM E308': ..., 'Integration': ..., 'astm2015': ...})**

Supported multi-spectral array to *CIE XYZ* tristimulus values conversion methods.

References

[ASTMInternational11], [ASTMInternational15a], [WS00c]

Aliases:

- 'astm2015': 'ASTM E308'

colour.wavelength_to_XYZ

colour.**wavelength_to_XYZ**(*wavelength: ArrayLike*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*) → NDArrayFloat

Convert 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** (ArrayLike) – Wavelength λ in nm.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Raises **ValueError** – If wavelength λ is not contained in the colour matching functions domain.

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> from colour import MSDS_CMFS
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> wavelength_to_XYZ(480, cmfs)
array([ 0.09564   ,  0.13902   ,  0.8129501...])
>>> wavelength_to_XYZ(480.5, cmfs)
array([ 0.0914287...,  0.1418350...,  0.7915726...])
```

Ancillary Objects

colour.colorimetry

handle_spectral_arguments([cmfs, ...])	Handle the spectral arguments of various *Colour* definitions performing spectral computations.

colour.colorimetry.handle_spectral_arguments

colour.colorimetry.**handle_spectral_arguments**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None, illuminant: colour.colorimetry.spectrum.SpectralDistribution | None = None, cmfs_default: str = 'CIE 1931 2 Degree Standard Observer', illuminant_default: str = 'D65', shape_default: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT, issue_runtime_warnings: bool = True*) → Tuple[*colour.colorimetry.spectrum.MultiSpectralDistributions, colour.colorimetry.spectrum.SpectralDistribution*]

Handle the spectral arguments of various *Colour* definitions performing spectral computations.

- If `cmfs` is not given, one is chosen according to `cmfs_default`. The returned colour matching functions adopt the spectral shape given by `shape_default`.
- If `illuminant` is not given, one is chosen according to `illuminant_default`. The returned illuminant adopts the spectral shape of the returned colour matching functions.
- If `illuminant` is given, the returned illuminant spectral shape is aligned to that of the returned colour matching functions.

Parameters

- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | None`) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (`colour.colorimetry.spectrum.SpectralDistribution | None`) – Illuminant spectral distribution, default to *CIE Standard Illuminant D65*.
- **cmfs_default** (`str`) – The default colour matching functions to use if `cmfs` is not given.
- **illuminant_default** (`str`) – The default illuminant to use if `illuminant` is not given.
- **shape_default** (`colour.colorimetry.spectrum.SpectralShape`) – The default spectral shape to align the final colour matching functions and illuminant.
- **issue_runtime_warnings** (`bool`) – Whether to issue the runtime warnings.

Returns Colour matching functions and illuminant.

Return type `tuple`

Examples

```
>>> cmfs, illuminant = handle_spectral_arguments()
>>> cmfs.name, cmfs.shape, illuminant.name, illuminant.shape
('CIE 1931 2 Degree Standard Observer', SpectralShape(360.0, 780.0, 1.0), 'D65',␣
↪SpectralShape(360.0, 780.0, 1.0))
>>> cmfs, illuminant = handle_spectral_arguments(
...     shape_default=SpectralShape(400, 700, 20)
... )
>>> cmfs.name, cmfs.shape, illuminant.name, illuminant.shape
('CIE 1931 2 Degree Standard Observer', SpectralShape(400.0, 700.0, 20.0), 'D65',␣
↪SpectralShape(400.0, 700.0, 20.0))
```

ASTM E308-15

colour.colorimetry

sd_to_XYZ_ASTME308(sd[, cmfs, illuminant, ...])	Convert given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.
msds_to_XYZ_ASTME308(msds[, cmfs, ...])	Convert given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

colour.colorimetry.sd_to_XYZ_ASTME308

colour.colorimetry.**sd_to_XYZ_ASTME308**(*sd: SpectralDistribution, cmfs: MultiSpectralDistributions | None = None, illuminant: SpectralDistribution | None = None, use_practice_range: bool = True, mi_5nm_omission_method: bool = True, mi_20nm_interpolation_method: bool = True, k: Real | None = None*) → NDArrayFloat

Convert given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

Parameters

- **sd** (SpectralDistribution) – Spectral distribution.
- **cmfs** (MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **use_practice_range** (bool) – 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) – 5 nm measurement intervals spectral distribution conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** (bool) – 20 nm measurement intervals spectral distribution conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **k** (Real | None) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- When k is set to a value other than *None*, the computed *CIE XYZ* tristimulus values are assumed to be absolute and are thus converted from percentages by a final division by 100.

References

[ASTMInternational15a]

Examples

```
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS, SpectralDistribution
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> illuminant = SDS_ILLUMINANTS["D65"]
>>> shape = SpectralShape(400, 700, 20)
>>> data = np.array(
...     [
...         0.0641,
...         0.0645,
...         0.0562,
...         0.0537,
...         0.0559,
...         0.0651,
...         0.0705,
...         0.0772,
...         0.0870,
...         0.1128,
...         0.1360,
...         0.1511,
...         0.1688,
...         0.1996,
...         0.2397,
...         0.2852,
...     ]
... )
>>> sd = SpectralDistribution(data, shape)
>>> sd_to_XYZ_ASTME308(sd, cmfs, illuminant)
...
array([ 10.8401953...,   9.6841740...,   6.2158913...])
```

The default CMFS are the *CIE 1931 2 Degree Standard Observer*, and the default illuminant is *CIE Illuminant E*:

```
>>> sd_to_XYZ_ASTME308(sd)
...
array([ 11.7781589...,   9.9585580...,   5.7408602...])
```

colour.colorimetry.msds_to_XYZ_ASTME308

colour.colorimetry.**msds_to_XYZ_ASTME308**(*msds: MultiSpectralDistributions*, *cmfs: MultiSpectralDistributions | None = None*, *illuminant: SpectralDistribution | None = None*, *k: Real | None = None*, *use_practice_range: bool = True*, *mi_5nm_omission_method: bool = True*, *mi_20nm_interpolation_method: bool = True*) → NDArrayFloat

Convert given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

Parameters

- **msds** (MultiSpectralDistributions) – Multi-spectral distributions.
- **cmfs** (MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **k** (Real | None) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.
- **use_practice_range** (bool) – 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) – 5 nm measurement intervals multi-spectral distributions conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** (bool) – 20 nm measurement intervals multi-spectral distributions conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- When k is set to a value other than *None*, the computed *CIE XYZ* tristimulus values are assumed to be absolute and are thus converted from percentages by a final division by 100.

References

[WS00c]

Examples

```
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> illuminant = SDS_ILLUMINANTS["D65"]
>>> shape = SpectralShape(400, 700, 60)
>>> data = np.array(
...     [
...         [
...             0.0137,
...             0.0159,
...             0.0096,
...             0.0111,
...             0.0179,
...             0.1057,
...             0.0433,
...             0.0258,
...             0.0248,
...             0.0186,
...             0.0310,
...             0.0473,
...         ],
...         [
...             0.0913,
...             0.3145,
...             0.2582,
...             0.0709,
...             0.2971,
...             0.4620,
...             0.2683,
...             0.0831,
...             0.1203,
...             0.1292,
...             0.1682,
...             0.3221,
...         ],
...         [
...             0.0152,
...             0.0842,
...             0.4139,
...             0.0220,
...             0.5630,
...             0.1918,
...             0.2373,
...             0.0430,
...             0.0054,
...             0.0079,
...             0.3719,
...             0.2268,
...         ],
...         [
...             0.0281,
```

(continues on next page)

(continued from previous page)

```
...             0.0907,
...             0.2228,
...             0.1249,
...             0.2375,
...             0.5625,
...             0.0518,
...             0.3230,
...             0.0065,
...             0.4006,
...             0.0861,
...             0.3161,
...         ],
...         [
...             0.1918,
...             0.7103,
...             0.0041,
...             0.1817,
...             0.0024,
...             0.4209,
...             0.0118,
...             0.2302,
...             0.1860,
...             0.9404,
...             0.0041,
...             0.1124,
...         ],
...         [
...             0.0430,
...             0.0437,
...             0.3744,
...             0.0020,
...             0.5819,
...             0.0027,
...             0.0823,
...             0.0081,
...             0.3625,
...             0.3213,
...             0.7849,
...             0.0024,
...         ],
...     ]
... )
>>> msds = MultiSpectralDistributions(data, shape)
>>> msds = msds.align(SpectralShape(400, 700, 20))
>>> msds_to_XYZ_ASTME308(msds, cmfs, illuminant)
...
array([[  7.5052758...,   3.9557516...,   8.38929  ...],
       [ 26.9408494...,  15.0987746...,  28.6631260...],
       [ 16.7047370...,  28.2089815...,  25.6556751...],
       [ 11.5711808...,   8.6445071...,   6.5587827...],
       [ 18.7428858...,  35.0626352...,  30.1778517...],
       [ 45.1224886...,  39.6238997...,  43.5813345...],
       [  8.1786985...,  13.0950215...,  25.9326459...],
       [ 22.4462888...,  19.3115133...,   7.9304333...],
       [  6.5764361...,   2.5305945...,  11.07253  ...],
       [ 43.9113380...,  28.0003541...,  11.6852531...],
```

(continues on next page)

(continued from previous page)

```
       [  8.5496209...,  19.6913570...,  17.7400079...],
       [ 23.8866733...,  26.2147704...,  30.6297684...]])
```

The default CMFS are the *CIE 1931 2 Degree Standard Observer*, and the default illuminant is *CIE Illuminant E*:

```
>>> msds_to_XYZ_ASTME308(msds)
...
array([[  8.2439318...,   4.2617641...,   7.5977409...],
       [ 29.6290771...,  16.1443076...,  25.8640484...],
       [ 16.6819067...,  27.2271403...,  22.9490590...],
       [ 12.5543694...,   9.0705685...,   5.9516323...],
       [ 18.5921357...,  33.6508573...,  26.9511144...],
       [ 47.6698072...,  40.4630866...,  39.5612904...],
       [  7.8336896...,  12.3711768...,  23.3654245...],
       [ 24.1486630...,  20.0621956...,   7.2438655...],
       [  7.2323703...,   2.8033217...,  10.0510790...],
       [ 48.7322793...,  30.2614779...,  10.6377135...],
       [  8.3365770...,  18.6690888...,  15.8517212...],
       [ 24.6240657...,  26.0805317...,  27.6706915...]])
```

Ancillary Objects

`colour.colorimetry`

`sd_to_XYZ_tristimulus_weighting_factors_ASTME308(sd)`	Convert given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant using a table of tristimulus weighting factors according to practise *ASTM E308-15* method.
`adjust_tristimulus_weighting_factors_ASTME308(W, ...)`	Adjust given table of tristimulus weighting factors to account for a shorter wavelengths range of the test spectral shape compared to the reference spectral shape using practise *ASTM E308-15* method: Weights at the wavelengths for which data are not available are added to the weights at the shortest and longest wavelength for which spectral data are available.
`lagrange_coefficients_ASTME2022([interval, ...])`	Compute the *Lagrange Coefficients* for given interval size using practise *ASTM E2022-11* method.
`tristimulus_weighting_factors_ASTME2022(...)`	Return a table of tristimulus weighting factors for given colour matching functions and illuminant using practise *ASTM E2022-11* method.

colour.colorimetry.sd_to_XYZ_tristimulus_weighting_factors_ASTME308

colour.colorimetry.**sd_to_XYZ_tristimulus_weighting_factors_ASTME308**(*sd: SpectralDistribution, cmfs: MultiSpectralDistributions | None = None, illuminant: SpectralDistribution | None = None, k: Real | None = None*) → NDArrayFloat

Convert given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant using a table of tristimulus weighting factors according to practise *ASTM E308-15* method.

Parameters

- **sd** (`SpectralDistribution`) – Spectral distribution.
- **cmfs** (`MultiSpectralDistributions | None`) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (`SpectralDistribution | None`) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **k** (`Real | None`) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns *CIE XYZ* tristimulus values.

Return type `numpy.ndarray`

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[ASTMInternational15a]

Examples

```
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS, SpectralDistribution
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> illuminant = SDS_ILLUMINANTS["D65"]
>>> shape = SpectralShape(400, 700, 20)
>>> data = np.array(
...     [
...         0.0641,
...         0.0645,
...         0.0562,
...         0.0537,
...         0.0559,
...         0.0651,
...         0.0705,
...         0.0772,
...         0.0870,
...         0.1128,
...         0.1360,
```

(continues on next page)

(continued from previous page)

```
...         0.1511,
...         0.1688,
...         0.1996,
...         0.2397,
...         0.2852,
...     ]
... )
>>> sd = SpectralDistribution(data, shape)
>>> sd_to_XYZ_tristimulus_weighting_factors_ASTME308(
...     sd, cmfs, illuminant
... )
array([ 10.8405832...,   9.6844909...,   6.2155622...])
```

The default CMFS are the *CIE 1931 2 Degree Standard Observer*, and the default illuminant is *CIE Illuminant E*:

```
>>> sd_to_XYZ_tristimulus_weighting_factors_ASTME308(sd)
...
array([ 11.7786111...,   9.9589055...,   5.7403205...])
```

colour.colorimetry.adjust_tristimulus_weighting_factors_ASTME308

colour.colorimetry.**adjust_tristimulus_weighting_factors_ASTME308**(*W: ArrayLike*, *shape_r: colour.colorimetry.spectrum.SpectralShape*, *shape_t: colour.colorimetry.spectrum.SpectralShape*) → NDArrayFloat

Adjust 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** (ArrayLike) – Tristimulus weighting factors table.
- **shape_r** (colour.colorimetry.spectrum.SpectralShape) – Reference spectral shape.
- **shape_t** (colour.colorimetry.spectrum.SpectralShape) – Test spectral shape.

Returns Adjusted tristimulus weighting factors.

Return type numpy.ndarray

References

[ASTMInternational15a]

Examples

```
>>> from colour import (
...     MSDS_CMFS,
...     SpectralDistribution,
...     SpectralShape,
...     sd_CIE_standard_illuminant_A,
... )
>>> from colour.utilities import numpy_print_options
>>> cmfs = MSDS_CMFS["CIE 1964 10 Degree Standard Observer"]
>>> A = sd_CIE_standard_illuminant_A(cmfs.shape)
>>> W = tristimulus_weighting_factors_ASTME2022(
...     cmfs, A, SpectralShape(360, 830, 20)
... )
>>> with numpy_print_options(suppress=True):
...     adjust_tristimulus_weighting_factors_ASTME308(
...         W, SpectralShape(360, 830, 20), SpectralShape(400, 700, 20)
...     )
...
...
array([[  0.0509543...,   0.0040971...,   0.2144280...],
       [  0.7734225...,   0.0779839...,   3.6965732...],
       [  1.9000905...,   0.3037005...,   9.7554195...],
       [  1.9707727...,   0.8552809...,  11.4867325...],
       [  0.7183623...,   2.1457000...,   6.7845806...],
       [  0.0426667...,   4.8985328...,   2.3208000...],
       [  1.5223302...,   9.6471138...,   0.7430671...],
       [  5.6770329...,  14.4609708...,   0.1958194...],
       [ 12.4451744...,  17.4742541...,   0.0051827...],
       [ 20.5535772...,  17.5838219...,  -0.0026512...],
       [ 25.3315384...,  14.8957035...,   0.        ...],
       [ 21.5711570...,  10.0796619...,   0.        ...],
       [ 12.1785817...,   5.0680655...,   0.        ...],
       [  4.6675746...,   1.8303239...,   0.        ...],
       [  1.3236117...,   0.5129694...,   0.        ...],
       [  0.4171109...,   0.1618194...,   0.        ...]])
```

colour.colorimetry.lagrange_coefficients_ASTME2022

colour.colorimetry.**lagrange_coefficients_ASTME2022**(*interval: int = 10, interval_type: Union[Literal['Boundary', 'Inner'], str] = 'Inner'*) → NDArrayFloat

Compute the *Lagrange Coefficients* for given interval size using practise *ASTM E2022-11* method.

Parameters

- **interval** (int) – Interval size in nm.
- **interval_type** (Union[Literal['Boundary', 'Inner'], str]) – 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 numpy.ndarray

References

[ASTMInternational11]

Examples

```
>>> lagrange_coefficients_ASTME2022(10, "inner")
...
array([[-0.028...,  0.940...,  0.104..., -0.016...],
       [-0.048...,  0.864...,  0.216..., -0.032...],
       [-0.059...,  0.773...,  0.331..., -0.045...],
       [-0.064...,  0.672...,  0.448..., -0.056...],
       [-0.062...,  0.562...,  0.562..., -0.062...],
       [-0.056...,  0.448...,  0.672..., -0.064...],
       [-0.045...,  0.331...,  0.773..., -0.059...],
       [-0.032...,  0.216...,  0.864..., -0.048...],
       [-0.016...,  0.104...,  0.940..., -0.028...]])
>>> lagrange_coefficients_ASTME2022(10, "boundary")
...
array([[ 0.85...,  0.19..., -0.04...],
       [ 0.72...,  0.36..., -0.08...],
       [ 0.59...,  0.51..., -0.10...],
       [ 0.48...,  0.64..., -0.12...],
       [ 0.37...,  0.75..., -0.12...],
       [ 0.28...,  0.84..., -0.12...],
       [ 0.19...,  0.91..., -0.10...],
       [ 0.12...,  0.96..., -0.08...],
       [ 0.05...,  0.99..., -0.04...]])
```

colour.colorimetry.tristimulus_weighting_factors_ASTME2022

colour.colorimetry.**tristimulus_weighting_factors_ASTME2022**(*cmfs: MultiSpectralDistributions, illuminant: SpectralDistribution, shape: SpectralShape, k: Real | None = None*) → NDArrayFloat

Return 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** (MultiSpectralDistributions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.
- **shape** (SpectralShape) – Shape used to build the table, only the interval is needed.
- **k** (Real | None) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required

to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns Tristimulus weighting factors table.

Return type `numpy.ndarray`

Raises `ValueError` – If the colour matching functions or illuminant intervals are not equal to 1 nm.

Notes

- Input colour matching functions and illuminant intervals are expected to be equal to 1 nm. If the illuminant data is not available at 1 nm interval, it needs to be interpolated using *CIE* recommendations: The method developed by *Sprague (1880)* should be used for interpolating functions having a uniformly spaced independent variable and a *Cubic Spline* method for non-uniformly spaced independent variable.

References

[ASTMInternational11]

Examples

```
>>> from colour import (
...     MSDS_CMFS,
...     SpectralDistribution,
...     SpectralShape,
...     sd_CIE_standard_illuminant_A,
... )
>>> from colour.utilities import numpy_print_options
>>> cmfs = MSDS_CMFS["CIE 1964 10 Degree Standard Observer"]
>>> A = sd_CIE_standard_illuminant_A(cmfs.shape)
>>> with numpy_print_options(suppress=True):
...     tristimulus_weighting_factors_ASTME2022(
...         cmfs, A, SpectralShape(360, 830, 20)
...     )
...
...
array([[ -0.0002981...,  -0.0000317...,  -0.0013301...],
       [ -0.0087155...,  -0.0008915...,  -0.0407436...],
       [  0.0599679...,   0.0050203...,   0.2565018...],
       [  0.7734225...,   0.0779839...,   3.6965732...],
       [  1.9000905...,   0.3037005...,   9.7554195...],
       [  1.9707727...,   0.8552809...,  11.4867325...],
       [  0.7183623...,   2.1457000...,   6.7845806...],
       [  0.0426667...,   4.8985328...,   2.3208000...],
       [  1.5223302...,   9.6471138...,   0.7430671...],
       [  5.6770329...,  14.4609708...,   0.1958194...],
       [ 12.4451744...,  17.4742541...,   0.0051827...],
       [ 20.5535772...,  17.5838219...,  -0.0026512...],
       [ 25.3315384...,  14.8957035...,   0.        ...],
       [ 21.5711570...,  10.0796619...,   0.        ...],
```

(continues on next page)

(continued from previous page)

```
        [ 12.1785817...,   5.0680655...,   0.        ...],
        [  4.6675746...,   1.8303239...,   0.        ...],
        [  1.3236117...,   0.5129694...,   0.        ...],
        [  0.3175325...,   0.1230084...,   0.        ...],
        [  0.0746341...,   0.0290243...,   0.        ...],
        [  0.0182990...,   0.0071606...,   0.        ...],
        [  0.0047942...,   0.0018888...,   0.        ...],
        [  0.0013293...,   0.0005277...,   0.        ...],
        [  0.0004254...,   0.0001704...,   0.        ...],
        [  0.0000962...,   0.0000389...,   0.        ...]])
```

Integration

`colour.colorimetry`

`sd_to_XYZ_integration`(sd[, cmfs, ...])	Convert given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to classical integration method.
`msds_to_XYZ_integration`(msds[, cmfs, ...])	Convert given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant.

colour.colorimetry.sd_to_XYZ_integration

colour.colorimetry.**sd_to_XYZ_integration**(*sd: ArrayLike* | SpectralDistribution | MultiSpectralDistributions, *cmfs:* MultiSpectralDistributions | *None* = *None*, *illuminant:* SpectralDistribution | *None* = *None*, *k: Real* | *None* = *None*, *shape:* SpectralShape | *None* = *None*) → NDArrayFloat

Convert given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to classical integration method.

The spectral distribution can be either a `colour.SpectralDistribution` class instance or an *ArrayLike* in which case the `shape` must be passed.

Parameters

- **sd** (ArrayLike | SpectralDistribution | MultiSpectralDistributions) – Spectral distribution, if an *ArrayLike* the wavelengths are expected to be in the last axis, e.g. for a spectral array with 77 bins, `sd` shape could be (77,) or (1, 77).
- **cmfs** (MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **k** (Real | None) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required

to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

- **shape** (`SpectralShape` | `None`) – Spectral shape of the spectral distribution, `cmfs` and `illuminant` will be aligned to it if `sd` is an *ArrayLike*.

Returns *CIE XYZ* tristimulus values.

Return type `numpy.ndarray`

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- When k is set to a value other than *None*, the computed *CIE XYZ* tristimulus values are assumed to be absolute and are thus converted from percentages by a final division by 100.
- The code path using the *ArrayLike* spectral distribution produces results different to the code path using a `colour.SpectralDistribution` class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the spectral distribution to the colour matching functions.

References

[WS00c]

Examples

```
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS, SpectralDistribution
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> illuminant = SDS_ILLUMINANTS["D65"]
>>> shape = SpectralShape(400, 700, 20)
>>> data = np.array(
...     [
...         0.0641,
...         0.0645,
...         0.0562,
...         0.0537,
...         0.0559,
...         0.0651,
...         0.0705,
...         0.0772,
...         0.0870,
...         0.1128,
...         0.1360,
...         0.1511,
...         0.1688,
...         0.1996,
...         0.2397,
...         0.2852,
...     ]
```

(continues on next page)

(continued from previous page)

```
... )
>>> sd = SpectralDistribution(data, shape)
>>> sd_to_XYZ_integration(sd, cmfs, illuminant)
...
array([ 10.8404805...,   9.6838697...,   6.2115722...])
>>> sd_to_XYZ_integration(data, cmfs, illuminant, shape=shape)
...
array([ 10.8993917...,   9.6986145...,   6.2540301...])
```

The default CMFS are the *CIE 1931 2 Degree Standard Observer*, and the default illuminant is *CIE Illuminant E*:

```
>>> sd_to_XYZ_integration(sd)
...
array([ 11.7786939...,   9.9583972...,   5.7371816...])
```

colour.colorimetry.msds_to_XYZ_integration

colour.colorimetry.**msds_to_XYZ_integration**(*msds: ArrayLike* | SpectralDistribution | MultiSpectralDistributions, *cmfs:* MultiSpectralDistributions | *None* = *None*, *illuminant:* SpectralDistribution | *None* = *None*, *k: Real* | *None* = *None*, *shape:* SpectralShape | *None* = *None*) → NDArrayFloat

Convert given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant.

The multi-spectral distributions can be either a `colour.MultiSpectralDistributions` class instance or an *ArrayLike* in which case the `shape` must be passed.

Parameters

- **msds** (`ArrayLike | SpectralDistribution | MultiSpectralDistributions`) – Multi-spectral distributions, if an *ArrayLike* the wavelengths are expected to be in the last axis, e.g. for a 512x384 multi-spectral image with 77 bins, `msds` shape should be (384, 512, 77).
- **cmfs** (`MultiSpectralDistributions | None`) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (`SpectralDistribution | None`) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **k** (`Real | None`) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.
- **shape** (`SpectralShape | None`) – Spectral shape of the multi-spectral distributions, `cmfs` and `illuminant` will be aligned to it if `msds` is an *ArrayLike*.

Returns *CIE XYZ* tristimulus values, for a 512x384 multi-spectral image with 77 bins, the output shape will be (384, 512, 3).

Return type numpy.ndarray

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- When k is set to a value other than *None*, the computed *CIE XYZ* tristimulus values are assumed to be absolute and are thus converted from percentages by a final division by 100.
- The code path using the *ArrayLike* multi-spectral distributions produces results different to the code path using a colour.MultiSpectralDistributions class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the multi-spectral distributions to the colour matching functions.
- If precision is required, it is possible to interpolate the multi-spectral distributions with scipy.interpolate.interp1d class on the last / tail axis as follows:

```
interpolator = scipy.interpolate.interp1d(
    wavelengths,
    values,
    axis=-1,
    kind="linear",
    fill_value="extrapolate",
)
values_i = interpolator(wavelengths_i)
```

References

[WS00c]

Examples

```
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> illuminant = SDS_ILLUMINANTS["D65"]
>>> shape = SpectralShape(400, 700, 60)
>>> data = np.array(
...     [
...         [
...             0.0137,
...             0.0159,
...             0.0096,
...             0.0111,
...             0.0179,
...             0.1057,
...             0.0433,
...             0.0258,
...             0.0248,
...             0.0186,
...             0.0310,
```

(continues on next page)

(continued from previous page)

```
...             0.0473,
...         ],
...         [
...             0.0913,
...             0.3145,
...             0.2582,
...             0.0709,
...             0.2971,
...             0.4620,
...             0.2683,
...             0.0831,
...             0.1203,
...             0.1292,
...             0.1682,
...             0.3221,
...         ],
...         [
...             0.0152,
...             0.0842,
...             0.4139,
...             0.0220,
...             0.5630,
...             0.1918,
...             0.2373,
...             0.0430,
...             0.0054,
...             0.0079,
...             0.3719,
...             0.2268,
...         ],
...         [
...             0.0281,
...             0.0907,
...             0.2228,
...             0.1249,
...             0.2375,
...             0.5625,
...             0.0518,
...             0.3230,
...             0.0065,
...             0.4006,
...             0.0861,
...             0.3161,
...         ],
...         [
...             0.1918,
...             0.7103,
...             0.0041,
...             0.1817,
...             0.0024,
...             0.4209,
...             0.0118,
...             0.2302,
...             0.1860,
...             0.9404,
...             0.0041,
```

(continues on next page)

(continued from previous page)

```
...             0.1124,
...         ],
...         [
...             0.0430,
...             0.0437,
...             0.3744,
...             0.0020,
...             0.5819,
...             0.0027,
...             0.0823,
...             0.0081,
...             0.3625,
...             0.3213,
...             0.7849,
...             0.0024,
...         ],
...     ]
... )
>>> msds = MultiSpectralDistributions(data, shape)
>>> msds_to_XYZ_integration(msds, cmfs, illuminant)
...
array([[  7.5029704...,   3.9487844...,   8.4034669...],
       [ 26.9259681...,  15.0724609...,  28.7057807...],
       [ 16.7032188...,  28.2172346...,  25.6455984...],
       [ 11.5767013...,   8.6400993...,   6.5768406...],
       [ 18.7314793...,  35.0750364...,  30.1457266...],
       [ 45.1656756...,  39.6136917...,  43.6783499...],
       [  8.1755696...,  13.0934177...,  25.9420944...],
       [ 22.4676286...,  19.3099080...,   7.9637549...],
       [  6.5781241...,   2.5255349...,  11.0930768...],
       [ 43.9147364...,  27.9803924...,  11.7292655...],
       [  8.5365923...,  19.7030166...,  17.7050933...],
       [ 23.9088250...,  26.2129529...,  30.6763148...]])
>>> data = np.reshape(data, (2, 6, 6))
>>> msds_to_XYZ_integration(data, cmfs, illuminant, shape=shape)
...
array([[[  1.3104332...,   1.1377026...,   1.8267926...],
        [  2.1875548...,   2.2510619...,   3.0721540...],
        [ 16.8714661...,  17.7063715...,  35.8709902...],
        [ 12.1648722...,  12.7222194...,  10.4880888...],
        [ 16.0419431...,  23.0985768...,  11.1479902...],
        [  9.2391014...,   3.8301575...,   5.4703803...]],

       [[ 13.8734231...,  17.3942194...,  11.0364103...],
        [ 27.7096381...,  20.8626722...,  35.5581690...],
        [ 22.7886687...,  11.4769218...,  78.3300659...],
        [ 51.1284864...,  52.2463568...,  26.1483754...],
        [ 14.4749229...,  20.5011495...,   6.6228107...],
        [ 33.6001365...,  36.3242617...,   2.8254217...]]])
```

The default CMFS are the *CIE 1931 2 Degree Standard Observer*, and the default illuminant is *CIE Illuminant E*:

```
>>> msds_to_XYZ_integration(msds)
...
array([[  8.2415862...,   4.2543993...,   7.6100842...],
```

(continues on next page)

(continued from previous page)

```
       [ 29.6144619...,  16.1158465...,  25.9015472...],
       [ 16.6799560...,  27.2350547...,  22.9413337...],
       [ 12.5597688...,   9.0667136...,   5.9670327...],
       [ 18.5804689...,  33.6618109...,  26.9249733...],
       [ 47.7113308...,  40.4573249...,  39.6439145...],
       [  7.830207 ...,  12.3689624...,  23.3742655...],
       [ 24.1695370...,  20.0629815...,   7.2718670...],
       [  7.2333751...,   2.7982097...,  10.0688374...],
       [ 48.7358074...,  30.2417164...,  10.6753233...],
       [  8.3231013...,  18.6791507...,  15.8228184...],
       [ 24.6452277...,  26.0809382...,  27.7106399...]])
```

Spectral Bandpass Dependence Correction

colour

bandpass_correction(sd[, method])	Implement spectral bandpass dependence correction on given spectral distribution using given method.
BANDPASS_CORRECTION_METHODS	Supported spectral bandpass dependence correction methods.

colour.bandpass_correction

colour.**bandpass_correction**(*sd: colour.colorimetry.spectrum.SpectralDistribution, method: Union[Literal['Stearns 1988'], str] = 'Stearns 1988'*) → *colour.colorimetry.spectrum.SpectralDistribution*

Implement spectral bandpass dependence correction on given spectral distribution using given method.

Parameters

- **sd** (colour.colorimetry.spectrum.SpectralDistribution) – Spectral distribution.
- **method** (Union[Literal['Stearns 1988'], str]) – Correction method.

Returns Spectral bandpass dependence corrected spectral distribution.

Return type colour.SpectralDistribution

References

[SS88], [WRC12a]

Examples

```
>>> from colour import SpectralDistribution
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> with numpy_print_options(suppress=True):
...     bandpass_correction(SpectralDistribution(data))
...
...
SpectralDistribution([[ 500.        ,    0.0646518...],
                      [ 520.        ,    0.0704293...],
                      [ 540.        ,    0.0769485...],
                      [ 560.        ,    0.0856928...],
                      [ 580.        ,    0.1129644...],
                      [ 600.        ,    0.1379256...]],
                     SpragueInterpolator,
                     {},
                     Extrapolator,
                     {'method': 'Constant', 'left': None, 'right': None})
```

colour.BANDPASS_CORRECTION_METHODS

`colour.`**`BANDPASS_CORRECTION_METHODS = CanonicalMapping({'Stearns 1988': ...})`**

Supported spectral bandpass dependence correction methods.

Stearns and Stearns (1988)

`colour.colorimetry`

`bandpass_correction_Stearns1988`(sd)	Implement spectral bandpass dependence correction on given spectral distribution using *Stearns and Stearns (1988)* method.

colour.colorimetry.bandpass_correction_Stearns1988

`colour.colorimetry.`**`bandpass_correction_Stearns1988`**(*sd: colour.colorimetry.spectrum.SpectralDistribution*) → *colour.colorimetry.spectrum.SpectralDistribution*

Implement spectral bandpass dependence correction on given spectral distribution using *Stearns and Stearns (1988)* method.

Parameters **sd** (`colour.colorimetry.spectrum.SpectralDistribution`) – Spectral distribution.

Returns Spectral bandpass dependence corrected spectral distribution.

Return type `colour.SpectralDistribution`

References

[SS88], [WRC12a]

Examples

```
>>> from colour import SpectralDistribution
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> with numpy_print_options(suppress=True):
...     bandpass_correction_Stearns1988(SpectralDistribution(data))
...
...
SpectralDistribution([[ 500.        ,    0.0646518...],
                      [ 520.        ,    0.0704293...],
                      [ 540.        ,    0.0769485...],
                      [ 560.        ,    0.0856928...],
                      [ 580.        ,    0.1129644...],
                      [ 600.        ,    0.1379256...]],
                     SpragueInterpolator,
                     {},
                     Extrapolator,
                     {'method': 'Constant', 'left': None, 'right': None})
```

Colour Matching Functions

`colour.colorimetry`

`LMS_ConeFundamentals`([data, domain, labels])	Implement support for the Stockman and Sharpe *LMS* cone fundamentals colour matching functions.
`RGB_ColourMatchingFunctions`([data, domain, ...])	Implement support for the *CIE RGB* colour matching functions.
`XYZ_ColourMatchingFunctions`([data, domain, ...])	Implement support for the *CIE* Standard Observers *XYZ* colour matching functions.

colour.colorimetry.LMS_ConeFundamentals

class colour.colorimetry.**LMS_ConeFundamentals**(*data: ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*)

Bases: `colour.colorimetry.spectrum.MultiSpectralDistributions`

Implement support for the Stockman and Sharpe *LMS* cone fundamentals colour matching functions.

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) – Data to be stored in the multi-spectral distributions.
- **domain** (ArrayLike | SpectralShape | None) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths property.
- **labels** (Sequence | None) – Names to use for the colour.SpectralDistribution class instances.
- **extrapolator** – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **name** – Multi-spectral distributions name.
- **display_labels** – Multi-spectral distributions labels for figures, default to colour.colorimetry.LMS_ConeFundamentals.labels property value.
- **kwargs** (Any) –

Return type None

__init__(*data: ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*) → None

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) –
- **domain** (ArrayLike | SpectralShape | None) –
- **labels** (Sequence | None) –
- **kwargs** (Any) –

Return type None

colour.colorimetry.RGB_ColourMatchingFunctions

class colour.colorimetry.**RGB_ColourMatchingFunctions**(*data: ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*)

Bases: `colour.colorimetry.spectrum.MultiSpectralDistributions`

Implement support for the *CIE RGB* colour matching functions.

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) – Data to be stored in the multi-spectral distributions.
- **domain** (ArrayLike | SpectralShape | None) – Values to initialise the multiple `colour.SpectralDistribution` class instances `colour.continuous.Signal.wavelengths` attribute with. If both `data` and `domain` arguments are defined, the latter will be used to initialise the `colour.continuous.Signal.wavelengths` property.
- **labels** (Sequence | None) – Names to use for the `colour.SpectralDistribution` class instances.
- **extrapolator** – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the `colour.SpectralDistribution` class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **name** – Multi-spectral distributions name.
- **display_labels** – Multi-spectral distributions labels for figures, default to `colour.colorimetry.RGB_ColourMatchingFunctions.labels` property value.
- **kwargs** (Any) –

Return type None

__init__(*data: ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*) → None

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) –
- **domain** (ArrayLike | SpectralShape | None) –
- **labels** (Sequence | None) –
- **kwargs** (Any) –

Return type None

colour.colorimetry.XYZ_ColourMatchingFunctions

class colour.colorimetry.**XYZ_ColourMatchingFunctions**(*data: ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*)

Bases: `colour.colorimetry.spectrum.MultiSpectralDistributions`

Implement support for the *CIE* Standard Observers *XYZ* colour matching functions.

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) – Data to be stored in the multi-spectral distributions.
- **domain** (ArrayLike | SpectralShape | None) – Values to initialise the multiple `colour.SpectralDistribution` class instances `colour.continuous.Signal.wavelengths` attribute with. If both `data` and `domain` arguments are defined, the latter will be used to initialise the `colour.continuous.Signal.wavelengths` property.
- **labels** (Sequence | None) – Names to use for the `colour.SpectralDistribution` class instances.
- **extrapolator** – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the `colour.SpectralDistribution` class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **name** – Multi-spectral distributions name.
- **display_labels** – Multi-spectral distributions labels for figures, default to `colour.colorimetry.XYZ_ColourMatchingFunctions.labels` property value.
- **kwargs** (Any) –

Return type None

__init__(*data: ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None = None, domain: ArrayLike | SpectralShape | None = None, labels: Sequence | None = None, **kwargs: Any*) → None

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | MultiSpectralDistributions | Sequence | Series | Signal | SpectralDistribution | None) –
- **domain** (ArrayLike | SpectralShape | None) –

- **labels** (Sequence | None) –
- **kwargs** (Any) –

Return type None

Dataset

colour

MSDS_CMFS	Multi-spectral distributions of the colour matching functions.

colour.MSDS_CMFS

colour.**MSDS_CMFS = LazyCanonicalMapping({'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman & Sharpe 10 Degree Cone Fundamentals': ..., 'Smith & Pokorny 1975 Normal Trichromats': ..., 'Wright & Guild 1931 2 Degree RGB CMFs': ..., 'Stiles & Burch 1955 2 Degree RGB CMFs': ..., 'Stiles & Burch 1959 10 Degree RGB CMFs': ..., 'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ..., 'CIE 2015 2 Degree Standard Observer': ..., 'CIE 2015 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ..., 'CIE 2012 2 Degree Standard Observer': ..., 'CIE 2012 10 Degree Standard Observer': ...})**

Multi-spectral distributions of the colour matching functions.

References

[Bro09], [CVRLb], [CVRLc], [CVRLd], [SS00], [CVRLe], [Mac10]

Ancillary Objects

colour.colorimetry

MSDS_CMFS_LMS	Multi-spectral distributions of the *LMS* colour matching functions.
MSDS_CMFS_RGB	Multi-spectral distributions of the *RGB* colour matching functions.
MSDS_CMFS_STANDARD_OBSERVER	Multi-spectral distributions of the *CIE* Standard Observer colour matching functions.

colour.colorimetry.MSDS_CMFS_LMS

colour.colorimetry.**MSDS_CMFS_LMS = LazyCanonicalMapping({'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman & Sharpe 10 Degree Cone Fundamentals': ..., 'Smith & Pokorny 1975 Normal Trichromats': ...})**

Multi-spectral distributions of the *LMS* colour matching functions.

References

[SS00], [Mac10]

colour.colorimetry.MSDS_CMFS_RGB

colour.colorimetry.**MSDS_CMFS_RGB = LazyCanonicalMapping({'Wright & Guild 1931 2 Degree RGB CMFs': ..., 'Stiles & Burch 1955 2 Degree RGB CMFs': ..., 'Stiles & Burch 1959 10 Degree RGB CMFs': ...})**

Multi-spectral distributions of the *RGB* colour matching functions.

References

[Bro09], [CVRLd], [CVRLe]

colour.colorimetry.MSDS_CMFS_STANDARD_OBSERVER

colour.colorimetry.**MSDS_CMFS_STANDARD_OBSERVER = LazyCanonicalMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ..., 'CIE 2015 2 Degree Standard Observer': ..., 'CIE 2015 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ..., 'CIE 2012 2 Degree Standard Observer': ..., 'CIE 2012 10 Degree Standard Observer': ...})**

Multi-spectral distributions of the *CIE* Standard Observer colour matching functions.

References

[CVRLb], [CVRLc]

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(...)	Convert *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(...)	Convert *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(...)	Convert *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(...)	Convert *Stockman & Sharpe 2 Degree Cone Fundamentals* colour matching functions into the *CIE 2015 2 Degree Standard Observer* colour matching functions.
LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs(...)	Convert *Stockman & Sharpe 10 Degree Cone Fundamentals* colour matching functions into the *CIE 2015 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: ArrayLike*) → NDArrayFloat

Convert *Wright & Guild 1931 2 Degree RGB CMFs* colour matching functions into the *CIE 1931 2 Degree Standard Observer* colour matching functions.

Parameters **wavelength** (ArrayLike) – Wavelength λ in nm.

Returns *CIE 1931 2 Degree Standard Observer* spectral tristimulus values.

Return type numpy.ndarray

Notes

- Data for the *CIE 1931 2 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[WS00f]

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: ArrayLike*) → NDArrayFloat

Convert *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *CIE 1964 10 Degree Standard Observer* colour matching functions.

Parameters **wavelength** (ArrayLike) – Wavelength λ in nm.

Returns *CIE 1964 10 Degree Standard Observer* spectral tristimulus values.

Return type numpy.ndarray

Notes

- Data for the *CIE 1964 10 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[WS00j]

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: ArrayLike*) → NDArrayFloat

Convert *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *Stockman & Sharpe 10 Degree Cone Fundamentals* spectral sensitivity functions.

Parameters **wavelength** (ArrayLike) – Wavelength λ in nm.

Returns *Stockman & Sharpe 10 Degree Cone Fundamentals* spectral tristimulus values.

Return type numpy.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: ArrayLike*) → NDArrayFloat

Convert *Stockman & Sharpe 2 Degree Cone Fundamentals* colour matching functions into the *CIE 2015 2 Degree Standard Observer* colour matching functions.

Parameters **wavelength** (ArrayLike) – Wavelength λ in nm.

Returns *CIE 2015 2 Degree Standard Observer* spectral tristimulus values.

Return type numpy.ndarray

Notes

- Data for the *CIE 2015 2 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[CVRLg]

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: ArrayLike*) → NDArrayFloat

Convert *Stockman & Sharpe 10 Degree Cone Fundamentals* colour matching functions into the *CIE 2015 10 Degree Standard Observer* colour matching functions.

Parameters **wavelength** (ArrayLike) – Wavelength λ in nm.

Returns *CIE 2015 10 Degree Standard Observer* spectral tristimulus values.

Return type numpy.ndarray

Notes

- Data for the *CIE 2015 10 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[CVRLf]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs(700)
...
array([ 0.0098162...,  0.0037761...,  0.        ])
```

Illuminants and Light Sources

Dataset

colour

CCS_ILLUMINANTS	Chromaticity coordinates of the illuminants.
SDS_ILLUMINANTS	Spectral distributions of the illuminants.
CCS_LIGHT_SOURCES	Chromaticity coordinates of the light sources.
SDS_LIGHT_SOURCES	Spectral distributions of the light sources.
TVS_ILLUMINANTS	*CIE XYZ* tristimulus values of the illuminants.
TVS_ILLUMINANTS_HUNTERLAB	*CIE XYZ* tristimulus values of the *HunterLab* illuminants.

colour.CCS_ILLUMINANTS

colour.**CCS_ILLUMINANTS = CanonicalMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})**

Chromaticity coordinates of the illuminants.

> **Warning:** *DCI-P3* illuminant has no associated spectral distribution. *DCI* has no official reference spectral measurement for this whitepoint. The closest matching spectral distribution is *Kinoton 75P* projector.

Notes

CIE Illuminant D Series D60 illuminant chromaticity coordinates were computed as follows:

```
CCT = 6000 * 1.4388 / 1.438
xy = colour.temperature.CCT_to_xy_CIE_D(CCT)

sd = colour.sd_CIE_illuminant_D_series(xy)
colour.XYZ_to_xy(
    colour.sd_to_XYZ(
        sd, colour.MSDS_CMFS["CIE 1964 10 Degree Standard Observer"]
    )
)
```

- *CIE Illuminant D Series D50* illuminant and *CIE Standard Illuminant D Series D65* chromaticity coordinates are rounded to 4 decimals as given in the typical RGB colourspaces litterature. Their chromaticity coordinates as given in [CIET14804f] are (0.34567, 0.35851) and (0.31272, 0.32903) respectively.
- *CIE* illuminants with chromaticity coordinates not defined in the reference [Wikipedia06b] have been calculated using their correlated colour temperature and `colour.temperature.CCT_to_xy_CIE_D()` `colour.sd_CIE_illuminant_D_series()` and / or `colour.sd_to_XYZ()` definitions.
- *ICC D50* chromaticity coordinates were computed with `colour.XYZ_to_xy()` definition from the *CIE XYZ* tristimulus values as given by *ICC*: [96.42, 100.00, 82.49].

References

[CIET14804f], [DigitalCInitiatives07], [InternationalOfStandardization02], [InternationalC-Consortium10], [PLASANAmerica15] [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESP-Subcommittee14b], [Wikipedia06b]

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.SDS_ILLUMINANTS

```
colour.SDS_ILLUMINANTS = LazyCanonicalMapping({'A': ..., 'B': ..., 'C': ..., 'D50': ..., 'D55': ..., 'D60': ..., 'D65': ..., 'D75': ..., 'E': ..., 'FL1': ..., 'FL2': ..., 'FL3': ..., 'FL4': ..., 'FL5': ..., 'FL6': ..., 'FL7': ..., 'FL8': ..., 'FL9': ..., 'FL10': ..., 'FL11': ..., 'FL12': ..., 'FL3.1': ..., 'FL3.2': ..., 'FL3.3': ..., 'FL3.4': ..., 'FL3.5': ..., 'FL3.6': ..., 'FL3.7': ..., 'FL3.8': ..., 'FL3.9': ..., 'FL3.10': ..., 'FL3.11': ..., 'FL3.12': ..., 'FL3.13': ..., 'FL3.14': ..., 'FL3.15': ..., 'HP1': ..., 'HP2': ..., 'HP3': ..., 'HP4': ..., 'HP5': ..., 'LED-B1': ..., 'LED-B2': ..., 'LED-B3': ..., 'LED-B4': ..., 'LED-B5': ..., 'LED-BH1': ..., 'LED-RGB1': ..., 'LED-V1': ..., 'LED-V2': ..., 'ID65': ..., 'ID50': ..., 'ISO 7589 Photographic Daylight': ..., 'ISO 7589 Sensitometric Daylight': ..., 'ISO 7589 Studio Tungsten': ..., 'ISO 7589 Sensitometric Studio Tungsten': ..., 'ISO 7589 Photoflood': ..., 'ISO 7589 Sensitometric Photoflood': ..., 'ISO 7589 Sensitometric Printer': ...})
```

Spectral distributions of the illuminants.

Notes

- *CIE 15:2004* recommends using linear interpolation for *CIE Standard Illuminant D Series*, for consistency all the illuminants are using a linear interpolator.

References

[CSH+18], [CIE04], [CIE], [InternationalOfStandardization02]

colour.CCS_LIGHT_SOURCES

```
colour.CCS_LIGHT_SOURCES = CanonicalMapping({'CIE 1931 2 Degree Standard Observer': ...,
'CIE 1964 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})
```

Chromaticity coordinates of the light sources.

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.SDS_LIGHT_SOURCES

```
colour.SDS_LIGHT_SOURCES = LazyCanonicalMapping({'Natural': ..., 'Philips TL-84': ...,
'SA': ..., 'SC': ..., 'T8 Luxline Plus White': ..., 'T8 Polylux 3000': ..., 'T8 Polylux
4000': ..., 'Thorn Kolor-rite': ..., 'Cool White FL': ..., 'Daylight FL': ..., 'HPS': ...,
'Incandescent': ..., 'LPS': ..., 'Mercury': ..., 'Metal Halide': ..., 'Neodimium
Incandescent': ..., 'Super HPS': ..., 'Triphosphor FL': ..., '3-LED-1 (457/540/605)': ...,
'3-LED-2 (473/545/616)': ..., '3-LED-2 Yellow': ..., '3-LED-3 (465/546/614)': ...,
'3-LED-4 (455/547/623)': ..., '4-LED No Yellow': ..., '4-LED Yellow': ..., '4-LED-1
(461/526/576/624)': ..., '4-LED-2 (447/512/573/627)': ..., 'Luxeon WW 2880': ...,
'PHOS-1': ..., 'PHOS-2': ..., 'PHOS-3': ..., 'PHOS-4': ..., 'Phosphor LED YAG': ..., '60
A/W (Soft White)': ..., 'C100S54 (HPS)': ..., 'C100S54C (HPS)': ..., 'F32T8/TL830
(Triphosphor)': ..., 'F32T8/TL835 (Triphosphor)': ..., 'F32T8/TL841 (Triphosphor)': ...,
'F32T8/TL850 (Triphosphor)': ..., 'F32T8/TL865/PLUS (Triphosphor)': ..., 'F34/CW/RS/EW
(Cool White FL)': ..., 'F34T12/LW/RS/EW': ..., 'F34T12WW/RS/EW (Warm White FL)': ...,
'F40/C50 (Broadband FL)': ..., 'F40/C75 (Broadband FL)': ..., 'F40/CWX (Broadband FL)':
..., 'F40/DX (Broadband FL)': ..., 'F40/DXTP (Delux FL)': ..., 'F40/N (Natural FL)': ...,
'H38HT-100 (Mercury)': ..., 'H38JA-100/DX (Mercury DX)': ..., 'MHC100/U/MP/3K': ...,
'MHC100/U/MP/4K': ..., 'SDW-T 100W/LV (Super HPS)': ..., 'Kinoton 75P': ...})
```

Spectral distributions of the light sources.

Notes

- *CIE 15:2004* recommends using linear interpolation for *CIE Standard Illuminant D Series*, for consistency all the illuminants are using a linear interpolator.

References

[Hou15], [OD08], [Poi80]

colour.TVS_ILLUMINANTS

```
colour.TVS_ILLUMINANTS = CanonicalMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})
```

CIE XYZ tristimulus values of the illuminants.

References

[CSH+18]

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.TVS_ILLUMINANTS_HUNTERLAB

```
colour.TVS_ILLUMINANTS_HUNTERLAB = CanonicalMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})
```

CIE XYZ tristimulus values of the *HunterLab* illuminants.

References

[HunterLab08a], [HunterLab08b]

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

Ancillary Objects

`colour.colorimetry`

SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES	*CIE Illuminant D Series* $S_n(\lambda)$ spectral distributions.

colour.colorimetry.SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES

```
colour.colorimetry.SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES = LazyCanonicalMapping({'S0': ..., 'S1': ..., 'S2': ...})
```

CIE Illuminant D Series $S_n(\lambda)$ spectral distributions.

References

[Lin07], [WS00k]

Dominant Wavelength and Purity

colour

dominant_wavelength(xy, xy_n[, cmfs, inverse])	Return 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])	Return 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])	Return the *excitation purity* P_e for given colour stimulus xy.
colorimetric_purity(xy, xy_n[, cmfs])	Return the *colorimetric purity* P_c for given colour stimulus xy.

colour.dominant_wavelength

colour.**dominant_wavelength**(*xy: ArrayLike*, *xy_n: ArrayLike*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*, *inverse: bool = False*) → Tuple[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ced1bd510>, <sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ced1bff50>, <sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ced1bc210>]

Return 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 as xy_wl first intersection coordinates will be set to the *complementary dominant wavelength* intersection coordinates with the spectral locus.

Parameters

- **xy** (ArrayLike) – Colour stimulus *CIE xy* chromaticity coordinates.
- **xy_n** (ArrayLike) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **inverse** (bool) – Inverse the computation direction to retrieve the *complementary wavelength*.

Returns *Dominant wavelength*, first intersection point *CIE xy* chromaticity coordinates, second intersection point *CIE xy* chromaticity coordinates.

Return type tuple

References

[CIET14804b], [Erdb]

Examples

Dominant wavelength computation:

```
>>> from colour.colorimetry import MSDS_CMFS
>>> from pprint import pprint
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> pprint(dominant_wavelength(xy, xy_n, cmfs))
(array(616...),
 array([ 0.6835474...,  0.3162840...]),
 array([ 0.6835474...,  0.3162840...]))
```

Complementary dominant wavelength is returned if the first intersection is located on the line of purples:

```
>>> xy = np.array([0.37605506, 0.24452225])
>>> pprint(dominant_wavelength(xy, xy_n))
(array(-509.0),
 array([ 0.4572314...,  0.1362814...]),
 array([ 0.0104096...,  0.7320745...]))
```

colour.complementary_wavelength

colour.**complementary_wavelength**(*xy: ArrayLike*, *xy_n: ArrayLike*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*) → Tuple[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cec6ba790>, <sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cec6ba750>, <sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cec6c7a50>]

Return 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** (`ArrayLike`) – Colour stimulus *CIE xy* chromaticity coordinates.
- **xy_n** (`ArrayLike`) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | None`) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.

Returns *Complementary wavelength*, first intersection point *CIE xy* chromaticity coordinates, second intersection point *CIE xy* chromaticity coordinates.

Return type `tuple`

References

[CIET14804b], [Erdb]

Examples

Complementary wavelength computation:

```
>>> from colour.colorimetry import MSDS_CMFS
>>> from pprint import pprint
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> xy = np.array([0.37605506, 0.24452225])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> pprint(complementary_wavelength(xy, xy_n, cmfs))
(array(509.0),
 array([ 0.0104096...,  0.7320745...]),
 array([ 0.0104096...,  0.7320745...]))
```

Dominant wavelength is returned if the first intersection is located on the line of purples:

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> pprint(complementary_wavelength(xy, xy_n))
(array(492.0),
 array([ 0.0364795 ,  0.3384712...]),
 array([ 0.0364795 ,  0.3384712...]))
```

colour.excitation_purity

`colour.excitation_purity`(*xy: ArrayLike*, *xy_n: ArrayLike*, *cmfs:* colour.colorimetry.spectrum.MultiSpectralDistributions | *None* = *None*) → NDArrayFloat

Return the *excitation purity* P_e for given colour stimulus xy.

Parameters

- **xy** (`ArrayLike`) – Colour stimulus *CIE xy* chromaticity coordinates.
- **xy_n** (`ArrayLike`) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | None`) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.

Returns *Excitation purity* P_e.

Return type `np.float` or `numpy.ndarray`

References

[CIET14804b], [Erdb]

Examples

```
>>> from colour.colorimetry import MSDS_CMFS
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> excitation_purity(xy, xy_n, cmfs)
0.6228856...
```

colour.colorimetric_purity

colour.**colorimetric_purity**(*xy: ArrayLike*, *xy_n: ArrayLike*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*) → NDArrayFloat

Return the *colorimetric purity* P_c for given colour stimulus xy.

Parameters

- **xy** (ArrayLike) – Colour stimulus *CIE xy* chromaticity coordinates.
- **xy_n** (ArrayLike) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.

Returns *Colorimetric purity* P_c.

Return type np.float or numpy.ndarray

References

[CIET14804b], [Erdb]

Examples

```
>>> from colour.colorimetry import MSDS_CMFS
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> colorimetric_purity(xy, xy_n, cmfs)
0.6135828...
```

Luminous Efficiency Functions

colour

luminous_efficacy(sd[, lef])	Return the *luminous efficacy* in $lm \cdot W^{-1}$ of given spectral distribution using given luminous efficiency function.
luminous_efficiency(sd[, lef])	Return the *luminous efficiency* of given spectral distribution using given luminous efficiency function.
luminous_flux(sd[, lef, K_m])	Return the *luminous flux* for given spectral distribution using given luminous efficiency function.
sd_mesopic_luminous_efficiency_function(L_p)	Return the mesopic luminous efficiency function $V_m(\lambda)$ for given photopic luminance L_p.

colour.luminous_efficacy

colour.**luminous_efficacy**(*sd:* colour.colorimetry.spectrum.SpectralDistribution, *lef:* colour.colorimetry.spectrum.SpectralDistribution | *None* = *None*) → float

Return the *luminous efficacy* in $lm \cdot W^{-1}$ of given spectral distribution using given luminous efficiency function.

Parameters

- **sd** (colour.colorimetry.spectrum.SpectralDistribution) – test spectral distribution
- **lef** (colour.colorimetry.spectrum.SpectralDistribution | None) – $V(\lambda)$ luminous efficiency function, default to the *CIE 1924 Photopic Standard Observer*.

Returns Luminous efficacy in $lm \cdot W^{-1}$.

Return type float

References

[Wikipedia05b]

Examples

```
>>> from colour import SDS_LIGHT_SOURCES
>>> sd = SDS_LIGHT_SOURCES["Neodimium Incandescent"]
>>> luminous_efficacy(sd)
136.2170803...
```

colour.luminous_efficiency

colour.**luminous_efficiency**(*sd:* colour.colorimetry.spectrum.SpectralDistribution, *lef:* colour.colorimetry.spectrum.SpectralDistribution | *None* = *None*) → float

Return the *luminous efficiency* of given spectral distribution using given luminous efficiency function.

Parameters

- **sd** (colour.colorimetry.spectrum.SpectralDistribution) – test spectral distribution
- **lef** (colour.colorimetry.spectrum.SpectralDistribution | None) – $V(\lambda)$ luminous efficiency function, default to the *CIE 1924 Photopic Standard Observer*.

Returns Luminous efficiency.

Return type float

References

[Wikipedia03b]

Examples

```
>>> from colour import SDS_LIGHT_SOURCES
>>> sd = SDS_LIGHT_SOURCES["Neodimium Incandescent"]
>>> luminous_efficiency(sd)
0.1994393...
```

colour.luminous_flux

colour.**luminous_flux**(*sd: colour.colorimetry.spectrum.SpectralDistribution, lef: colour.colorimetry.spectrum.SpectralDistribution | None = None, K_m: float = CONSTANT_K_M*) → float

Return the *luminous flux* for given spectral distribution using given luminous efficiency function.

Parameters

- **sd** (colour.colorimetry.spectrum.SpectralDistribution) – test spectral distribution
- **lef** (colour.colorimetry.spectrum.SpectralDistribution | None) – $V(\lambda)$ luminous efficiency function, default to the *CIE 1924 Photopic Standard Observer*.
- **K_m** (float) – $lm \cdot W^{-1}$ maximum photopic luminous efficiency.

Returns Luminous flux.

Return type float

References

[Wikipedia03b]

Examples

```
>>> from colour import SDS_LIGHT_SOURCES
>>> sd = SDS_LIGHT_SOURCES["Neodimium Incandescent"]
>>> luminous_flux(sd)
23807.6555273...
```

colour.sd_mesopic_luminous_efficiency_function

colour.**sd_mesopic_luminous_efficiency_function**(*L_p: float, source: Union[Literal['Blue Heavy', 'Red Heavy'], str] = 'Blue Heavy', method: Union[Literal['MOVE', 'LRC'], str] = 'MOVE', photopic_lef: colour.colorimetry.spectrum.SpectralDistribution | None = None, scotopic_lef: colour.colorimetry.spectrum.SpectralDistribution | None = None*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return the mesopic luminous efficiency function $V_m(\lambda)$ for given photopic luminance L_p.

Parameters

- **L_p** (float) – Photopic luminance L_p.
- **source** (Union[Literal['Blue Heavy', 'Red Heavy'], str]) – Light source colour temperature.
- **method** (Union[Literal['MOVE', 'LRC'], str]) – Method to calculate the weighting factor.
- **photopic_lef** (colour.colorimetry.spectrum.SpectralDistribution | None) – $V(\lambda)$ photopic luminous efficiency function, default to the *CIE 1924 Photopic Standard Observer*.
- **scotopic_lef** (colour.colorimetry.spectrum.SpectralDistribution | None) – $V'(\lambda)$ scotopic luminous efficiency function, default to the *CIE 1951 Scotopic Standard Observer*.

Returns Mesopic luminous efficiency function $V_m(\lambda)$.

Return type colour.SpectralDistribution

References

[Wikipedia05c]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_mesopic_luminous_efficiency_function(0.2)
...
SpectralDistribution([[ 380.        ,    0.000424 ...],
                      [ 381.        ,    0.0004781...],
                      [ 382.        ,    0.0005399...],
                      [ 383.        ,    0.0006122...],
                      [ 384.        ,    0.0006961...],
                      [ 385.        ,    0.0007929...],
                      [ 386.        ,    0.000907 ...],
                      [ 387.        ,    0.0010389...],
                      [ 388.        ,    0.0011923...],
                      [ 389.        ,    0.0013703...],
                      [ 390.        ,    0.0015771...],
                      [ 391.        ,    0.0018167...],
                      [ 392.        ,    0.0020942...],
                      [ 393.        ,    0.0024160...],
```

(continues on next page)

(continued from previous page)

```
                        [ 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...],
```

(continues on next page)

(continued from previous page)

```
[ 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...],
```

(continues on next page)

(continued from previous page)

```
                  [ 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...],
```

(continues on next page)

(continued from previous page)

```
                    [ 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...],
```

(continues on next page)

(continued from previous page)

```
                                [ 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...],
```

(continues on next page)

(continued from previous page)

```
                [ 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...],
```

(continues on next page)

(continued from previous page)

```
                 [ 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...]],
                SpragueInterpolator,
                {},
                Extrapolator,
                {'method': 'Constant', 'left': None, 'right': None})
```

Dataset

colour

SDS_LEFS	Spectral distributions of the luminous efficiency functions.

colour.SDS_LEFS

colour.**SDS_LEFS = LazyCanonicalMapping({'CIE 1924 Photopic Standard Observer': ..., 'Judd Modified CIE 1951 Photopic Standard Observer': ..., 'Judd-Vos Modified CIE 1978 Photopic Standard Observer': ..., 'CIE 1964 Photopic 10 Degree Standard Observer': ..., 'CIE 2008 2 Degree Physiologically Relevant LEF': ..., 'CIE 2008 10 Degree Physiologically Relevant LEF': ..., 'cie_2_1924': ..., 'cie_10_1964': ..., 'CIE 1951 Scotopic Standard Observer': ..., 'cie_1951': ...})**

Spectral distributions of the luminous efficiency functions.

References

[CVRLa], [CVRLc], [Wikipedia05c]

Ancillary Objects

colour.colorimetry

SDS_LEFS_PHOTOPIC	Spectral distributions of the photopic luminous efficiency functions.
SDS_LEFS_SCOTOPIC	Spectral distributions of the scotopic luminous efficiency functions.

colour.colorimetry.SDS_LEFS_PHOTOPIC

colour.colorimetry.**SDS_LEFS_PHOTOPIC = LazyCanonicalMapping({'CIE 1924 Photopic Standard Observer': ..., 'Judd Modified CIE 1951 Photopic Standard Observer': ..., 'Judd-Vos Modified CIE 1978 Photopic Standard Observer': ..., 'CIE 1964 Photopic 10 Degree Standard Observer': ..., 'CIE 2008 2 Degree Physiologically Relevant LEF': ..., 'CIE 2008 10 Degree Physiologically Relevant LEF': ..., 'cie_2_1924': ..., 'cie_10_1964': ...})**

Spectral distributions of the photopic luminous efficiency functions.

References

[CVRLa], [CVRLc]

Aliases:

- 'cie_2_1924': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 Photopic 10 Degree Standard Observer'

colour.colorimetry.SDS_LEFS_SCOTOPIC

`colour.colorimetry.SDS_LEFS_SCOTOPIC = LazyCanonicalMapping({'CIE 1951 Scotopic Standard Observer': ..., 'cie_1951': ...})`

Spectral distributions of the scotopic luminous efficiency functions.

References

[CVRLc]

Aliases:

- 'cie_1951': 'CIE 1951 Scotopic Standard Observer'

Spectral Uniformity

colour

`spectral_uniformity`(sds[, ...])	Compute the *spectral uniformity* (or *spectral flatness*) of given spectral distributions.

colour.spectral_uniformity

colour.**spectral_uniformity**(*sds: collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions*, *use_second_order_derivatives: bool = False*) → NDArrayFloat

Compute the *spectral uniformity* (or *spectral flatness*) of given spectral distributions.

Spectral uniformity $(r')^2$ is computed as follows:

$mean((r'_1)^2, (r'_2)^2, ..., (r'_n)^2)$

where $(r'_i)^2$ is the first-order derivative, squared, of the reflectance r_i of a test sample.

Parameters

- **sds** (`collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions`) – Spectral distributions or multi-spectral distributions to compute the spectral uniformity of. *sds* can be a single `colour.MultiSpectralDistributions` class instance, a list of `colour.MultiSpectralDistributions` class instances or a List of `colour.SpectralDistribution` class instances.
- **use_second_order_derivatives** (`bool`) – Whether to use the second-order derivatives in the computations.

Returns Spectral uniformity.

Return type `numpy.ndarray`

> **Warning:** The spectral distributions must have the same spectral shape.

References

[DFH+15]

Examples

```
>>> from colour.quality.datasets import SDS_TCS
>>> spectral_uniformity(SDS_TCS.values())
array([  9.5514285...e-06,   1.1482142...e-05,   1.8784285...e-05,
         2.8711428...e-05,   3.1971428...e-05,   3.2342857...e-05,
         3.3850000...e-05,   3.9925714...e-05,   4.1333571...e-05,
         2.4002142...e-05,   5.7621428...e-06,   1.4757142...e-06,
         9.7928571...e-07,   2.0057142...e-06,   3.7157142...e-06,
         5.7678571...e-06,   7.5557142...e-06,   7.4635714...e-06,
         5.7492857...e-06,   3.8692857...e-06,   3.5407142...e-06,
         4.4742857...e-06,   5.6435714...e-06,   7.6371428...e-06,
         1.0171428...e-05,   1.2254285...e-05,   1.4810000...e-05,
         1.6517142...e-05,   1.5430714...e-05,   1.4536428...e-05,
         1.4037857...e-05,   1.1587857...e-05,   1.0743571...e-05,
         1.0979285...e-05,   1.0398571...e-05,   8.2971428...e-06,
         6.3057142...e-06,   5.0942857...e-06,   4.8500000...e-06,
         5.5371428...e-06,   6.4128571...e-06,   7.2592857...e-06,
         7.7750000...e-06,   7.1607142...e-06,   6.6635714...e-06,
         6.7328571...e-06,   7.5307142...e-06,   1.0733571...e-05,
         1.6234285...e-05,   2.2570714...e-05,   2.7056428...e-05,
         2.7781428...e-05,   2.5025714...e-05,   1.7966428...e-05,
         1.0505000...e-05,   5.9657142...e-06,   3.6421428...e-06,
         2.1664285...e-06,   1.2935714...e-06,   8.3642857...e-07,
         7.2500000...e-07,   6.3928571...e-07,   6.6285714...e-07,
         8.5571428...e-07,   1.4507142...e-06,   2.2542857...e-06,
         3.4142857...e-06,   4.9864285...e-06,   6.4907142...e-06,
         7.8928571...e-06,   9.1664285...e-06,   9.9521428...e-06,
         9.7664285...e-06,   9.3150000...e-06,   8.9092857...e-06,
         8.1578571...e-06,   6.8935714...e-06,   5.5721428...e-06,
         4.4592857...e-06,   3.4778571...e-06,   2.7650000...e-06,
         2.3114285...e-06,   1.7092857...e-06,   1.1771428...e-06,
         9.8428571...e-07,   8.8285714...e-07,   7.4142857...e-07,
         7.0142857...e-07,   7.0857142...e-07,   6.6642857...e-07,
         7.5928571...e-07,   8.7000000...e-07,   8.2714285...e-07,
         7.1714285...e-07,   6.6000000...e-07])
```

Lightness Computation

colour

lightness(Y[, method])	Return the *Lightness* L of given *luminance* Y using given method.
LIGHTNESS_METHODS	Supported *Lightness* computation methods.

colour.lightness

colour.**lightness**(*Y: ArrayLike, method: Union[Literal['Abebe 2017', 'CIE 1976', 'Glasser 1958', 'Fairchild 2010', 'Fairchild 2011', 'Wyszecki 1963'], str] = 'CIE 1976', **kwargs: Any*) → NDArrayFloat

Return the *Lightness* L of given *luminance* Y using given method.

Parameters

- **Y** (ArrayLike) – *Luminance* Y.
- **method** (Union[Literal['Abebe 2017', 'CIE 1976', 'Glasser 1958', 'Fairchild 2010', 'Fairchild 2011', 'Wyszecki 1963'], str]) – Computation method.
- **Y_n** – {colour.colorimetry.lightness_Abebe2017(), colour.colorimetry.lightness_CIE1976()}, White reference *luminance* Y_n.
- **epsilon** – {colour.colorimetry.lightness_Fairchild2010(), colour.colorimetry.lightness_Fairchild2011()}, ϵ exponent.
- **kwargs** (Any) –

Returns *Lightness* L.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 100]	[0, 1]

References

[APLR17], [CIET14804h], [FW10], [FC11], [GMRS58], [Wikipedia07a], [Wys63], [WS00l]

Examples

```
>>> lightness(12.19722535)
41.5278758...
>>> lightness(12.19722535, Y_n=100)
41.5278758...
>>> lightness(12.19722535, Y_n=95)
42.5199307...
>>> lightness(12.19722535, method="Glasser 1958")
39.8351264...
>>> lightness(12.19722535, method="Wyszecki 1963")
40.5475745...
>>> lightness(12.19722535, epsilon=0.710, method="Fairchild 2011")
...
29.8295108...
>>> lightness(12.19722535, epsilon=0.710, method="Fairchild 2011")
...
```

(continues on next page)

(continued from previous page)

```
29.8295108...
>>> lightness(12.19722535, method="Abebe 2017")
...
48.6955571...
```

colour.LIGHTNESS_METHODS

colour.**LIGHTNESS_METHODS = CanonicalMapping({'Glasser 1958': ..., 'Wyszecki 1963': ..., 'CIE 1976': ..., 'Fairchild 2010': ..., 'Fairchild 2011': ..., 'Abebe 2017': ..., 'Lstar1976': ...})**

Supported *Lightness* computation methods.

References

[CIET14804h], [FW10], [FC11], [GMRS58], [Wys63], [WS00l]

Aliases:

- 'Lstar1976': 'CIE 1976'

Glasser, Mckinney, Reilly and Schnelle (1958)

colour.colorimetry

lightness_Glasser1958(Y)	Return the *Lightness* L of given *luminance* Y using *Glasser et al. (1958)* method.

colour.colorimetry.lightness_Glasser1958

colour.colorimetry.**lightness_Glasser1958**(*Y: ArrayLike*) → NDArrayFloat

Return the *Lightness* L of given *luminance* Y using *Glasser et al. (1958)* method.

Parameters **Y** (ArrayLike) – *Luminance* Y.

Returns *Lightness* L.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 100]	[0, 1]

References

[GMRS58]

Examples

```
>>> lightness_Glasser1958(12.19722535)
39.8351264...
```

Wyszecki (1963)

colour.colorimetry

lightness_Wyszecki1963(Y)	Return the *Lightness* W of given *luminance* Y using *Wyszecki (1963)* method.

colour.colorimetry.lightness_Wyszecki1963

colour.colorimetry.**lightness_Wyszecki1963**(*Y: ArrayLike*) → NDArrayFloat

Return the *Lightness* W of given *luminance* Y using *Wyszecki (1963)* method.

Parameters Y (ArrayLike) – *Luminance* Y.

Returns *Lightness* W.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
W	[0, 100]	[0, 1]

References

[Wys63]

Examples

```
>>> lightness_Wyszecki1963(12.19722535)
40.5475745...
```

CIE 1976

colour.colorimetry

lightness_CIE1976(Y[, Y_n])	Return the *Lightness* L^* of given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.
intermediate_lightness_function_CIE1976(Y[, Y_n])	Return the intermediate value $f(Y/Yn)$ in the *Lightness* L^* computation for given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

colour.colorimetry.lightness_CIE1976

colour.colorimetry.**lightness_CIE1976**(*Y: ArrayLike*, *Y_n: ArrayLike = 100*) → NDArrayFloat

Return the *Lightness* L^* of given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

Parameters

- **Y** (ArrayLike) – *Luminance* Y.
- **Y_n** (ArrayLike) – White reference *luminance* Y_n.

Returns *Lightness* L^*.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
L_star	[0, 100]	[0, 1]

References

[CIET14804h], [WS00l]

Examples

```
>>> lightness_CIE1976(12.19722535)
41.5278758...
```

colour.colorimetry.intermediate_lightness_function_CIE1976

colour.colorimetry.**intermediate_lightness_function_CIE1976**(*Y: ArrayLike*, *Y_n: ArrayLike = 100*) → NDArrayFloat

Return the intermediate value $f(Y/Yn)$ in the *Lightness* L^* computation for given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

Parameters

- **Y** (ArrayLike) – *Luminance* Y.
- **Y_n** (ArrayLike) – White reference *luminance* Y_n.

Returns Intermediate value $f(Y/Yn)$.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 100]

Range	Scale - Reference	Scale - 1
f_Y_Y_n	[0, 1]	[0, 1]

References

[CIET14804h], [WS00l]

Examples

```
>>> intermediate_lightness_function_CIE1976(12.19722535)
...
0.4959299...
>>> intermediate_lightness_function_CIE1976(12.19722535, 95)
...
0.5044821...
```

Fairchild and Wyble (2010)

colour.colorimetry

lightness_Fairchild2010(Y[, epsilon])	Compute *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Wyble (2010)* method according to *Michaelis-Menten* kinetics.

colour.colorimetry.lightness_Fairchild2010

colour.colorimetry.**lightness_Fairchild2010**(*Y: ArrayLike, epsilon: ArrayLike = 1.836*) → NDArrayFloat

Compute *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Wyble (2010)* method according to *Michaelis-Menten* kinetics.

Parameters

- **Y** (ArrayLike) – *Luminance* Y.
- **epsilon** (ArrayLike) – ϵ exponent.

Returns *Lightness* L_{hdr}.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

References

[FW10]

Examples

```
>>> lightness_Fairchild2010(12.19722535 / 100)
31.9963902...
```

Fairchild and Chen (2011)

colour.colorimetry

lightness_Fairchild2011(Y[, epsilon, method])	Compute *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Chen (2011)* method according to *Michaelis-Menten* kinetics.

colour.colorimetry.lightness_Fairchild2011

colour.colorimetry.**lightness_Fairchild2011**(*Y: ArrayLike, epsilon: ArrayLike = 0.474, method: Union[Literal['hdr-CIELAB', 'hdr-IPT'], str] = 'hdr-CIELAB'*) → NDArrayFloat

Compute *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Chen (2011)* method according to *Michaelis-Menten* kinetics.

Parameters

- **Y** (ArrayLike) – *Luminance* Y.

- **epsilon** (ArrayLike) – ϵ exponent.
- **method** (Union[Literal['hdr-CIELAB', 'hdr-IPT'], str]) – *Lightness* L_{hdr} computation method.

Returns *Lightness* L_{hdr}.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

References

[FC11]

Examples

```
>>> lightness_Fairchild2011(12.19722535 / 100)
51.8529584...
>>> lightness_Fairchild2011(12.19722535 / 100, method="hdr-IPT")
...
51.6431084...
```

Abebe, Pouli, Larabi and Reinhard (2017)

colour.colorimetry

lightness_Abebe2017(Y[, Y_n, method])	Compute *Lightness* L of given *luminance* Y using *Abebe, Pouli, Larabi and Reinhard (2017)* method according to *Michaelis-Menten* kinetics or *Stevens's Power Law*.

colour.colorimetry.lightness_Abebe2017

colour.colorimetry.**lightness_Abebe2017**(*Y: ArrayLike, Y_n: ArrayLike = 100, method: Union[Literal['Michaelis-Menten', 'Stevens'], str] = 'Michaelis-Menten'*) → NDArrayFloat

Compute *Lightness* L of given *luminance* Y using *Abebe, Pouli, Larabi and Reinhard (2017)* method according to *Michaelis-Menten* kinetics or *Stevens's Power Law*.

Parameters

- **Y** (ArrayLike) – *Luminance* Y in cd/m^2.
- **Y_n** (ArrayLike) – Adapting luminance Y_n in cd/m^2.
- **method** (Union[Literal['Michaelis-Menten', 'Stevens'], str]) – *Lightness* L computation method.

Returns *Lightness* L.

Return type numpy.ndarray

Notes

- *Abebe, Pouli, Larabi and Reinhard (2017)* method uses absolute luminance levels, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
Y	UN	UN
Y_n	UN	UN

Range	Scale - Reference	Scale - 1
L	UN	UN

References

[APLR17]

Examples

```
>>> lightness_Abebe2017(12.19722535)
0.4869555...
>>> lightness_Abebe2017(12.19722535, method="Stevens")
...
0.4745447...
```

Luminance Computation

colour

luminance(LV[, method])	Return the *luminance* Y of given *Lightness* L^* or given *Munsell* value V.
LUMINANCE_METHODS	Supported *luminance* computation methods.

colour.luminance

colour.**luminance**(*LV: ArrayLike, method: Union[Literal['Abebe 2017', 'CIE 1976', 'Glasser 1958', 'Fairchild 2010', 'Fairchild 2011', 'Wyszecki 1963'], str] = 'CIE 1976', **kwargs: Any*) → NDArrayFloat

Return the *luminance* Y of given *Lightness* L^* or given *Munsell* value V.

Parameters

- **LV** (ArrayLike) – *Lightness* L^* or *Munsell* value V.
- **method** (Union[Literal['Abebe 2017', 'CIE 1976', 'Glasser 1958', 'Fairchild 2010', 'Fairchild 2011', 'Wyszecki 1963'], str]) – Computation method.

- **Y_n** – {colour.colorimetry.luminance_Abebe2017(), colour.colorimetry.luminance_CIE1976()}, White reference *luminance* Y_n.
- **epsilon** – {colour.colorimetry.lightness_Fairchild2010(), colour.colorimetry.lightness_Fairchild2011()}, ϵ exponent.
- **kwargs** (Any) –

Returns *Luminance* Y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
LV	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

References

[APLR17], [ASTMInternational08], [CIET14804h], [FW10], [FC11], [NNJ43], [Wikipedia01c], [WS00l]

Examples

```
>>> luminance(41.527875844653451)
12.1972253...
>>> luminance(41.527875844653451, Y_n=100)
12.1972253...
>>> luminance(42.51993072812094, Y_n=95)
12.1972253...
>>> luminance(4.08244375 * 10, method="Newhall 1943")
...
12.5500788...
>>> luminance(4.08244375 * 10, method="ASTM D1535")
...
12.2363426...
>>> luminance(29.829510892279330, epsilon=0.710, method="Fairchild 2011")
...
12.1972253...
```

colour.LUMINANCE_METHODS

colour.**LUMINANCE_METHODS = CanonicalMapping({'Newhall 1943': ..., 'ASTM D1535': ..., 'CIE 1976': ..., 'Fairchild 2010': ..., 'Fairchild 2011': ..., 'Abebe 2017': ..., 'astm2008': ..., 'cie1976': ...})**

Supported *luminance* computation methods.

References

[ASTMInternational08], [CIET14804h], [FW10], [FC11], [NNJ43], [WS00l]

Aliases:

- 'astm2008': 'ASTM D1535'
- 'cie1976': 'CIE 1976'

Newhall, Nickerson and Judd (1943)

`colour.colorimetry`

`luminance_Newhall1943`(V)	Return the *luminance* R_Y of given *Munsell* value V using *Newhall et al. (1943)* method.

colour.colorimetry.luminance_Newhall1943

`colour.colorimetry.`**`luminance_Newhall1943`**(*V: ArrayLike*) → NDArrayFloat

Return the *luminance* R_Y of given *Munsell* value V using *Newhall et al. (1943)* method.

Parameters **V** (`ArrayLike`) – *Munsell* value V.

Returns *Luminance* R_Y.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

Range	Scale - Reference	Scale - 1
R_Y	[0, 100]	[0, 1]

References

[NNJ43]

Examples

```
>>> luminance_Newhall1943(4.08244375)
12.5500788...
```

CIE 1976

colour.colorimetry

luminance_CIE1976(L_star[, Y_n])	Return the *luminance* Y of given *Lightness* L^* with given reference white *luminance* Y_n.
intermediate_luminance_function_CIE1976(f_Y_Y_n)	Return the *luminance* Y in the *luminance* Y computation for given intermediate value $f(Y/Yn)$ using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

colour.colorimetry.luminance_CIE1976

colour.colorimetry.**luminance_CIE1976**(*L_star: ArrayLike*, *Y_n: ArrayLike = 100*) → NDArrayFloat

Return the *luminance* Y of given *Lightness* L^* with given reference white *luminance* Y_n.

Parameters

- **L_star** (ArrayLike) – *Lightness* L^*
- **Y_n** (ArrayLike) – White reference *luminance* Y_n.

Returns *Luminance* Y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_star	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

References

[CIET14804h], [WS00l]

Examples

```
>>> luminance_CIE1976(41.527875844653451)
12.1972253...
>>> luminance_CIE1976(41.527875844653451, 95)
11.5873640...
```

colour.colorimetry.intermediate_luminance_function_CIE1976

colour.colorimetry.**intermediate_luminance_function_CIE1976**(*f_Y_Y_n: ArrayLike*, *Y_n: ArrayLike = 100*) → NDArrayFloat

Return the *luminance* Y in the *luminance* Y computation for given intermediate value $f(Y/Yn)$ using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

Parameters

- **f_Y_Y_n** (ArrayLike) – Intermediate value $f(Y/Yn)$.
- **Y_n** (ArrayLike) – White reference *luminance* Y_n.

Returns *Luminance* Y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
f_Y_Y_n	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 100]

References

[CIET14804h], [WS00l]

Examples

```
>>> intermediate_luminance_function_CIE1976(0.495929964178047)
...
12.1972253...
>>> intermediate_luminance_function_CIE1976(0.504482161449319, 95)
...
12.1972253...
```

ASTM D1535-08e1

colour.colorimetry

luminance_ASTMD1535(V)	Return the *luminance* Y of given *Munsell* value V using *ASTM D1535-08e1* method.

colour.colorimetry.luminance_ASTMD1535

`colour.colorimetry.luminance_ASTMD1535(V: ArrayLike) → NDArrayFloat`

Return the *luminance* Y of given *Munsell* value V using *ASTM D1535-08e1* method.

Parameters **V** (ArrayLike) – *Munsell* value V.

Returns *Luminance* Y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

References

[ASTMInternational08]

Examples

```
>>> luminance_ASTMD1535(4.08244375)
12.2363426...
```

Fairchild and Wyble (2010)

`colour.colorimetry`

luminance_Fairchild2010(L_hdr[, epsilon])	Compute *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Wyble (2010)* method according to *Michaelis-Menten* kinetics.

colour.colorimetry.luminance_Fairchild2010

`colour.colorimetry.luminance_Fairchild2010(L_hdr: ArrayLike, epsilon: ArrayLike = 1.836) → NDArrayFloat`

Compute *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Wyble (2010)* method according to *Michaelis-Menten* kinetics.

Parameters

- **L_hdr** (ArrayLike) – *Lightness* L_{hdr}.
- **epsilon** (ArrayLike) – ϵ exponent.

Returns *Luminance* Y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

References

[FW10]

Examples

```
>>> luminance_Fairchild2010(31.996390226262736, 1.836)
...
0.1219722...
```

Fairchild and Chen (2011)

`colour.colorimetry`

luminance_Fairchild2011(L_hdr[, epsilon, method])	Compute *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Chen (2011)* method according to *Michaelis-Menten* kinetics.

colour.colorimetry.luminance_Fairchild2011

colour.colorimetry.**luminance_Fairchild2011**(*L_hdr: ArrayLike, epsilon: ArrayLike = 0.474, method: Union[Literal['hdr-CIELAB', 'hdr-IPT'], str] = 'hdr-CIELAB'*) → NDArrayFloat

Compute *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Chen (2011)* method according to *Michaelis-Menten* kinetics.

Parameters

- **L_hdr** (ArrayLike) – *Lightness* L_{hdr}.
- **epsilon** (ArrayLike) – ϵ exponent.
- **method** (Union[Literal['hdr-CIELAB', 'hdr-IPT'], str]) – *Lightness* L_{hdr} computation method.

Returns *Luminance* Y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

References

[FC11]

Examples

```
>>> luminance_Fairchild2011(51.852958445912506)
0.1219722...
>>> luminance_Fairchild2011(51.643108411718522, method="hdr-IPT")
...
0.1219722...
```

Whiteness Computation

colour

whiteness(XYZ, XYZ_0[, method])	Return the *whiteness* W using given method.
WHITENESS_METHODS	Supported *whiteness* computation methods.

colour.whiteness

colour.**whiteness**(*XYZ: ArrayLike, XYZ_0: ArrayLike, method: Union[Literal['ASTM E313', 'CIE 2004', 'Berger 1959', 'Ganz 1979', 'Stensby 1968', 'Taube 1960'], str] = 'CIE 2004', **kwargs: Any*) → NDArrayFloat

Return the *whiteness* W using given method.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of the sample.
- **XYZ_0** (ArrayLike) – *CIE XYZ* tristimulus values of the reference white.
- **method** (Union[Literal['ASTM E313', 'CIE 2004', 'Berger 1959', 'Ganz 1979', 'Stensby 1968', 'Taube 1960'], str]) – Computation method.
- **observer** – {colour.colorimetry.whiteness_CIE2004()}, *CIE Standard Observer* used for computations, *tint* T or T_{10} value is dependent on viewing field angular subtense.
- **kwargs** (Any) –

Returns *Whiteness* W.

Return type np.float or numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
W	[0, 100]	[0, 1]

References

[CIET14804d], [WS00i], [XRitePantone12], [Wikipedia04c]

Examples

```
>>> import numpy as np
>>> from colour.models import xyY_to_XYZ
>>> XYZ = xyY_to_XYZ(np.array([0.3167, 0.3334, 100]))
>>> XYZ_0 = xyY_to_XYZ(np.array([0.3139, 0.3311, 100]))
>>> whiteness(XYZ, XYZ_0)
array([ 93.85...,  -1.305...])
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> whiteness(XYZ, XYZ_0, method="Taube 1960")
91.4071738...
```

colour.WHITENESS_METHODS

colour.**WHITENESS_METHODS = CanonicalMapping({'Berger 1959': ..., 'Taube 1960': ..., 'Stensby 1968': ..., 'ASTM E313': ..., 'Ganz 1979': ..., 'CIE 2004': ..., 'cie2004': ...})**

Supported *whiteness* computation methods.

References

[CIET14804d], [XRitePantone12]

Aliases:

- 'cie2004': 'CIE 2004'

Berger (1959)

`colour.colorimetry`

`whiteness_Berger1959`(XYZ, XYZ_0)	Return the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Berger (1959)* method.

colour.colorimetry.whiteness_Berger1959

`colour.colorimetry.whiteness_Berger1959(XYZ: ArrayLike, XYZ_0: ArrayLike) → NDArrayFloat`

Return the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Berger (1959)* method.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of the sample.
- **XYZ_0** (ArrayLike) – *CIE XYZ* tristimulus values of the reference white.

Returns *Whiteness* WI.

Return type np.float or numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

- *Whiteness* WI values larger than 33.33 indicate a bluish white and values smaller than 33.33 indicate a yellowish white.

References

[XRitePantone12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> whiteness_Berger1959(XYZ, XYZ_0)
30.3638017...
```

Taube (1960)

`colour.colorimetry`

whiteness_Taube1960(XYZ, XYZ_0)	Return the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Taube (1960)* method.

colour.colorimetry.whiteness_Taube1960

`colour.colorimetry.whiteness_Taube1960`(*XYZ: ArrayLike*, *XYZ_0: ArrayLike*) → NDArrayFloat

Return the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Taube (1960)* method.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of the sample.
- **XYZ_0** (ArrayLike) – *CIE XYZ* tristimulus values of the reference white.

Returns *Whiteness* WI.

Return type np.float or numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

- *Whiteness* WI values larger than 100 indicate a bluish white and values smaller than 100 indicate a yellowish white.

References

[XRitePantone12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> whiteness_Taube1960(XYZ, XYZ_0)
91.4071738...
```

Stensby (1968)

`colour.colorimetry`

whiteness_Stensby1968(Lab)	Return 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: ArrayLike*) → NDArrayFloat

Return the *whiteness* index WI of given sample *CIE L*a*b** colourspace array using *Stensby (1968)* method.

Parameters **Lab** (ArrayLike) – *CIE L*a*b** colourspace array of the sample.

Returns *Whiteness* WI.

Return type np.float or numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

- *Whiteness* WI values larger than 100 indicate a bluish white and values smaller than 100 indicate a yellowish white.

References

[XRitePantone12]

Examples

```
>>> import numpy as np
>>> Lab = np.array([100.00000000, -2.46875131, -16.72486654])
>>> whiteness_Stensby1968(Lab)
142.7683456...
```

ASTM E313

colour.colorimetry

whiteness_ASTME313(XYZ)	Return the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *ASTM E313* method.

colour.colorimetry.whiteness_ASTME313

colour.colorimetry.**whiteness_ASTME313**(*XYZ: ArrayLike*) → NDArrayFloat

Return the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *ASTM E313* method.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of the sample.

Returns *Whiteness* WI.

Return type np.float or numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

References

[XRitePantone12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> whiteness_ASTME313(XYZ)
55.7400000...
```

Ganz and Griesser (1979)

colour.colorimetry

whiteness_Ganz1979(xy, Y)	Return the *whiteness* index W and *tint* T of given sample *CIE xy* chromaticity coordinates using *Ganz and Griesser (1979)* method.

colour.colorimetry.whiteness_Ganz1979

colour.colorimetry.**whiteness_Ganz1979**(*xy: ArrayLike*, *Y: ArrayLike*) → NDArrayFloat

Return the *whiteness* index W and *tint* T of given sample *CIE xy* chromaticity coordinates using *Ganz and Griesser (1979)* method.

Parameters

- **xy** (ArrayLike) – Chromaticity coordinates *CIE xy* of the sample.
- **Y** (ArrayLike) – Tristimulus Y value of the sample.

Returns *Whiteness* W and *tint* T.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WT	[0, 100]	[0, 1]

- The formula coefficients are valid for *CIE Standard Illuminant D Series D65* and *CIE 1964 10 Degree Standard Observer*.
- Positive output *tint* T values indicate a greener tint while negative values indicate a redder tint.
- Whiteness differences of less than 5 Ganz units appear to be indistinguishable to the human eye.
- Tint differences of less than 0.5 Ganz units appear to be indistinguishable to the human eye.

References

[XRitePantone12]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.3167, 0.3334])
>>> whiteness_Ganz1979(xy, 100)
array([ 85.6003766...,   0.6789003...])
```

CIE 2004

`colour.colorimetry`

`whiteness_CIE2004`(xy, Y, xy_n[, observer])	Return the *whiteness* W or W_{10} and *tint* T or T_{10} of given sample *CIE xy* chromaticity coordinates using *CIE 2004* method.

colour.colorimetry.whiteness_CIE2004

`colour.colorimetry.`**`whiteness_CIE2004`**(*xy: ArrayLike*, *Y: ArrayLike*, *xy_n: ArrayLike*, *observer: Literal['CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'] = 'CIE 1931 2 Degree Standard Observer'*) → NDArrayFloat

Return the *whiteness* W or W_{10} and *tint* T or T_{10} of given sample *CIE xy* chromaticity coordinates using *CIE 2004* method.

Parameters

- **xy** (`ArrayLike`) – Chromaticity coordinates *CIE xy* of the sample.
- **Y** (`ArrayLike`) – Tristimulus Y value of the sample.
- **xy_n** (`ArrayLike`) – Chromaticity coordinates *xy_n* of a perfect diffuser.

- **observer** (`Literal['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 `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WT	[0, 100]	[0, 1]

- This method may be used only for samples whose values of W or W_{10} lie within the following limits: greater than 40 and less than 5Y - 280, or 5Y10 - 280.
- This method may be used only for samples whose values of T or T_{10} lie within the following limits: greater than -4 and less than +2.
- Output *whiteness* W or W_{10} values larger than 100 indicate a bluish white while values smaller than 100 indicate a yellowish white.
- Positive output *tint* T or T_{10} values indicate a greener tint while negative values indicate a redder tint.

References

[CIET14804d]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.3167, 0.3334])
>>> xy_n = np.array([0.3139, 0.3311])
>>> whiteness_CIE2004(xy, 100, xy_n)
array([ 93.85...,  -1.305...])
```

Yellowness Computation

`colour`

`yellowness`(XYZ[, method])	Return the *yellowness* W using given method.
`YELLOWNESS_METHODS`	Supported *yellowness* computation methods.

colour.yellowness

colour.**yellowness**(*XYZ: ArrayLike*, *method: Union[Literal['ASTM D1925', 'ASTM E313', 'ASTM E313 Alternative'], str] = 'ASTM E313'*, ***kwargs: Any*) → NDArrayFloat

Return the *yellowness* W using given method.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values of the sample.
- **method** (Union[Literal['ASTM D1925', 'ASTM E313', 'ASTM E313 Alternative'], str]) – Computation method.
- **C_XZ** – {colour.colorimetry.yellowness_ASTME313()}, Coefficients C_X and C_Z for the *CIE 1931 2 Degree Standard Observer* and *CIE 1964 10 Degree Standard Observer* and *CIE Illuminant C* and *CIE Standard Illuminant D65*.
- **kwargs** (Any) –

Returns *Yellowness* Y.

Return type np.float or numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
YI	[0, 100]	[0, 1]

References

[ASTMInternational15b], [XRitePantone12]

Examples

```
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness(XYZ)
4.3400000...
>>> yellowness(XYZ, method="ASTM E313 Alternative")
11.0650000...
>>> yellowness(XYZ, method="ASTM D1925")
10.2999999...
```

colour.YELLOWNESS_METHODS

colour.**YELLOWNESS_METHODS = CanonicalMapping({'ASTM D1925': ..., 'ASTM E313 Alternative': ..., 'ASTM E313': ...})**

Supported *yellowness* computation methods.

References

[ASTMInternational15b], [XRitePantone12]

ASTM D1925

`colour.colorimetry`

`yellowness_ASTMD1925`(XYZ)	Return the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using *ASTM D1925* method.

colour.colorimetry.yellowness_ASTMD1925

`colour.colorimetry.`**`yellowness_ASTMD1925`**(*XYZ: ArrayLike*) → NDArrayFloat

Return 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`** (`ArrayLike`) – *CIE XYZ* tristimulus values of the sample.

Returns *Yellowness* YI.

Return type `np.float` or `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
YI	[0, 100]	[0, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Illuminant C*.

References

[ASTMInternational15b], [XRitePantone12]

Examples

```
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness_ASTMD1925(XYZ)
10.2999999...
```

ASTM E313

`colour.colorimetry`

`yellowness_ASTME313_alternative`(XYZ)	Return the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using the alternative *ASTM E313* method.
`YELLOWNESS_COEFFICIENTS_ASTME313`	Coefficients C_X and C_Z for the *ASTM E313 yellowness* index YI computation method.
`yellowness_ASTME313`(XYZ[, C_XZ])	Return the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using *ASTM E313* method.

colour.colorimetry.yellowness_ASTME313_alternative

`colour.colorimetry.`**`yellowness_ASTME313_alternative`**(*XYZ: ArrayLike*) → NDArrayFloat

Return the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using the alternative *ASTM E313* method.

In the original form of *Test Method E313*, an alternative equation was recommended for a *yellowness* index. In terms of colorimeter readings, it was $YI = 100(1-B/G)$ where B and G are, respectively, blue and green colorimeter readings. Its derivation assumed that, because of the limitation of the concept to yellow (or blue) colors, it was not necessary to take account of variations in the amber or red colorimeter reading A. This equation is no longer recommended.

Parameters **`XYZ`** (`ArrayLike`) – *CIE XYZ* tristimulus values of the sample.

Returns *Yellowness* YI.

Return type `np.float` or `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
`XYZ`	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
`YI`	[0, 100]	[0, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Illuminant C*.

References

[ASTMInternational15b], [XRitePantone12]

Examples

```
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness_ASTME313_alternative(XYZ)
11.0650000...
```

colour.colorimetry.YELLOWNESS_COEFFICIENTS_ASTME313

colour.colorimetry.**YELLOWNESS_COEFFICIENTS_ASTME313 = CanonicalMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})**

Coefficients C_X and C_Z for the *ASTM E313 yellowness* index YI computation method.

References

[ASTMInternational15b]

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.colorimetry.yellowness_ASTME313

colour.colorimetry.**yellowness_ASTME313**(*XYZ: ArrayLike, C_XZ: ArrayLike = YELLOWNESS_COEFFICIENTS_ASTME313['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Return 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** (ArrayLike) – *CIE XYZ* tristimulus values of the sample.
- **C_XZ** (ArrayLike) – Coefficients C_X and C_Z for the *CIE 1931 2 Degree Standard Observer* and *CIE 1964 10 Degree Standard Observer* and *CIE Illuminant C* and *CIE Standard Illuminant D65*.

Returns *Yellowness* YI.

Return type np.float or numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
YI	[0, 100]	[0, 1]

References

[ASTMInternational15b]

Examples

```
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness_ASTME313(XYZ)
4.3400000...
```

Constants

CIE

colour.constants

CONSTANT_K_M	Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.
CONSTANT_KP_M	Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

colour.constants.CONSTANT_K_M

colour.constants.**CONSTANT_K_M = 683.0**

Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.

Notes

- To be adequate for all practical applications the K_m value has been rounded from the original 683.002 value.

References

[WS00d]

colour.constants.CONSTANT_KP_M

`colour.constants.CONSTANT_KP_M = 1700.0`

Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

Notes

- To be adequate for all practical applications the K'_m value has been rounded from the original 1700.06 value.

References

[WS00d]

CODATA

colour.constants

CONSTANT_AVOGADRO	Avogadro constant.
CONSTANT_BOLTZMANN	Boltzmann constant.
CONSTANT_LIGHT_SPEED	Speed of light in vacuum.
CONSTANT_PLANCK	Planck constant.

colour.constants.CONSTANT_AVOGADRO

`colour.constants.CONSTANT_AVOGADRO = 6.02214179e+23`

Avogadro constant.

colour.constants.CONSTANT_BOLTZMANN

`colour.constants.CONSTANT_BOLTZMANN = 1.38065e-23`

Boltzmann constant.

colour.constants.CONSTANT_LIGHT_SPEED

`colour.constants.CONSTANT_LIGHT_SPEED = 299792458.0`

Speed of light in vacuum.

colour.constants.CONSTANT_PLANCK

`colour.constants.CONSTANT_PLANCK = 6.62607e-34`

Planck constant.

Common

`colour.constants`

DEFAULT_INT_DTYPE	alias of `numpy.int64`
DEFAULT_FLOAT_DTYPE	alias of `numpy.float64`
EPSILON	Double-precision floating-point number type, compatible with Python *float* and C `double`.
FLOATING_POINT_NUMBER_PATTERN	str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str
INTEGER_THRESHOLD	int threshold value when checking if a float point number is almost an int.

colour.constants.DEFAULT_INT_DTYPE

colour.constants.**DEFAULT_INT_DTYPE**

alias of `numpy.int64`

colour.constants.DEFAULT_FLOAT_DTYPE

colour.constants.**DEFAULT_FLOAT_DTYPE**

alias of `numpy.float64`

colour.constants.EPSILON

colour.constants.**EPSILON = 2.220446049250313e-16**

Double-precision floating-point number type, compatible with Python *float* and C `double`.

Character code `'d'`

Canonical name *numpy.double*

Alias *numpy.float_*

Alias on this platform (Linux x86_64) *numpy.float64*: 64-bit precision floating-point number type: sign bit, 11 bits exponent, 52 bits mantissa.

colour.constants.FLOATING_POINT_NUMBER_PATTERN

colour.constants.**FLOATING_POINT_NUMBER_PATTERN = '[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)?'**

str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

colour.constants.INTEGER_THRESHOLD

`colour.constants.INTEGER_THRESHOLD = 0.001`

int threshold value when checking if a float point number is almost an int.

Contrast Sensitivity

Contrast Sensitivity

`colour`

`contrast_sensitivity_function([method])`	Return the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by given method.
`CONTRAST_SENSITIVITY_METHODS`	Supported contrast sensitivity methods.

colour.contrast_sensitivity_function

`colour.contrast_sensitivity_function`(*method: Union[Literal['Barten 1999'], str] = 'Barten 1999', **kwargs*) → NDArrayFloat

Return the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by given method.

Parameters

- **method** (Union[Literal['Barten 1999'], str]) – Computation method.
- **E** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Retinal illuminance E in Trolands.
- **k** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Signal-to-noise (SNR) ratio k.
- **n** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Quantum efficiency of the eye n.
- **N_max** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Maximum number of cycles N_{max} over which the eye can integrate the information.
- **p** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Photon conversion factor p in $photons \div seconds \div degrees^2 \div Trolands$ that depends on the light source.
- **phi_0** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Spectral density ϕ_0 in $seconds degrees^2$ of the neural noise.
- **sigma** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.
- **T** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Integration time T in seconds of the eye.
- **u** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Spatial frequency u, the cycles per degree.
- **u_0** – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Spatial frequency u_0 in $cycles \div degrees$ above which the lateral inhibition ceases.

- **X_0** – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Angular size X_0 in degrees of the object in the x direction.
- **Y_0** – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Angular size Y_0 in degrees of the object in the y direction.
- **X_max** – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Maximum angular size X_{max} in degrees of the integration area in the x direction.
- **Y_max** – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Maximum angular size Y_{max} in degrees of the integration area in the y direction.

Returns Contrast sensitivity S.

Return type numpy.ndarray

References

[Bar99], [Bar03], [CKMW04], [InternationalTUnion15c],

Examples

```
>>> contrast_sensitivity_function(u=4)
360.8691122...
>>> contrast_sensitivity_function("Barten 1999", u=4)
360.8691122...
```

colour.CONTRAST_SENSITIVITY_METHODS

`colour.CONTRAST_SENSITIVITY_METHODS = CanonicalMapping({'Barten 1999': ...})`

Supported contrast sensitivity methods.

References

[Bar99], [Bar03], [CKMW04], [InternationalTUnion15c],

Barten (1999) Contrast Sensitivity Function

`colour.contrast`

contrast_sensitivity_function_Barten1999(u)	Return the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by *Barten (1999)*.

colour.contrast.contrast_sensitivity_function_Barten1999

colour.contrast.**contrast_sensitivity_function_Barten1999**(*u: ArrayLike, sigma: ArrayLike = sigma_Barten1999(0.5 / 60, 0.08 / 60, 2.1), k: ArrayLike = 3.0, T: ArrayLike = 0.1, X_0: ArrayLike = 60, Y_0: ArrayLike | None = None, X_max: ArrayLike = 12, Y_max: ArrayLike | None = None, N_max: ArrayLike = 15, n: ArrayLike = 0.03, p: ArrayLike = 1.2274 * 10 ** 6, E: ArrayLike = retinal_illuminance_Barten1999(20, 2.1), phi_0: ArrayLike = 3 * 10 ** - 8, u_0: ArrayLike = 7*) → NDArrayFloat

Return the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by *Barten (1999)*.

Contrast sensitivity is defined as the inverse of the modulation threshold of a sinusoidal luminance pattern. The modulation threshold of this pattern is generally defined by 50% probability of detection. The contrast sensitivity function or CSF gives the contrast sensitivity as a function of spatial frequency. In the CSF, the spatial frequency is expressed in angular units with respect to the eye. It reaches a maximum between 1 and 10 cycles per degree with a fall off at higher and lower spatial frequencies.

Parameters

- **u** (ArrayLike) – Spatial frequency u, the cycles per degree.
- **sigma** (ArrayLike) – Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.
- **k** (ArrayLike) – Signal-to-noise (SNR) ratio k.
- **T** (ArrayLike) – Integration time T in seconds of the eye.
- **X_0** (ArrayLike) – Angular size X_0 in degrees of the object in the x direction.
- **Y_0** (ArrayLike | None) – Angular size Y_0 in degrees of the object in the y direction.
- **X_max** (ArrayLike) – Maximum angular size X_{max} in degrees of the integration area in the x direction.
- **Y_max** (ArrayLike | None) – Maximum angular size Y_{max} in degrees of the integration area in the y direction.
- **N_max** (ArrayLike) – Maximum number of cycles N_{max} over which the eye can integrate the information.
- **n** (ArrayLike) – Quantum efficiency of the eye n.
- **p** (ArrayLike) – Photon conversion factor p in $photons \div seconds \div degrees^2 \div Trolands$ that depends on the light source.
- **E** (ArrayLike) – Retinal illuminance E in Trolands.
- **phi_0** (ArrayLike) – Spectral density ϕ_0 in $seconds degrees^2$ of the neural noise.
- **u_0** (ArrayLike) – Spatial frequency u_0 in $cycles \div degrees$ above which the lateral inhibition ceases.

Returns Contrast sensitivity S.

Return type numpy.ndarray

Warning: This definition expects σ_0 and C_{ab} used in the computation of σ to be given in degrees and $degrees \div mm$ respectively. However, in the literature, the values for σ_0 and C_{ab} are usually given in $arcmin$ and $arcmin \div mm$ respectively, thus they need to be divided by 60.

Notes

- The formula holds for bilateral viewing and for equal dimensions of the object in x and y direction. For monocular vision, the contrast sensitivity is a factor $\sqrt{2}$ smaller.
- *Barten (1999)* CSF default values for the k, σ_0, C_{ab}, T, X_{max}, N_{max}, n, ϕ_0 and u_0 constants are valid for a standard observer with good vision and with an age between 20 and 30 years.
- The other constants have been filled using reference data from *Figure 31* in [InternationalTUnion15c] but must be adapted to the current use case.
- The product of u, the cycles per degree, and X_0, the number of degrees, gives the number of cycles P_c in a pattern. Therefore, X_0 can be made a variable dependent on u such as $X_0 = P_c/u$.

References

[Bar99], [Bar03], [CKMW04], [InternationalTUnion15c],

Examples

```
>>> contrast_sensitivity_function_Barten1999(4)
360.8691122...
```

Reproducing *Figure 31* in [InternationalTUnion15c] illustrating the minimum detectable contrast according to *Barten (1999)* model with the assumed conditions for UHDTV applications. The minimum detectable contrast MDC is then defined as follows:

```
:math:`MDC = 1 / CSF * 2 * (1 / 1.27)`
```

where 2 is used for the conversion from modulation to contrast and $1/1.27$ is used for the conversion from sinusoidal to rectangular waves.

```
>>> from scipy.optimize import fmin
>>> settings_BT2246 = {
...     "k": 3.0,
...     "T": 0.1,
...     "X_max": 12,
...     "N_max": 15,
...     "n": 0.03,
...     "p": 1.2274 * 10**6,
...     "phi_0": 3 * 10**-8,
...     "u_0": 7,
... }
>>>
>>> def maximise_spatial_frequency(L):
...     maximised_spatial_frequency = []
...     for L_v in L:
...         X_0 = 60
...         d = pupil_diameter_Barten1999(L_v, X_0)
...         sigma = sigma_Barten1999(0.5 / 60, 0.08 / 60, d)
```

(continues on next page)

(continued from previous page)

```
...         E = retinal_illuminance_Barten1999(L_v, d, True)
...         maximised_spatial_frequency.append(
...             fmin(
...                 lambda x: (
...                     -contrast_sensitivity_function_Barten1999(
...                         u=x,
...                         sigma=sigma,
...                         X_0=X_0,
...                         E=E,
...                         **settings_BT2246
...                     )
...                 ),
...                 0,
...                 disp=False,
...             )[0]
...         )
...     return as_float(np.array(maximised_spatial_frequency))
...
>>>
>>> L = np.logspace(np.log10(0.01), np.log10(100), 10)
>>> X_0 = Y_0 = 60
>>> d = pupil_diameter_Barten1999(L, X_0, Y_0)
>>> sigma = sigma_Barten1999(0.5 / 60, 0.08 / 60, d)
>>> E = retinal_illuminance_Barten1999(L, d)
>>> u = maximise_spatial_frequency(L)
>>> (
...     1
...     / contrast_sensitivity_function_Barten1999(
...         u=u, sigma=sigma, E=E, X_0=X_0, Y_0=Y_0, **settings_BT2246
...     )
...     * 2
...     * (1 / 1.27)
... )
...
array([ 0.0218764...,  0.0141848...,  0.0095244...,  0.0066805...,  0.0049246...,
        0.0038228...,  0.0031188...,  0.0026627...,  0.0023674...,  0.0021814...])
```

Ancillary Objects

`colour.contrast`

optical_MTF_Barten1999(u[, sigma])	Return the optical modulation transfer function (MTF) M_{opt} of the eye using *Barten (1999)* method.
pupil_diameter_Barten1999(L[, X_0, Y_0])	Return the pupil diameter for given luminance and object or stimulus angular size using *Barten (1999)* method.
sigma_Barten1999([sigma_0, C_ab, d])	Return the standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process using *Barten (1999)* method.
retinal_illuminance_Barten1999(L[, d, ...])	Return the retinal illuminance E in Trolands for given average luminance L and pupil diameter d using *Barten (1999)* method.
maximum_angular_size_Barten1999(u[, X_0, ...])	Return the maximum angular size X of the object considered using *Barten (1999)* method.

colour.contrast.optical_MTF_Barten1999

colour.contrast.**optical_MTF_Barten1999**(*u: ArrayLike*, *sigma: ArrayLike = 0.01*) → NDArrayFloat

Return the optical modulation transfer function (MTF) M_{opt} of the eye using *Barten (1999)* method.

Parameters

- **u** (ArrayLike) – Spatial frequency u, the cycles per degree.
- **sigma** (ArrayLike) – Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.

Returns Optical modulation transfer function (MTF) M_{opt} of the eye.

Return type numpy.ndarray

References

[Bar99], [Bar03], [CKMW04], [InternationalTUnion15c],

Examples

```
>>> optical_MTF_Barten1999(4, 0.01)
0.9689107...
```

colour.contrast.pupil_diameter_Barten1999

colour.contrast.**pupil_diameter_Barten1999**(*L: ArrayLike*, *X_0: ArrayLike = 60*, *Y_0: ArrayLike | None = None*) → NDArrayFloat

Return the pupil diameter for given luminance and object or stimulus angular size using *Barten (1999)* method.

Parameters

- **L** (ArrayLike) – Average luminance L in cd/m^2.
- **X_0** (ArrayLike) – Angular size of the object X_0 in degrees in the x direction.
- **Y_0** (ArrayLike | None) – Angular size of the object X_0 in degrees in the y direction.

Returns Pupil diameter.

Return type numpy.ndarray

References

[Bar99], [Bar03], [CKMW04], [InternationalTUnion15c], [WY12]

Notes

- The *Log* function is using base 10 as indicated by [WY12].

Examples

```
>>> pupil_diameter_Barten1999(100, 60, 60)
2.7931307...
```

colour.contrast.sigma_Barten1999

colour.contrast.**sigma_Barten1999**(*sigma_0: ArrayLike = 0.5 / 60, C_ab: ArrayLike = 0.08 / 60, d: ArrayLike = 2.1*) → NDArrayFloat

Return the standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process using *Barten (1999)* method.

The σ quantity depends on the pupil diameter d of the eye lens. For very small pupil diameters, σ increases inversely proportionally with pupil size because of diffraction, and for large pupil diameters, σ increases about linearly with pupil size because of chromatic aberration and others aberrations.

Parameters

- **sigma_0** (ArrayLike) – Constant σ_0 in degrees.
- **C_ab** (ArrayLike) – Spherical aberration of the eye C_{ab} in $degrees \div mm$.
- **d** (ArrayLike) – Pupil diameter d in millimeters.

Returns Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.

Return type numpy.ndarray

Warning: This definition expects σ_0 and C_{ab} to be given in degrees and $degrees \div mm$ respectively. However, in the literature, the values for σ_0 and C_{ab} are usually given in $arcmin$ and $arcmin \div mm$ respectively, thus they need to be divided by 60.

References

[Bar99], [Bar03], [CKMW04], [InternationalTUnion15c],

Examples

```
>>> sigma_Barten1999(0.5 / 60, 0.08 / 60, 2.1)
0.0087911...
```

colour.contrast.retinal_illuminance_Barten1999

colour.contrast.**retinal_illuminance_Barten1999**(*L: ArrayLike, d: ArrayLike = 2.1, apply_stiles_crawford_effect_correction: bool = True*) → NDArrayFloat

Return the retinal illuminance E in Trolands for given average luminance L and pupil diameter d using *Barten (1999)* method.

Parameters

- **L** (ArrayLike) – Average luminance L in cd/m^2.
- **d** (ArrayLike) – Pupil diameter d in millimeters.
- **apply_stiles_crawford_effect_correction** (bool) – Whether to apply the correction for *Stiles-Crawford* effect.

Returns Retinal illuminance E in Trolands.

Return type numpy.ndarray

Notes

- This definition is for use with photopic viewing conditions and thus corrects for the Stiles-Crawford effect by default, i.e. directional sensitivity of the cone cells with lower response of cone cells receiving light from the edge of the pupil.

References

[Bar99], [Bar03], [CKMW04], [InternationalTUnion15c],

Examples

```
>>> retinal_illuminance_Barten1999(100, 2.1)
330.4115803...
>>> retinal_illuminance_Barten1999(100, 2.1, False)
346.3605900...
```

colour.contrast.maximum_angular_size_Barten1999

colour.contrast.**maximum_angular_size_Barten1999**(*u: ArrayLike, X_0: ArrayLike = 60, X_max: ArrayLike = 12, N_max: ArrayLike = 15*) → NDArrayFloat

Return the maximum angular size X of the object considered using *Barten (1999)* method.

Parameters

- **u** (ArrayLike) – Spatial frequency u, the cycles per degree.
- **X_0** (ArrayLike) – Angular size X_0 in degrees of the object in the x direction.
- **X_max** (ArrayLike) – Maximum angular size X_{max} in degrees of the integration area in the x direction.
- **N_max** (ArrayLike) – Maximum number of cycles N_{max} over which the eye can integrate the information.

Returns Maximum angular size X of the object considered.

Return type numpy.ndarray

References

[Bar99], [Bar03], [CKMW04], [InternationalTUnion15c],

Examples

```
>>> maximum_angular_size_Barten1999(4)
3.5729480...
```

Continuous Signal

Continuous Signal

colour.continuous

AbstractContinuousFunction([name])	Define the base class for abstract continuous function.
Signal([data, domain])	Define the base class for continuous signal.
MultiSignals([data, domain, labels])	Define the base class for multi-continuous signals, a container for multiple colour.continuous.Signal sub-class instances.

colour.continuous.AbstractContinuousFunction

class colour.continuous.**AbstractContinuousFunction**(*name: str | None = None*)

Bases: abc.ABC, colour.utilities.callback.MixinCallback

Define 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 variable $x \in \mathbb{R}$ returns a corresponding range variable $y \in \mathbb{R}$. 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 property.

Parameters **name** (str | None) – Continuous function name.

Return type None

Attributes

- name
- dtype
- domain
- range
- interpolator
- interpolator_kwargs
- extrapolator

- `extrapolator_kwargs`
- `function`

Methods

- `__init__()`
- `__str__()`
- `__repr__()`
- `__hash__()`
- `__getitem__()`
- `__setitem__()`
- `__contains__()`
- `__iter__()`
- `__len__()`
- `__eq__()`
- `__ne__()`
- `__iadd__()`
- `__add__()`
- `__isub__()`
- `__sub__()`
- `__imul__()`
- `__mul__()`
- `__idiv__()`
- `__div__()`
- `__ipow__()`
- `__pow__()`
- `arithmetical_operation()`
- `fill_nan()`
- `domain_distance()`
- `is_uniform()`
- `copy()`

`__init__`(*name: str | None = None*) → None

Parameters **`name`** (`str | None`) –

Return type None

`property name: str`

Getter and setter property for the abstract continuous function name.

Parameters **`value`** – Value to set the abstract continuous function name with.

Returns Abstract continuous function name.

Return type `str`

abstract property dtype: Type[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cec4023d0>]

Getter and setter property for the abstract continuous function dtype, must be reimplemented by sub-classes.

Parameters **value** – Value to set the abstract continuous function dtype with.

Returns Abstract continuous function dtype.

Return type Type[DTypeFloat]

abstract property domain: NDArrayFloat

Getter and setter property for the abstract continuous function independent domain variable x, must be reimplemented by sub-classes.

Parameters **value** – Value to set the abstract continuous function independent domain variable x with.

Returns Abstract continuous function independent domain variable x.

Return type numpy.ndarray

abstract property range: NDArrayFloat

Getter and setter property for the abstract continuous function corresponding range variable y, must be reimplemented by sub-classes.

Parameters **value** – Value to set the abstract continuous function corresponding range variable y with.

Returns Abstract continuous function corresponding range variable y.

Return type numpy.ndarray

abstract property interpolator: Type[colour.hints.ProtocolInterpolator]

Getter and setter property for the abstract continuous function interpolator type, must be reimplemented by sub-classes.

Parameters **value** – Value to set the abstract continuous function interpolator type with.

Returns Abstract continuous function interpolator type.

Return type Type[*ProtocolInterpolator*]

abstract property interpolator_kwargs: dict

Getter and setter property for the abstract continuous function interpolator instantiation time arguments, must be reimplemented by sub-classes.

Parameters **value** – Value to set the abstract continuous function interpolator instantiation time arguments to.

Returns Abstract continuous function interpolator instantiation time arguments.

Return type dict

abstract property extrapolator: Type[colour.hints.ProtocolExtrapolator]

Getter and setter property for the abstract continuous function extrapolator type, must be reimplemented by sub-classes.

Parameters **value** – Value to set the abstract continuous function extrapolator type with.

Returns Abstract continuous function extrapolator type.

Return type Type[*ProtocolExtrapolator*]

abstract property extrapolator_kwargs: dict

Getter and setter property for the abstract continuous function extrapolator instantiation time arguments, must be reimplemented by sub-classes.

Parameters value – Value to set the abstract continuous function extrapolator instantiation time arguments to.

Returns Abstract continuous function extrapolator instantiation time arguments.

Return type dict

abstract property function: Callable

Getter property for the abstract continuous function callable, must be reimplemented by sub-classes.

Returns Abstract continuous function callable.

Return type Callable

abstract __str__() → str

Return a formatted string representation of the abstract continuous function, must be reimplemented by sub-classes.

Returns Formatted string representation.

Return type str

abstract __repr__() → str

Return an evaluable string representation of the abstract continuous function, must be reimplemented by sub-classes.

Returns Evaluable string representation.

Return type str

abstract __hash__() → int

Return the abstract continuous function hash.

Returns Object hash.

Return type int

abstract __getitem__(*x: ArrayLike* | *slice*) → NDArrayFloat

Return the corresponding range variable y for independent domain variable x, must be reimplemented by sub-classes.

Parameters x (ArrayLike | slice) – Independent domain variable x.

Returns Variable y range value.

Return type numpy.ndarray

abstract __setitem__(*x: ArrayLike* | *slice*, *y: ArrayLike*)

Set the corresponding range variable y for independent domain variable x, must be reimplemented by sub-classes.

Parameters

- **x** (ArrayLike | slice) – Independent domain variable x.
- **y** (ArrayLike) – Corresponding range variable y.

abstract __contains__(*x: ArrayLike* | *slice*) → bool

Return whether the abstract continuous function contains given independent domain variable x, must be reimplemented by sub-classes.

Parameters x (ArrayLike | slice) – Independent domain variable x.

Returns Whether x domain value is contained.

Return type bool

__iter__() → *collections.abc.Generator*

Return a generator for the abstract continuous function.

Yields *Generator* – Abstract continuous function generator.

Return type *collections.abc.Generator*

__len__() → int

Return the abstract continuous function independent domain x variable elements count.

Returns Independent domain variable x elements count.

Return type int

abstract __eq__(*other: Any*) → bool

Return whether the abstract continuous function is equal to given other object, must be reimplemented by sub-classes.

Parameters other (Any) – Object to test whether it is equal to the abstract continuous function.

Returns Whether given object is equal to the abstract continuous function.

Return type bool

abstract __ne__(*other: Any*) → bool

Return whether the abstract continuous function is not equal to given other object, must be reimplemented by sub-classes.

Parameters other (Any) – Object to test whether it is not equal to the abstract continuous function.

Returns Whether given object is not equal to the abstract continuous function.

Return type bool

__add__(*a: ArrayLike | Self*) → Self

Implement support for addition.

Parameters a (ArrayLike | Self) – Variable a to add.

Returns Variable added abstract continuous function.

Return type colour.continuous.AbstractContinuousFunction

__iadd__(*a: ArrayLike | Self*) → Self

Implement support for in-place addition.

Parameters a (ArrayLike | Self) – Variable a to add in-place.

Returns In-place variable added abstract continuous function.

Return type colour.continuous.AbstractContinuousFunction

__sub__(*a: ArrayLike | Self*) → Self

Implement support for subtraction.

Parameters a (ArrayLike | Self) – Variable a to subtract.

Returns Variable subtracted abstract continuous function.

Return type colour.continuous.AbstractContinuousFunction

__isub__(*a: ArrayLike | Self*) → Self

Implement support for in-place subtraction.

Parameters a (ArrayLike | Self) – Variable a to subtract in-place.

Returns In-place variable subtracted abstract continuous function.

Return type `colour.continuous.AbstractContinuousFunction`

`__mul__`(*a: ArrayLike | Self*) → Self
Implement support for multiplication.

Parameters **a** (`ArrayLike | Self`) – Variable a to multiply by.

Returns Variable multiplied abstract continuous function.

Return type `colour.continuous.AbstractContinuousFunction`

`__imul__`(*a: ArrayLike | Self*) → Self
Implement support for in-place multiplication.

Parameters **a** (`ArrayLike | Self`) – Variable a to multiply by in-place.

Returns In-place variable multiplied abstract continuous function.

Return type `colour.continuous.AbstractContinuousFunction`

`__div__`(*a: ArrayLike | Self*) → Self
Implement support for division.

Parameters **a** (`ArrayLike | Self`) – Variable a to divide by.

Returns Variable divided abstract continuous function.

Return type `colour.continuous.AbstractContinuousFunction`

`__idiv__`(*a: ArrayLike | Self*) → Self
Implement support for in-place division.

Parameters **a** (`ArrayLike | Self`) – Variable a to divide by in-place.

Returns In-place variable divided abstract continuous function.

Return type `colour.continuous.AbstractContinuousFunction`

`__itruediv__`(*a: ArrayLike | Self*) → Self
Implement support for in-place division.

Parameters **a** (`ArrayLike | Self`) – Variable a to divide by in-place.

Returns In-place variable divided abstract continuous function.

Return type `colour.continuous.AbstractContinuousFunction`

`__weakref__`
list of weak references to the object (if defined)

`__truediv__`(*a: ArrayLike | Self*) → Self
Implement support for division.

Parameters **a** (`ArrayLike | Self`) – Variable a to divide by.

Returns Variable divided abstract continuous function.

Return type `colour.continuous.AbstractContinuousFunction`

`__pow__`(*a: ArrayLike | Self*) → Self
Implement support for exponentiation.

Parameters **a** (`ArrayLike | Self`) – Variable a to exponentiate by.

Returns Variable exponentiated abstract continuous function.

Return type `colour.continuous.AbstractContinuousFunction`

__ipow__(*a: ArrayLike | Self*) → Self

Implement support for in-place exponentiation.

Parameters **a** (ArrayLike | Self) – Variable a to exponentiate by in-place.

Returns In-place variable exponentiated abstract continuous function.

Return type colour.continuous.AbstractContinuousFunction

abstract arithmetical_operation(*a: ArrayLike | Self, operation: Literal['+', '-', '*', '/', '**'], in_place: bool = False*) → Self

Perform given arithmetical operation with operand a, the operation can be either performed on a copy or in-place, must be reimplemented by sub-classes.

Parameters

- **a** (ArrayLike | Self) – Operand a.
- **operation** (Literal['+', '-', '*', '/', '**']) – Operation to perform.
- **in_place** (bool) – Operation happens in place.

Returns Abstract continuous function.

Return type colour.continuous.AbstractContinuousFunction

abstract fill_nan(*method: Union[Literal['Interpolation', 'Constant'], str] = 'Interpolation', default: Real = 0*) → Self

Fill NaNs in independent domain variable x and corresponding range variable y using given method, must be reimplemented by sub-classes.

Parameters

- **method** (Union[Literal['Interpolation', 'Constant'], str]) – *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with default.
- **default** (Real) – Value to use with the *Constant* method.

Returns NaNs filled abstract continuous function.

Return type colour.continuous.AbstractContinuousFunction

domain_distance(*a: ArrayLike*) → NDArrayFloat

Return the euclidean distance between given array and independent domain x closest element.

Parameters **a** (ArrayLike) – Variable a to compute the euclidean distance with independent domain variable x.

Returns Euclidean distance between independent domain variable x and given variable a.

Return type numpy.ndarray

is_uniform() → bool

Return if independent domain variable x is uniform.

Returns Is independent domain variable x uniform.

Return type bool

copy() → Self

Return a copy of the sub-class instance.

Returns Abstract continuous function copy.

Return type colour.continuous.AbstractContinuousFunction

colour.continuous.Signal

class colour.continuous.**Signal**(*data: ArrayLike | dict | Self | Series | None = None, domain: ArrayLike | None = None, **kwargs: Any*)

Bases: `colour.continuous.abstract.AbstractContinuousFunction`

Define the base class for continuous signal.

The class implements the `Signal.function()` method so that evaluating the function for any independent domain variable $x \in \mathbb{R}$ returns a corresponding range variable $y \in \mathbb{R}$. 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` property corresponds with the function dependent and already known range stored in the `colour.continuous.Signal.range` property.

Important: Specific documentation about getting, setting, indexing and slicing the continuous signal values is available in the *Spectral Representation and Continuous Signal* section.

Parameters

- **data** (ArrayLike | dict | Self | Series | None) – Data to be stored in the continuous signal.
- **domain** (ArrayLike | None) – 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` property.
- **dtype** – float point data type.
- **extrapolator** – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function.
- **interpolator** – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function.
- **name** – Continuous signal name.
- **kwargs** (Any) –

Return type None

Attributes

- `dtype`
- `domain`
- `range`
- `interpolator`
- `interpolator_kwargs`
- `extrapolator`
- `extrapolator_kwargs`
- `function`

Methods

- `__init__()`
- `__str__()`
- `__repr__()`
- `__hash__()`
- `__getitem__()`
- `__setitem__()`
- `__contains__()`
- `__eq__()`
- `__ne__()`
- `arithmetical_operation()`
- `signal_unpack_data()`
- `fill_nan()`
- `to_series()`

Examples

Instantiation with implicit *domain*:

```
>>> range_ = np.linspace(10, 100, 10)
>>> print(Signal(range_))
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
```

Instantiation with explicit *domain*:

```
>>> domain = np.arange(100, 1100, 100)
>>> print(Signal(range_, domain))
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Instantiation with a *dict*:

```
>>> print(Signal(dict(zip(domain, range_))))
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Instantiation with a *Pandas* 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: ArrayLike | dict | Self | Series | None = None*, *domain: ArrayLike | None = None*, ***kwargs: Any*) → None

Parameters

- **data** (ArrayLike | dict | Self | Series | None) –
- **domain** (ArrayLike | None) –
- **kwargs** (Any) –

Return type None

property dtype: Type[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cecd18190>]

Getter and setter property for the continuous signal dtype.

Parameters **value** – Value to set the continuous signal dtype with.

Returns Continuous signal dtype.

Return type DTypeFloat

property domain: NDArrayFloat

Getter and setter property for the continuous signal independent domain variable x.

Parameters **value** – Value to set the continuous signal independent domain variable x with.

Returns Continuous signal independent domain variable x.

Return type numpy.ndarray

property range: NDArrayFloat

Getter and setter property for the continuous signal corresponding range variable y.

Parameters **value** – Value to set the continuous signal corresponding range y variable with.

Returns Continuous signal corresponding range variable y.

Return type numpy.ndarray

property interpolator: Type[colour.hints.ProtocolInterpolator]

Getter and setter property for the continuous signal interpolator type.

Parameters **value** – Value to set the continuous signal interpolator type with.

Returns Continuous signal interpolator type.

Return type Type[*ProtocolInterpolator*]

property interpolator_kwargs: dict

Getter and setter property for the continuous signal interpolator instantiation time arguments.

Parameters **value** – Value to set the continuous signal interpolator instantiation time arguments to.

Returns Continuous signal interpolator instantiation time arguments.

Return type dict

property extrapolator: Type[colour.hints.ProtocolExtrapolator]

Getter and setter property for the continuous signal extrapolator type.

Parameters **value** – Value to set the continuous signal extrapolator type with.

Returns Continuous signal extrapolator type.

Return type Type[*ProtocolExtrapolator*]

property extrapolator_kwargs: dict

Getter and setter property for the continuous signal extrapolator instantiation time arguments.

Parameters **value** – Value to set the continuous signal extrapolator instantiation time arguments to.

Returns Continuous signal extrapolator instantiation time arguments.

Return type dict

property function: Callable

Getter property for the continuous signal callable.

Returns Continuous signal callable.

Return type Callable

__str__() → str

Return a formatted string representation of the continuous signal.

Returns Formatted string representation.

Return type str

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> print(Signal(range_))
[[   0.    10.]
 [   1.    20.]
 [   2.    30.]
 [   3.    40.]
 [   4.    50.]
 [   5.    60.]
 [   6.    70.]
 [   7.    80.]
 [   8.    90.]
 [   9.   100.]]
```

__repr__() → str

Return an evaluable string representation of the continuous signal.

Returns Evaluable string representation.

Return type str

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> Signal(range_)
Signal([[   0.,    10.],
        [   1.,    20.],
        [   2.,    30.],
        [   3.,    40.],
        [   4.,    50.],
        [   5.,    60.],
        [   6.,    70.],
        [   7.,    80.],
        [   8.,    90.],
        [   9.,   100.]],
       KernelInterpolator,
       {},
       Extrapolator,
       {'method': 'Constant', 'left': nan, 'right': nan})
```

__hash__() → int

Return the abstract continuous function hash.

Returns Object hash.

Return type `int`

__getitem__(*x: ArrayLike* | *slice*) → NDArrayFloat

Return the corresponding range variable y for independent domain variable x.

Parameters **x** (`ArrayLike | slice`) – Independent domain variable x.

Returns Variable y range value.

Return type `numpy.ndarray`

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal = Signal(range_)
>>> print(signal)
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> signal[0]
10.0
>>> signal[np.array([0, 1, 2])]
array([ 10.,  20.,  30.])
>>> signal[0:3]
array([ 10.,  20.,  30.])
>>> signal[np.linspace(0, 5, 5)]
array([ 10.        ,  22.8348902...,  34.8004492...,  47.5535392...,  60.        
↪])
```

__setitem__(*x: ArrayLike* | *slice*, *y: ArrayLike*)

Set the corresponding range variable y for independent domain variable x.

Parameters

- **x** (`ArrayLike | slice`) – Independent domain variable x.
- **y** (`ArrayLike`) – Corresponding range variable y.

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal = Signal(range_)
>>> print(signal)
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> signal[0] = 20
>>> signal[0]
20.0
>>> signal[np.array([0, 1, 2])] = 30
>>> signal[np.array([0, 1, 2])]
array([ 30.,  30.,  30.])
>>> signal[0:3] = 40
>>> signal[0:3]
array([ 40.,  40.,  40.])
>>> signal[np.linspace(0, 5, 5)] = 50
>>> print(signal)
[[   0.      50.  ]
 [   1.      40.  ]
 [   1.25    50.  ]
 [   2.      40.  ]
 [   2.5     50.  ]
 [   3.      40.  ]
 [   3.75    50.  ]
 [   4.      50.  ]
 [   5.      50.  ]
 [   6.      70.  ]
 [   7.      80.  ]
 [   8.      90.  ]
 [   9.     100.  ]]
>>> signal[np.array([0, 1, 2])] = np.array([10, 20, 30])
>>> print(signal)
[[   0.      10.  ]
 [   1.      20.  ]
 [   1.25    50.  ]
 [   2.      30.  ]
 [   2.5     50.  ]
 [   3.      40.  ]
 [   3.75    50.  ]
 [   4.      50.  ]
 [   5.      50.  ]
 [   6.      70.  ]
 [   7.      80.  ]
 [   8.      90.  ]
 [   9.     100.  ]]
```

__contains__(*x: ArrayLike*) → bool

Return whether the continuous signal contains given independent domain variable x.

Parameters **x** (ArrayLike) – Independent domain variable x.

Returns Whether x domain value is contained.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal = Signal(range_)
>>> 0 in signal
True
>>> 0.5 in signal
True
>>> 1000 in signal
False
```

__eq__(*other: Any*) → bool

Return whether the continuous signal is equal to given other object.

Parameters **other** (Any) – Object to test whether it is equal to the continuous signal.

Returns Whether given object is equal to the continuous signal.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal_1 = Signal(range_)
>>> signal_2 = Signal(range_)
>>> signal_1 == signal_2
True
>>> signal_2[0] = 20
>>> signal_1 == signal_2
False
>>> signal_2[0] = 10
>>> signal_1 == signal_2
True
>>> from colour.algebra import CubicSplineInterpolator
>>> signal_2.interpolator = CubicSplineInterpolator
>>> signal_1 == signal_2
False
```

__ne__(*other: Any*) → bool

Return whether the continuous signal is not equal to given other object.

Parameters **other** (Any) – Object to test whether it is not equal to the continuous signal.

Returns Whether given object is not equal to the continuous signal.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal_1 = Signal(range_)
>>> signal_2 = Signal(range_)
>>> signal_1 != signal_2
False
>>> signal_2[0] = 20
>>> signal_1 != signal_2
True
>>> signal_2[0] = 10
>>> signal_1 != signal_2
False
>>> from colour.algebra import CubicSplineInterpolator
>>> signal_2.interpolator = CubicSplineInterpolator
>>> signal_1 != signal_2
True
```

arithmetical_operation(*a: ArrayLike* | AbstractContinuousFunction, *operation: Literal['+', '-', '*', '/', '**']*, *in_place: bool = False*) → *AbstractContinuousFunction*

Perform given arithmetical operation with operand a, the operation can be either performed on a copy or in-place.

Parameters

- **a** (ArrayLike | AbstractContinuousFunction) – Operand a.
- **operation** (Literal['+', '-', '*', '/', '**']) – Operation to perform.
- **in_place** (bool) – Operation happens in place.

Returns Continuous signal.

Return type colour.continuous.Signal

Examples

Adding a single *numeric* variable:

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal_1 = Signal(range_)
>>> print(signal_1)
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> print(signal_1.arithmetical_operation(10, "+", True))
[[   0.   20.]
 [   1.   30.]
 [   2.   40.]
 [   3.   50.]
 [   4.   60.]
 [   5.   70.]
```

(continues on next page)

(continued from previous page)

```
[   6.   80.]
[   7.   90.]
[   8.  100.]
[   9.  110.]]
```

Adding an *ArrayLike* variable:

```
>>> a = np.linspace(10, 100, 10)
>>> print(signal_1.arithmetical_operation(a, "+", True))
[[   0.   30.]
 [   1.   50.]
 [   2.   70.]
 [   3.   90.]
 [   4.  110.]
 [   5.  130.]
 [   6.  150.]
 [   7.  170.]
 [   8.  190.]
 [   9.  210.]]
```

Adding a `colour.continuous.Signal` class:

```
>>> signal_2 = Signal(range_)
>>> print(signal_1.arithmetical_operation(signal_2, "+", True))
[[   0.   40.]
 [   1.   70.]
 [   2.  100.]
 [   3.  130.]
 [   4.  160.]
 [   5.  190.]
 [   6.  220.]
 [   7.  250.]
 [   8.  280.]
 [   9.  310.]]
```

static signal_unpack_data(*data=Optional[Union[ArrayLike, dict, Series, 'Signal']], domain: ArrayLike | None = None, dtype: Type[DTypeFloat] | None = None*) → tuple

Unpack given data for continuous signal instantiation.

Parameters

- **data** – Data to unpack for continuous signal instantiation.
- **domain** (ArrayLike | None) – Values to initialise the `colour.continuous.Signal.domain` attribute with. If both `data` and `domain` arguments are defined, the latter will be used to initialise the `colour.continuous.Signal.domain` property.
- **dtype** (Type[DTypeFloat] | None) – float point data type.

Returns Independent domain variable x and corresponding range variable y unpacked for continuous signal instantiation.

Return type `tuple`

Examples

Unpacking using implicit *domain*:

```
>>> range_ = np.linspace(10, 100, 10)
>>> domain, range_ = Signal.signal_unpack_data(range_)
>>> print(domain)
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using explicit *domain*:

```
>>> domain = np.arange(100, 1100, 100)
>>> domain, range = Signal.signal_unpack_data(range_, domain)
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using a *dict*:

```
>>> domain, range_ = Signal.signal_unpack_data(
...     dict(zip(domain, range_))
... )
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using a *Pandas* `pandas.Series`:

```
>>> if is_pandas_installed():
...     from pandas import Series
...
...     domain, range = Signal.signal_unpack_data(
...         Series(dict(zip(domain, range_)))
...     )
...
...
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using a `colour.continuous.Signal` class:

```
>>> domain, range_ = Signal.signal_unpack_data(Signal(range_, domain))
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

fill_nan(*method: Union[Literal['Interpolation', 'Constant'], str] = 'Interpolation', default: Real = 0*) → *colour.continuous.abstract.AbstractContinuousFunction*

Fill NaNs in independent domain variable x and corresponding range variable y using given method.

Parameters

- **method** (Union[Literal['Interpolation', 'Constant'], str]) – *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with default.
- **default** (Real) – Value to use with the *Constant* method.

Returns NaNs filled continuous signal.

Return type colour.continuous.Signal

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal = Signal(range_)
>>> signal[3:7] = np.nan
>>> print(signal)
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   nan]
 [   4.   nan]
 [   5.   nan]
 [   6.   nan]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> print(signal.fill_nan())
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> signal[3:7] = np.nan
>>> print(signal.fill_nan(method="Constant"))
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.    0.]
 [   4.    0.]
 [   5.    0.]
 [   6.    0.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
```

to_series() → pandas.core.series.Series

Convert the continuous signal to a *Pandas* pandas.Series class instance.

Returns Continuous signal as a *Pandas*pandas.Series class instance.

Return type pandas.Series

Examples

```
>>> if is_pandas_installed():
...     range_ = np.linspace(10, 100, 10)
...     signal = Signal(range_)
...     print(signal.to_series())
...
0.0     10.0
1.0     20.0
2.0     30.0
3.0     40.0
4.0     50.0
5.0     60.0
6.0     70.0
7.0     80.0
8.0     90.0
9.0    100.0
Name: Signal (...), dtype: float64
```

colour.continuous.MultiSignals

class colour.continuous.**MultiSignals**(*data: ArrayLike | DataFrame | dict | Self | Sequence | Series | Signal | None = None, domain: ArrayLike | None = None, labels: Sequence | None = None, **kwargs: Any*)

Bases: `colour.continuous.abstract.AbstractContinuousFunction`

Define the base class for multi-continuous signals, a container for multiple `colour.continuous.Signal` sub-class instances.

Important: Specific documentation about getting, setting, indexing and slicing the multi-continuous signals values is available in the *Spectral Representation and Continuous Signal* section.

Parameters

- **data** (ArrayLike | DataFrame | dict | Self | Sequence | Series | Signal | None) – Data to be stored in the multi-continuous signals.
- **domain** (ArrayLike | None) – 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** (Sequence | None) – Names to use for the `colour.continuous.Signal` sub-class instances.
- **dtype** – float point data type.
- **extrapolator** – Extrapolator class type to use as extrapolating function for the `colour.continuous.Signal` sub-class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the `colour.continuous.Signal` sub-class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the `colour.continuous.Signal` sub-class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the `colour.continuous.Signal` sub-class instances.

- **name** – multi-continuous signals name.
- **signal_type** – The `colour.continuous.Signal` sub-class type used for instances.
- **kwargs** (`Any`) –

Return type None

Attributes

- `dtype`
- `domain`
- `range`
- `interpolator`
- `interpolator_kwargs`
- `extrapolator`
- `extrapolator_kwargs`
- `function`
- `signals`
- `labels`
- `signal_type`

Methods

- `__init__()`
- `__str__()`
- `__repr__()`
- `__hash__()`
- `__getitem__()`
- `__setitem__()`
- `__contains__()`
- `__eq__()`
- `__ne__()`
- `arithmetical_operation()`
- `multi_signals_unpack_data()`
- `fill_nan()`
- `to_dataframe()`

Examples

Instantiation with implicit *domain* and a single signal:

```
>>> range_ = np.linspace(10, 100, 10)
>>> print(MultiSignals(range_))
[[   0.    10.]
 [   1.    20.]
 [   2.    30.]
 [   3.    40.]
 [   4.    50.]
 [   5.    60.]
 [   6.    70.]
 [   7.    80.]
 [   8.    90.]
 [   9.   100.]]
```

Instantiation with explicit *domain* and a single signal:

```
>>> domain = np.arange(100, 1100, 100)
>>> print(MultiSignals(range_, domain))
[[  100.     10.]
 [  200.     20.]
 [  300.     30.]
 [  400.     40.]
 [  500.     50.]
 [  600.     60.]
 [  700.     70.]
 [  800.     80.]
 [  900.     90.]
 [ 1000.    100.]]
```

Instantiation with multiple signals:

```
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> print(MultiSignals(range_, domain))
[[  100.     10.     20.     30.]
 [  200.     20.     30.     40.]
 [  300.     30.     40.     50.]
 [  400.     40.     50.     60.]
 [  500.     50.     60.     70.]
 [  600.     60.     70.     80.]
 [  700.     70.     80.     90.]
 [  800.     80.     90.    100.]
 [  900.     90.    100.    110.]
 [ 1000.    100.    110.    120.]]
```

Instantiation with a *dict*:

```
>>> print(MultiSignals(dict(zip(domain, range_))))
[[  100.     10.     20.     30.]
 [  200.     20.     30.     40.]
 [  300.     30.     40.     50.]
 [  400.     40.     50.     60.]
 [  500.     50.     60.     70.]
 [  600.     60.     70.     80.]
 [  700.     70.     80.     90.]
```

(continues on next page)

(continued from previous page)

```
[  800.     80.     90.    100.]
[  900.     90.    100.    110.]
[ 1000.    100.    110.    120.]]
```

Instantiation using a *Signal* sub-class:

```
>>> class NotSignal(Signal):
...     pass
...
```

```
>>> multi_signals = MultiSignals(range_, domain, signal_type=NotSignal)
>>> print(multi_signals)
[[  100.     10.     20.     30.]
 [  200.     20.     30.     40.]
 [  300.     30.     40.     50.]
 [  400.     40.     50.     60.]
 [  500.     50.     60.     70.]
 [  600.     60.     70.     80.]
 [  700.     70.     80.     90.]
 [  800.     80.     90.    100.]
 [  900.     90.    100.    110.]
 [ 1000.    100.    110.    120.]]
 >>> type(multi_signals.signals[0])
 <class 'multi_signals.NotSignal'>
```

Instantiation with a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...
...     print(
...         MultiSignals(
...             Series(dict(zip(domain, np.linspace(10, 100, 10))))
...         )
...     )
...
[[  100.     10.]
 [  200.     20.]
 [  300.     30.]
 [  400.     40.]
 [  500.     50.]
 [  600.     60.]
 [  700.     70.]
 [  800.     80.]
 [  900.     90.]
 [ 1000.    100.]]
```

Instantiation with a *Pandas* `pandas.DataFrame`:

```
>>> if is_pandas_installed():
...     from pandas import DataFrame
...
...     data = dict(zip(["a", "b", "c"], tsplit(range_)))
...     print(MultiSignals(DataFrame(data, domain)))
...
[[  100.     10.     20.     30.]
```

(continues on next page)

(continued from previous page)

```
[  200.    20.    30.    40.]
[  300.    30.    40.    50.]
[  400.    40.    50.    60.]
[  500.    50.    60.    70.]
[  600.    60.    70.    80.]
[  700.    70.    80.    90.]
[  800.    80.    90.   100.]
[  900.    90.   100.   110.]
[ 1000.   100.   110.   120.]]
```

Retrieving domain *y* variable for arbitrary range *x* variable:

```
>>> x = 150
>>> range_ = tstack([np.sin(np.linspace(0, 1, 10))] * 3)
>>> range_ += np.array([0.0, 0.25, 0.5])
>>> MultiSignals(range_, domain)[x]
array([ 0.0359701...,  0.2845447...,  0.5331193...])
>>> x = np.linspace(100, 1000, 3)
>>> MultiSignals(range_, domain)[x]
array([[  4.4085384...e-20,   2.5000000...e-01,   5.0000000...e-01],
       [  4.7669395...e-01,   7.2526859...e-01,   9.7384323...e-01],
       [  8.4147098...e-01,   1.0914709...e+00,   1.3414709...e+00]])
```

Using an alternative interpolating function:

```
>>> x = 150
>>> from colour.algebra import CubicSplineInterpolator
>>> MultiSignals(range_, domain, interpolator=CubicSplineInterpolator)[
...     x
... ]
array([ 0.0555274...,  0.3055274...,  0.5555274...])
>>> x = np.linspace(100, 1000, 3)
>>> MultiSignals(range_, domain, interpolator=CubicSplineInterpolator)[
...     x
... ]
array([[ 0.        ...,  0.25      ...,  0.5       ...],
       [ 0.4794253...,  0.7294253...,  0.9794253...],
       [ 0.8414709...,  1.0914709...,  1.3414709...]])
```

__init__(*data: ArrayLike | DataFrame | dict | Self | Sequence | Series | Signal | None = None, domain: ArrayLike | None = None, labels: Sequence | None = None, **kwargs: Any*) → None

Parameters

- **data** (ArrayLike | DataFrame | dict | Self | Sequence | Series | Signal | None) –
- **domain** (ArrayLike | None) –
- **labels** (Sequence | None) –
- **kwargs** (Any) –

Return type None

property dtype: Type[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6cec3ffcd0>]

Getter and setter property for the continuous signal dtype.

Parameters **value** – Value to set the continuous signal dtype with.

Returns Continuous signal dtype.

Return type Type[DTypeFloat]

property domain: NDArrayFloat

Getter and setter property for the colour.continuous.Signal sub-class instances independent domain variable x.

Parameters value – Value to set the colour.continuous.Signal sub-class instances independent domain variable x with.

Returns colour.continuous.Signal sub-class instances independent domain variable x.

Return type numpy.ndarray

property range: NDArrayFloat

Getter and setter property for the colour.continuous.Signal sub-class instances corresponding range variable y.

Parameters value – Value to set the colour.continuous.Signal sub-class instances corresponding range variable y with.

Returns colour.continuous.Signal sub-class instances corresponding range variable y.

Return type numpy.ndarray

property interpolator: Type[colour.hints.ProtocolInterpolator]

Getter and setter property for the colour.continuous.Signal sub-class instances interpolator type.

Parameters value – Value to set the colour.continuous.Signal sub-class instances interpolator type with.

Returns colour.continuous.Signal sub-class instances interpolator type.

Return type Type[*ProtocolInterpolator*]

property interpolator_kwargs: dict

Getter and setter property for the colour.continuous.Signal sub-class instances interpolator instantiation time arguments.

Parameters value – Value to set the colour.continuous.Signal sub-class instances interpolator instantiation time arguments to.

Returns colour.continuous.Signal sub-class instances interpolator instantiation time arguments.

Return type dict

property extrapolator: Type[colour.hints.ProtocolExtrapolator]

Getter and setter property for the colour.continuous.Signal sub-class instances extrapolator type.

Parameters value – Value to set the colour.continuous.Signal sub-class instances extrapolator type with.

Returns colour.continuous.Signal sub-class instances extrapolator type.

Return type Type[*ProtocolExtrapolator*]

property extrapolator_kwargs: dict

Getter and setter property for the colour.continuous.Signal sub-class instances extrapolator instantiation time arguments.

Parameters value – Value to set the colour.continuous.Signal sub-class instances extrapolator instantiation time arguments to.

Returns colour.continuous.Signal sub-class instances extrapolator instantiation time arguments.

Return type dict

property function: Callable

Getter property for the colour.continuous.Signal sub-class instances callable.

Returns colour.continuous.Signal sub-class instances callable.

Return type Callable

property signals: Dict[str, colour.continuous.signal.Signal]

Getter and setter property for the colour.continuous.Signal sub-class instances.

Parameters value – Attribute value.

Returns colour.continuous.Signal sub-class instances.

Return type dict

property labels: List[str]

Getter and setter property for the colour.continuous.Signal sub-class instance names.

Parameters value – Value to set the colour.continuous.Signal sub-class instance names.

Returns colour.continuous.Signal sub-class instance names.

Return type list

property signal_type: Type[colour.continuous.signal.Signal]

Getter property for the colour.continuous.Signal sub-class instances type.

Returns colour.continuous.Signal sub-class instances type.

Return type Type[*Signal*]

__str__() → str

Return a formatted string representation of the multi-continuous signals.

Returns Formatted string representation.

Return type str

Examples

```
>>> domain = np.arange(0, 10, 1)
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> print(MultiSignals(range_))
[[   0.    10.    20.    30.]
 [   1.    20.    30.    40.]
 [   2.    30.    40.    50.]
 [   3.    40.    50.    60.]
 [   4.    50.    60.    70.]
 [   5.    60.    70.    80.]
 [   6.    70.    80.    90.]
 [   7.    80.    90.   100.]
 [   8.    90.   100.   110.]
 [   9.   100.   110.   120.]]
```

__repr__() → str

Return an evaluable string representation of the multi-continuous signals.

Returns Evaluable string representation.

Return type str

Examples

```
>>> domain = np.arange(0, 10, 1)
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> MultiSignals(range_)
MultiSignals([[   0.,   10.,   20.,   30.],
              [   1.,   20.,   30.,   40.],
              [   2.,   30.,   40.,   50.],
              [   3.,   40.,   50.,   60.],
              [   4.,   50.,   60.,   70.],
              [   5.,   60.,   70.,   80.],
              [   6.,   70.,   80.,   90.],
              [   7.,   80.,   90.,  100.],
              [   8.,   90.,  100.,  110.],
              [   9.,  100.,  110.,  120.]],
             ['0', '1', '2'],
             KernelInterpolator,
             {},
             Extrapolator,
             {'method': 'Constant', 'left': nan, 'right': nan})
```

__hash__() → int

Return the abstract continuous function hash.

Returns Object hash.

Return type int

__getitem__(*x: ArrayLike | slice*) → NDArrayFloat

Return the corresponding range variable y for independent domain variable x.

Parameters x (ArrayLike | slice) – Independent domain variable x.

Returns Variable y range value.

Return type numpy.ndarray

Examples

```
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> multi_signals = MultiSignals(range_)
>>> print(multi_signals)
[[   0.   10.   20.   30.]
 [   1.   20.   30.   40.]
 [   2.   30.   40.   50.]
 [   3.   40.   50.   60.]
 [   4.   50.   60.   70.]
 [   5.   60.   70.   80.]
 [   6.   70.   80.   90.]
 [   7.   80.   90.  100.]
```

(continues on next page)

(continued from previous page)

```
 [   8.   90.  100.  110.]
 [   9.  100.  110.  120.]]
>>> multi_signals[0]
array([ 10.,  20.,  30.])
>>> multi_signals[np.array([0, 1, 2])]
array([[ 10.,  20.,  30.],
       [ 20.,  30.,  40.],
       [ 30.,  40.,  50.]])
>>> multi_signals[np.linspace(0, 5, 5)]
array([[ 10.        ...,  20.        ...,  30.        ...],
       [ 22.8348902...,  32.8046056...,  42.774321 ...],
       [ 34.8004492...,  44.7434347...,  54.6864201...],
       [ 47.5535392...,  57.5232546...,  67.4929700...],
       [ 60.        ...,  70.        ...,  80.        ...]])
>>> multi_signals[0:3]
array([[ 10.,  20.,  30.],
       [ 20.,  30.,  40.],
       [ 30.,  40.,  50.]])
>>> multi_signals[:, 0:2]
array([[  10.,   20.],
       [  20.,   30.],
       [  30.,   40.],
       [  40.,   50.],
       [  50.,   60.],
       [  60.,   70.],
       [  70.,   80.],
       [  80.,   90.],
       [  90.,  100.],
       [ 100.,  110.]])
```

__setitem__(*x: ArrayLike | slice*, *y: ArrayLike*)

Set the corresponding range variable y for independent domain variable x.

Parameters

- **x** (ArrayLike | slice) – Independent domain variable x.
- **y** (ArrayLike) – Corresponding range variable y.

Examples

```
>>> domain = np.arange(0, 10, 1)
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> multi_signals = MultiSignals(range_)
>>> print(multi_signals)
[[   0.   10.   20.   30.]
 [   1.   20.   30.   40.]
 [   2.   30.   40.   50.]
 [   3.   40.   50.   60.]
 [   4.   50.   60.   70.]
 [   5.   60.   70.   80.]
 [   6.   70.   80.   90.]
 [   7.   80.   90.  100.]
 [   8.   90.  100.  110.]
 [   9.  100.  110.  120.]]
```

(continues on next page)

(continued from previous page)

```
>>> multi_signals[0] = 20
>>> multi_signals[0]
array([ 20.,  20.,  20.])
>>> multi_signals[np.array([0, 1, 2])] = 30
>>> multi_signals[np.array([0, 1, 2])]
array([[ 30.,  30.,  30.],
       [ 30.,  30.,  30.],
       [ 30.,  30.,  30.]])
>>> multi_signals[np.linspace(0, 5, 5)] = 50
>>> print(multi_signals)
[[   0.     50.     50.     50.  ]
 [   1.     30.     30.     30.  ]
 [   1.25   50.     50.     50.  ]
 [   2.     30.     30.     30.  ]
 [   2.5    50.     50.     50.  ]
 [   3.     40.     50.     60.  ]
 [   3.75   50.     50.     50.  ]
 [   4.     50.     60.     70.  ]
 [   5.     50.     50.     50.  ]
 [   6.     70.     80.     90.  ]
 [   7.     80.     90.    100.  ]
 [   8.     90.    100.    110.  ]
 [   9.    100.    110.    120.  ]]
>>> multi_signals[np.array([0, 1, 2])] = np.array([10, 20, 30])
>>> print(multi_signals)
[[   0.     10.     20.     30.  ]
 [   1.     10.     20.     30.  ]
 [   1.25   50.     50.     50.  ]
 [   2.     10.     20.     30.  ]
 [   2.5    50.     50.     50.  ]
 [   3.     40.     50.     60.  ]
 [   3.75   50.     50.     50.  ]
 [   4.     50.     60.     70.  ]
 [   5.     50.     50.     50.  ]
 [   6.     70.     80.     90.  ]
 [   7.     80.     90.    100.  ]
 [   8.     90.    100.    110.  ]
 [   9.    100.    110.    120.  ]]
>>> y = np.arange(1, 10, 1).reshape(3, 3)
>>> multi_signals[np.array([0, 1, 2])] = y
>>> print(multi_signals)
[[   0.      1.      2.      3.  ]
 [   1.      4.      5.      6.  ]
 [   1.25   50.     50.     50.  ]
 [   2.      7.      8.      9.  ]
 [   2.5    50.     50.     50.  ]
 [   3.     40.     50.     60.  ]
 [   3.75   50.     50.     50.  ]
 [   4.     50.     60.     70.  ]
 [   5.     50.     50.     50.  ]
 [   6.     70.     80.     90.  ]
 [   7.     80.     90.    100.  ]
 [   8.     90.    100.    110.  ]
 [   9.    100.    110.    120.  ]]
>>> multi_signals[0:3] = 40
>>> multi_signals[0:3]
```

(continues on next page)

(continued from previous page)

```
array([[ 40.,  40.,  40.],
       [ 40.,  40.,  40.],
       [ 40.,  40.,  40.]])
>>> multi_signals[:, 0:2] = 50
>>> print(multi_signals)
[[   0.     50.     50.     40.  ]
 [   1.     50.     50.     40.  ]
 [   1.25   50.     50.     40.  ]
 [   2.     50.     50.      9.  ]
 [   2.5    50.     50.     50.  ]
 [   3.     50.     50.     60.  ]
 [   3.75   50.     50.     50.  ]
 [   4.     50.     50.     70.  ]
 [   5.     50.     50.     50.  ]
 [   6.     50.     50.     90.  ]
 [   7.     50.     50.    100.  ]
 [   8.     50.     50.    110.  ]
 [   9.     50.     50.    120.  ]]
```

__contains__(*x: ArrayLike | slice*) → bool

Return whether the multi-continuous signals contains given independent domain variable x.

Parameters **x** (ArrayLike | slice) – Independent domain variable x.

Returns Whether x domain value is contained.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> multi_signals = MultiSignals(range_)
>>> 0 in multi_signals
True
>>> 0.5 in multi_signals
True
>>> 1000 in multi_signals
False
```

__eq__(*other: Any*) → bool

Return whether the multi-continuous signals is equal to given other object.

Parameters **other** (Any) – Object to test whether it is equal to the multi-continuous signals.

Returns Whether given object is equal to the multi-continuous signals.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> multi_signals_1 = MultiSignals(range_)
>>> multi_signals_2 = MultiSignals(range_)
>>> multi_signals_1 == multi_signals_2
True
>>> multi_signals_2[0] = 20
>>> multi_signals_1 == multi_signals_2
False
>>> multi_signals_2[0] = 10
>>> multi_signals_1 == multi_signals_2
True
>>> from colour.algebra import CubicSplineInterpolator
>>> multi_signals_2.interpolator = CubicSplineInterpolator
>>> multi_signals_1 == multi_signals_2
False
```

__ne__(*other: Any*) → bool

Return whether the multi-continuous signals is not equal to given other object.

Parameters other (Any) – Object to test whether it is not equal to the multi-continuous signals.

Returns Whether given object is not equal to the multi-continuous signals.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> multi_signals_1 = MultiSignals(range_)
>>> multi_signals_2 = MultiSignals(range_)
>>> multi_signals_1 != multi_signals_2
False
>>> multi_signals_2[0] = 20
>>> multi_signals_1 != multi_signals_2
True
>>> multi_signals_2[0] = 10
>>> multi_signals_1 != multi_signals_2
False
>>> from colour.algebra import CubicSplineInterpolator
>>> multi_signals_2.interpolator = CubicSplineInterpolator
>>> multi_signals_1 != multi_signals_2
True
```

arithmetical_operation(*a: ArrayLike | AbstractContinuousFunction, operation: Literal['+', '-', '*', '/', '**'], in_place: bool = False*) → *AbstractContinuousFunction*

Perform given arithmetical operation with operand a, the operation can be either performed on a copy or in-place.

Parameters

- **a** (ArrayLike | AbstractContinuousFunction) – Operand a.
- **operation** (Literal['+', '-', '*', '/', '**']) – Operation to perform.
- **in_place** (bool) – Operation happens in place.

Returns multi-continuous signals.

Return type colour.continuous.MultiSignals

Examples

Adding a single *numeric* variable:

```
>>> domain = np.arange(0, 10, 1)
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> multi_signals_1 = MultiSignals(range_)
>>> print(multi_signals_1)
[[   0.    10.    20.    30.]
 [   1.    20.    30.    40.]
 [   2.    30.    40.    50.]
 [   3.    40.    50.    60.]
 [   4.    50.    60.    70.]
 [   5.    60.    70.    80.]
 [   6.    70.    80.    90.]
 [   7.    80.    90.   100.]
 [   8.    90.   100.   110.]
 [   9.   100.   110.   120.]]
>>> print(multi_signals_1.arithmetical_operation(10, "+", True))
[[   0.    20.    30.    40.]
 [   1.    30.    40.    50.]
 [   2.    40.    50.    60.]
 [   3.    50.    60.    70.]
 [   4.    60.    70.    80.]
 [   5.    70.    80.    90.]
 [   6.    80.    90.   100.]
 [   7.    90.   100.   110.]
 [   8.   100.   110.   120.]
 [   9.   110.   120.   130.]]
```

Adding an *ArrayLike* variable:

```
>>> a = np.linspace(10, 100, 10)
>>> print(multi_signals_1.arithmetical_operation(a, "+", True))
[[   0.    30.    40.    50.]
 [   1.    50.    60.    70.]
 [   2.    70.    80.    90.]
 [   3.    90.   100.   110.]
 [   4.   110.   120.   130.]
 [   5.   130.   140.   150.]
 [   6.   150.   160.   170.]
 [   7.   170.   180.   190.]
 [   8.   190.   200.   210.]
 [   9.   210.   220.   230.]]
```

```
>>> a = np.array([[10, 20, 30]])
>>> print(multi_signals_1.arithmetical_operation(a, "+", True))
[[   0.    40.    60.    80.]
 [   1.    60.    80.   100.]
 [   2.    80.   100.   120.]
 [   3.   100.   120.   140.]
 [   4.   120.   140.   160.]
 [   5.   140.   160.   180.]
 [   6.   160.   180.   200.]
```

(continues on next page)

(continued from previous page)

```
[   7.  180.  200.  220.]
[   8.  200.  220.  240.]
[   9.  220.  240.  260.]]
```

```
>>> a = np.arange(0, 30, 1).reshape([10, 3])
>>> print(multi_signals_1.arithmetical_operation(a, "+", True))
[[   0.   40.   61.   82.]
 [   1.   63.   84.  105.]
 [   2.   86.  107.  128.]
 [   3.  109.  130.  151.]
 [   4.  132.  153.  174.]
 [   5.  155.  176.  197.]
 [   6.  178.  199.  220.]
 [   7.  201.  222.  243.]
 [   8.  224.  245.  266.]
 [   9.  247.  268.  289.]]
```

Adding a `colour.continuous.Signal` sub-class:

```
>>> multi_signals_2 = MultiSignals(range_)
>>> print(
...     multi_signals_1.arithmetical_operation(
...         multi_signals_2, "+", True
...     )
... )
[[   0.   50.   81.  112.]
 [   1.   83.  114.  145.]
 [   2.  116.  147.  178.]
 [   3.  149.  180.  211.]
 [   4.  182.  213.  244.]
 [   5.  215.  246.  277.]
 [   6.  248.  279.  310.]
 [   7.  281.  312.  343.]
 [   8.  314.  345.  376.]
 [   9.  347.  378.  409.]]
```

static multi_signals_unpack_data(*data: ArrayLike | DataFrame | dict | MultiSignals | Sequence | Series | Signal | None = None, domain: ArrayLike | None = None, labels: Sequence | None = None, dtype: Type[DTypeFloat] | None = None, signal_type: Type[Signal] = Signal, **kwargs: Any*) → Dict[str, *Signal*]

Unpack given data for multi-continuous signals instantiation.

Parameters

- **data** (ArrayLike | DataFrame | dict | MultiSignals | Sequence | Series | Signal | None) – Data to unpack for multi-continuous signals instantiation.
- **domain** (ArrayLike | None) – 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` property.
- **labels** (Sequence | None) – Names to use for the `colour.continuous.Signal` sub-class instances.
- **dtype** (Type[DTypeFloat] | None) – float point data type.
- **signal_type** (Type[Signal]) – A `colour.continuous.Signal` sub-class type.

- **extrapolator** – Extrapolator class type to use as extrapolating function for the `colour.continuous.Signal` sub-class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the `colour.continuous.Signal` sub-class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the `colour.continuous.Signal` sub-class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the `colour.continuous.Signal` sub-class instances.
- **name** – multi-continuous signals name.
- **kwargs** (`Any`) –

Returns Mapping of labeled `colour.continuous.Signal` sub-class instances.

Return type `dict`

Examples

Unpacking using implicit *domain* and data for a single signal:

```
>>> range_ = np.linspace(10, 100, 10)
>>> signals = MultiSignals.multi_signals_unpack_data(range_)
>>> list(signals.keys())
['0']
>>> print(signals["0"])
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
```

Unpacking using explicit *domain* and data for a single signal:

```
>>> domain = np.arange(100, 1100, 100)
>>> signals = MultiSignals.multi_signals_unpack_data(range_, domain)
>>> list(signals.keys())
['0']
>>> print(signals["0"])
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Unpacking using data for multiple signals:

```
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> signals = MultiSignals.multi_signals_unpack_data(range_, domain)
>>> list(signals.keys())
['0', '1', '2']
>>> print(signals["2"])
[[  100.     30.]
 [  200.     40.]
 [  300.     50.]
 [  400.     60.]
 [  500.     70.]
 [  600.     80.]
 [  700.     90.]
 [  800.    100.]
 [  900.    110.]
 [ 1000.    120.]]
```

Unpacking using a *dict*:

```
>>> signals = MultiSignals.multi_signals_unpack_data(
...     dict(zip(domain, range_))
... )
>>> list(signals.keys())
['0', '1', '2']
>>> print(signals["2"])
[[  100.     30.]
 [  200.     40.]
 [  300.     50.]
 [  400.     60.]
 [  500.     70.]
 [  600.     80.]
 [  700.     90.]
 [  800.    100.]
 [  900.    110.]
 [ 1000.    120.]]
```

Unpacking using a sequence of *Signal* instances, note how the keys are `str` instances because the *Signal* names are used:

```
>>> signals = MultiSignals.multi_signals_unpack_data(
...     dict(zip(domain, range_))
... ).values()
>>> signals = MultiSignals.multi_signals_unpack_data(signals)
>>> list(signals.keys())
['0', '1', '2']
>>> print(signals["2"])
[[  100.     30.]
 [  200.     40.]
 [  300.     50.]
 [  400.     60.]
 [  500.     70.]
 [  600.     80.]
 [  700.     90.]
 [  800.    100.]
 [  900.    110.]
 [ 1000.    120.]]
```

Unpacking using *MultiSignals.multi_signals_unpack_data* method output:

```
>>> signals = MultiSignals.multi_signals_unpack_data(
...     dict(zip(domain, range_))
... )
>>> signals = MultiSignals.multi_signals_unpack_data(signals)
>>> list(signals.keys())
['0', '1', '2']
>>> print(signals["2"])
[[  100.    30.]
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
 [  700.    90.]
 [  800.   100.]
 [  900.   110.]
 [ 1000.   120.]]
```

Unpacking using a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...
...     signals = MultiSignals.multi_signals_unpack_data(
...         Series(dict(zip(domain, np.linspace(10, 100, 10))))
...     )
...     print(signals[0])
...
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Unpacking using a *Pandas* `pandas.DataFrame`:

```
>>> if is_pandas_installed():
...     from pandas import DataFrame
...
...     data = dict(zip(["a", "b", "c"], tsplit(range_)))
...     signals = MultiSignals.multi_signals_unpack_data(
...         DataFrame(data, domain)
...     )
...     print(signals["c"])
...
[[  100.    30.]
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
 [  700.    90.]
```

(continues on next page)

(continued from previous page)

```
[  800.   100.]
[  900.   110.]
[ 1000.   120.]]
```

fill_nan(*method: Union[Literal['Interpolation', 'Constant'], str] = 'Interpolation', default: Real = 0*) → *colour.continuous.abstract.AbstractContinuousFunction*

Fill NaNs in independent domain variable x and corresponding range variable y using given method.

Parameters

- **method** (Union[Literal['Interpolation', 'Constant'], str]) – *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with default.
- **default** (Real) – Value to use with the *Constant* method.

Returns

- colour.continuous.MultiSignals – NaNs filled multi-continuous signals.
- *>>> domain = np.arange(0, 10, 1)*
- *>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)*
- *>>> range_ += np.array([0, 10, 20])*
- *>>> multi_signals = MultiSignals(range_)*
- **>>> multi_signals[3** (*7] = np.nan*)
- *>>> print(multi_signals)*
- *[[0. 10. 20. 30.]* – [1. 20. 30. 40.] [2. 30. 40. 50.] [3. nan nan nan] [4. nan nan nan] [5. nan nan nan] [6. nan nan nan] [7. 80. 90. 100.] [8. 90. 100. 110.] [9. 100. 110. 120.]]
- *>>> print(multi_signals.fill_nan())*
- *[[0. 10. 20. 30.]* – [1. 20. 30. 40.] [2. 30. 40. 50.] [3. 40. 50. 60.] [4. 50. 60. 70.] [5. 60. 70. 80.] [6. 70. 80. 90.] [7. 80. 90. 100.] [8. 90. 100. 110.] [9. 100. 110. 120.]]
- **>>> multi_signals[3** (*7] = np.nan*)
- *>>> print(multi_signals.fill_nan(method="Constant"))*
- *[[0. 10. 20. 30.]* – [1. 20. 30. 40.] [2. 30. 40. 50.] [3. 0. 0. 0.] [4. 0. 0. 0.] [5. 0. 0. 0.] [6. 0. 0. 0.] [7. 80. 90. 100.] [8. 90. 100. 110.] [9. 100. 110. 120.]]

Return type *colour.continuous.abstract.AbstractContinuousFunction*

to_dataframe() → pandas.core.frame.DataFrame

Convert the continuous signal to a *Pandas* pandas.DataFrame class instance.

Returns Continuous signal as a *Pandas* pandas.DataFrame class instance.

Return type pandas.DataFrame

Examples

```
>>> if is_pandas_installed():
...     domain = np.arange(0, 10, 1)
...     range_ = tstack([np.linspace(10, 100, 10)] * 3)
...     range_ += np.array([0, 10, 20])
...     multi_signals = MultiSignals(range_)
...     print(multi_signals.to_dataframe())
...
         0      1      2
0.0   10.0   20.0   30.0
1.0   20.0   30.0   40.0
2.0   30.0   40.0   50.0
3.0   40.0   50.0   60.0
4.0   50.0   60.0   70.0
5.0   60.0   70.0   80.0
6.0   70.0   80.0   90.0
7.0   80.0   90.0  100.0
8.0   90.0  100.0  110.0
9.0  100.0  110.0  120.0
```

Corresponding Chromaticities

Prediction

colour

corresponding_chromaticities_prediction([...])	Return the corresponding chromaticities prediction for given chromatic adaptation model.
CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS	Aggregated corresponding chromaticities prediction models.
CorrespondingColourDataset(name, XYZ_r, ...)	Define a corresponding colour dataset.
CorrespondingChromaticitiesPrediction(name, ...)	Define a chromatic adaptation model prediction.

colour.corresponding_chromaticities_prediction

colour.**corresponding_chromaticities_prediction**(*experiment: Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.CorrespondingColourDataset] = 1, model: Union[Literal['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Zhai 2018', 'Von Kries'], str] = 'Von Kries', **kwargs: Any*) → Tuple[*colour.corresponding.prediction.CorrespondingChromaticitiesPred ...*]

Return the corresponding chromaticities prediction for given chromatic adaptation model.

Parameters

- **experiment** (Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.CorrespondingColourDataset]) – *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.
- **model** (Union[Literal['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Zhai 2018', 'Von Kries'], str]) – Chromatic adaptation model.

- **D_b** – {colour.corresponding.corresponding_chromaticities_prediction_Zhai2018()}, Degree of adaptation D_β of input illuminant β.
- **D_d** – {colour.corresponding.corresponding_chromaticities_prediction_Zhai2018()}, Degree of adaptation D_δ of output illuminant δ.
- **transform** – {colour.corresponding.corresponding_chromaticities_prediction_VonKries(), colour.corresponding.corresponding_chromaticities_prediction_Zhai2018()}, Chromatic adaptation transform.
- **XYZ_wo** – {colour.corresponding.corresponding_chromaticities_prediction_Zhai2018()}, Baseline illuminant (BI) o.
- **kwargs** (Any) –

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [CIET13294], [Fai91], [Fai13f], [Fai13a], [LLRH02], [WRC12d], [ZL18]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction(2, "CMCCAT2000")
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[((0.207, 0.486), (0.2083210..., 0.4727168...)),
 ((0.449, 0.511), (0.4459270..., 0.5077735...)),
 ((0.263, 0.505), (0.2640262..., 0.4955361...)),
 ((0.322, 0.545), (0.3316884..., 0.5431580...)),
 ((0.316, 0.537), (0.3222624..., 0.5357624...)),
 ((0.265, 0.553), (0.2710705..., 0.5501997...)),
 ((0.221, 0.538), (0.2261826..., 0.5294740...)),
 ((0.135, 0.532), (0.1439693..., 0.5190984...)),
 ((0.145, 0.472), (0.1494835..., 0.4556760...)),
 ((0.163, 0.331), (0.1563172..., 0.3164151...)),
 ((0.176, 0.431), (0.1763199..., 0.4127589...)),
 ((0.244, 0.349), (0.2287638..., 0.3499324...))]
```

colour.CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS

colour.**CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS = CanonicalMapping({'CIE 1994': ..., 'CMCCAT2000': ..., 'Fairchild 1990': ..., 'Von Kries': ..., 'Zhai 2018': ..., 'vonkries': ...})**

Aggregated corresponding chromaticities prediction models.

References

[Bre87], [CIET13294], [Fai91], [Fai13f], [Fai13a], [LLRH02], [WRC12d], [ZL18]

Aliases:

- 'vonkries': 'Von Kries'

colour.CorrespondingColourDataset

class colour.**CorrespondingColourDataset**(*name, XYZ_r, XYZ_t, XYZ_cr, XYZ_ct, Y_r, Y_t, B_r, B_t, metadata*)

Define a corresponding colour dataset.

Parameters

- **name** – Corresponding colour dataset name.
- **XYZ_r** – *CIE XYZ* tristimulus values of the reference illuminant.
- **XYZ_t** – *CIE XYZ* tristimulus values of the test illuminant.
- **XYZ_cr** – Corresponding *CIE XYZ* tristimulus values under the reference illuminant.
- **XYZ_ct** – Corresponding *CIE XYZ* tristimulus values under the test illuminant.
- **Y_r** – Reference white luminance Y_r in cd/m^2.
- **Y_t** – Test white luminance Y_t in cd/m^2.
- **B_r** – Luminance factor B_r of reference achromatic background as percentage.
- **B_t** – Luminance factor B_t of test achromatic background as percentage.
- **metadata** – Dataset metadata.

Notes

- This class is compatible with *Luo and Rhodes (1999) Corresponding-Colour Datasets* datasets.

References

[LR99]

Create new instance of CorrespondingColourDataset(name, XYZ_r, XYZ_t, XYZ_cr, XYZ_ct, Y_r, Y_t, B_r, B_t, metadata)

__init__()

Methods

__init__()	
count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

B_r	Alias for field number 7
B_t	Alias for field number 8
XYZ_cr	Alias for field number 3
XYZ_ct	Alias for field number 4
XYZ_r	Alias for field number 1
XYZ_t	Alias for field number 2
Y_r	Alias for field number 5
Y_t	Alias for field number 6
metadata	Alias for field number 9
name	Alias for field number 0

colour.CorrespondingChromaticitiesPrediction

class colour.**CorrespondingChromaticitiesPrediction**(*name*, *uv_t*, *uv_m*, *uv_p*)

Define a chromatic adaptation model prediction.

Parameters

- **name** – Test colour name.
- **uv_t** – Chromaticity coordinates uv_t^p of test colour.
- **uv_m** – Chromaticity coordinates uv_m^p of matching colour.
- **uv_p** – Chromaticity coordinates uv_p^p of predicted colour.

Create new instance of CorrespondingChromaticitiesPrediction(name, uv_t, uv_m, uv_p)

__init__()

Methods

__init__()	
count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

name	Alias for field number 0
uv_m	Alias for field number 2
uv_p	Alias for field number 3
uv_t	Alias for field number 1

Dataset

colour

BRENEMAN_EXPERIMENTS	*Breneman (1987)* experiments.
BRENEMAN_EXPERIMENT_PRIMARIES_CHROMATICITIES	*Breneman (1987)* experiments primaries chromaticities.

colour.BRENEMAN_EXPERIMENTS

```
colour.BRENEMAN_EXPERIMENTS = {1: (BrenemanExperimentResult(name='Illuminant',
uv_t=array([ 0.259, 0.526]), uv_m=array([ 0.2 , 0.475]), s_uv=array(None, dtype=object),
d_uv_i=array(None, dtype=object), d_uv_g=array(None, dtype=object)),
BrenemanExperimentResult(name='Gray', uv_t=array([ 0.259, 0.524]), uv_m=array([ 0.199,
0.487]), s_uv=array([4, 4]), d_uv_i=array([2, 3]), d_uv_g=array([0, 0])),
BrenemanExperimentResult(name='Red', uv_t=array([ 0.459, 0.522]), uv_m=array([ 0.42 ,
0.509]), s_uv=array([19, 4]), d_uv_i=array([-10, -7]), d_uv_g=array([-19, -3])),
BrenemanExperimentResult(name='Skin', uv_t=array([ 0.307, 0.526]), uv_m=array([ 0.249,
0.497]), s_uv=array([7, 4]), d_uv_i=array([-1, 1]), d_uv_g=array([-6, -1])),
BrenemanExperimentResult(name='Orange', uv_t=array([ 0.36 , 0.544]), uv_m=array([ 0.302,
0.548]), s_uv=array([12, 1]), d_uv_i=array([ 1, -2]), d_uv_g=array([-7, -6])),
BrenemanExperimentResult(name='Brown', uv_t=array([ 0.35 , 0.541]), uv_m=array([ 0.29 ,
0.537]), s_uv=array([11, 4]), d_uv_i=array([3, 0]), d_uv_g=array([-5, -3])),
BrenemanExperimentResult(name='Yellow', uv_t=array([ 0.318, 0.55 ]), uv_m=array([ 0.257,
0.554]), s_uv=array([8, 2]), d_uv_i=array([0, 2]), d_uv_g=array([-5, -5])),
BrenemanExperimentResult(name='Foliage', uv_t=array([ 0.258, 0.542]), uv_m=array([ 0.192,
0.529]), s_uv=array([4, 6]), d_uv_i=array([3, 2]), d_uv_g=array([ 3, -6])),
BrenemanExperimentResult(name='Green', uv_t=array([ 0.193, 0.542]), uv_m=array([ 0.129,
0.521]), s_uv=array([7, 5]), d_uv_i=array([3, 2]), d_uv_g=array([ 9, -7])),
BrenemanExperimentResult(name='Blue-green', uv_t=array([ 0.18 , 0.516]), uv_m=array([
0.133, 0.469]), s_uv=array([4, 6]), d_uv_i=array([-3, -2]), d_uv_g=array([ 2, -5])),
BrenemanExperimentResult(name='Blue', uv_t=array([ 0.186, 0.445]), uv_m=array([ 0.158,
0.34 ]), s_uv=array([13, 33]), d_uv_i=array([2, 7]), d_uv_g=array([ 1, 13])),
BrenemanExperimentResult(name='Sky', uv_t=array([ 0.226, 0.491]), uv_m=array([ 0.178,
0.426]), s_uv=array([ 3, 14]), d_uv_i=array([ 1, -3]), d_uv_g=array([ 0, -1])),
BrenemanExperimentResult(name='Purple', uv_t=array([ 0.278, 0.456]), uv_m=array([ 0.231,
0.365]), s_uv=array([ 4, 25]), d_uv_i=array([0, 2]), d_uv_g=array([-5, 7]))), 2:
(BrenemanExperimentResult(name='Illuminant', uv_t=array([ 0.222, 0.521]), uv_m=array([
0.204, 0.479]), s_uv=array(None, dtype=object), d_uv_i=array(None, dtype=object),
d_uv_g=array(None, dtype=object)), BrenemanExperimentResult(name='Gray', uv_t=array([
0.227, 0.517]), uv_m=array([ 0.207, 0.486]), s_uv=array([2, 5]), d_uv_i=array([-1, 0]),
d_uv_g=array([0, 0])), BrenemanExperimentResult(name='Red', uv_t=array([ 0.464, 0.52 ]),
uv_m=array([ 0.449, 0.511]), s_uv=array([22, 3]), d_uv_i=array([-8, -8]),
d_uv_g=array([-7, -2])), BrenemanExperimentResult(name='Skin', uv_t=array([ 0.286,
0.526]), uv_m=array([ 0.263, 0.505]), s_uv=array([7, 2]), d_uv_i=array([ 0, -1]),
d_uv_g=array([ 0, -1])), BrenemanExperimentResult(name='Orange', uv_t=array([ 0.348,
0.546]), uv_m=array([ 0.322, 0.545]), s_uv=array([13, 3]), d_uv_i=array([ 3, -1]),
d_uv_g=array([ 3, -2])), BrenemanExperimentResult(name='Brown', uv_t=array([ 0.34 ,
0.543]), uv_m=array([ 0.316, 0.537]), s_uv=array([11, 3]), d_uv_i=array([1, 1]),
d_uv_g=array([0, 0])), BrenemanExperimentResult(name='Yellow', uv_t=array([ 0.288,
0.554]), uv_m=array([ 0.265, 0.553]), s_uv=array([5, 2]), d_uv_i=array([-2, 2]),
d_uv_g=array([-1, -2])), BrenemanExperimentResult(name='Foliage', uv_t=array([ 0.244,
0.547]), uv_m=array([ 0.221, 0.538]), s_uv=array([4, 3]), d_uv_i=array([-2, 1]),
d_uv_g=array([ 0, -3])), BrenemanExperimentResult(name='Green', uv_t=array([ 0.156,
0.548]), uv_m=array([ 0.135, 0.532]), s_uv=array([4, 3]), d_uv_i=array([-1, 3]),
d_uv_g=array([ 3, -4])), BrenemanExperimentResult(name='Blue-green', uv_t=array([ 0.159,
0.511]), uv_m=array([ 0.145, 0.472]), s_uv=array([9, 7]), d_uv_i=array([-1, 2]),
d_uv_g=array([2, 1])), BrenemanExperimentResult(name='Blue', uv_t=array([ 0.16 , 0.406]),
uv_m=array([ 0.163, 0.331]), s_uv=array([23, 31]), d_uv_i=array([ 2, -3]),
d_uv_g=array([-1, 3])), BrenemanExperimentResult(name='Sky', uv_t=array([ 0.19 , 0.481]),
uv_m=array([ 0.176, 0.431]), s_uv=array([ 5, 24]), d_uv_i=array([ 2, -2]), d_uv_g=array([2,
0])), BrenemanExperimentResult(name='Purple', uv_t=array([ 0.258, 0.431]), uv_m=array([
0.244, 0.349]), s_uv=array([ 4, 19]), d_uv_i=array([-3, 13]), d_uv_g=array([-4, 19]))), 3:
(BrenemanExperimentResult(name='Illuminant', uv_t=array([ 0.223, 0.521]), uv_m=array([
0.206, 0.478]), s_uv=array(None, dtype=object), d_uv_i=array(None, dtype=object),
d_uv_g=array(None, dtype=object)), BrenemanExperimentResult(name='Gray', uv_t=array([
0.228, 0.517]), uv_m=array([ 0.211, 0.494]), s_uv=array([1, 3]), d_uv_i=array([0, 2]),
d_uv_g=array([0, 0])), BrenemanExperimentResult(name='Red', uv_t=array([ 0.462, 0.519]),
uv_m=array([ 0.448, 0.505]), s_uv=array([11, 4]), d_uv_i=array([-3, 6]), d_uv_g=array([-4,
6])), BrenemanExperimentResult(name='Skin', uv_t=array([ 0.285, 0.524]), uv_m=array([
0.267, 0.507]), s_uv=array([6, 3]), d_uv_i=array([-1, 1]), d_uv_g=array([-2, 1])),
BrenemanExperimentResult(name='Orange', uv_t=array([ 0.346, 0.546]), uv_m=array([ 0.325,
0.541]), s_uv=array([11, 3]), d_uv_i=array([ 1, -2]), d_uv_g=array([2, 3])),
BrenemanExperimentResult(name='Brown', uv_t=array([ 0.338, 0.543]), uv_m=array([ 0.321,
```

Breneman (1987) experiments.

References

[Bre87]

colour.BRENEMAN_EXPERIMENT_PRIMARIES_CHROMATICITIES

```
colour.BRENEMAN_EXPERIMENT_PRIMARIES_CHROMATICITIES = {1:
PrimariesChromaticityCoordinates(experiment=1, illuminants=array(['A', 'D65'],
dtype='<U3'), Y=array(1500), P_uvp=array([ 0.671, 0.519]), D_uvp=array([-0.586, 0.627]),
T_uvp=array([ 0.253, 0.016])), 2: PrimariesChromaticityCoordinates(experiment=2,
illuminants=array(['Projector', 'D55'], dtype='<U9'), Y=array(1500), P_uvp=array([ 0.675,
0.523]), D_uvp=array([-0.466, 0.617]), T_uvp=array([ 0.255, 0.018])), 3:
PrimariesChromaticityCoordinates(experiment=3, illuminants=array(['Projector', 'D55'],
dtype='<U9'), Y=array(75), P_uvp=array([ 0.664, 0.51 ]), D_uvp=array([-0.256, 0.729]),
T_uvp=array([ 0.244, 0.003])), 4: PrimariesChromaticityCoordinates(experiment=4,
illuminants=array(['A', 'D65'], dtype='<U3'), Y=array(75), P_uvp=array([ 0.674, 0.524]),
D_uvp=array([-0.172, 0.628]), T_uvp=array([ 0.218, -0.026])), 6:
PrimariesChromaticityCoordinates(experiment=6, illuminants=array(['A', 'D55'],
dtype='<U3'), Y=array(11100), P_uvp=array([ 0.659, 0.506]), D_uvp=array([-0.141, 0.615]),
T_uvp=array([ 0.249, 0.009])), 8: PrimariesChromaticityCoordinates(experiment=8,
illuminants=array(['A', 'D65'], dtype='<U3'), Y=array(350), P_uvp=array([ 0.659, 0.505]),
D_uvp=array([-0.246, 0.672]), T_uvp=array([ 0.235, -0.006])), 9:
PrimariesChromaticityCoordinates(experiment=9, illuminants=array(['A', 'D65'],
dtype='<U3'), Y=array(15), P_uvp=array([ 0.693, 0.546]), D_uvp=array([-0.446, 0.773]),
T_uvp=array([ 0.221, -0.023])), 11: PrimariesChromaticityCoordinates(experiment=11,
illuminants=array(['D55', 'green'], dtype='<U5'), Y=array(1560), P_uvp=array([ 0.68 ,
0.529]), D_uvp=array([ 0.018, 0.576]), T_uvp=array([ 0.307, 0.08 ])), 12:
PrimariesChromaticityCoordinates(experiment=12, illuminants=array(['D55', 'green'],
dtype='<U5'), Y=array(75), P_uvp=array([ 0.661, 0.505]), D_uvp=array([ 0.039, 0.598]),
T_uvp=array([ 0.345, 0.127]))}
```

Breneman (1987) experiments primaries chromaticities.

References

[Bre87]

BRENEMAN_EXPERIMENT_PRIMARIES_CHROMATICITIES : dict

Fairchild (1990)

`colour.corresponding`

`corresponding_chromaticities_prediction_Fairchild1990([…])`	Return the corresponding chromaticities prediction for *Fairchild (1990)* chromatic adaptation model.

colour.corresponding.corresponding_chromaticities_prediction_Fairchild1990

colour.corresponding.**corresponding_chromaticities_prediction_Fairchild1990**(*experiment: Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.Co... = 1*) → Tuple[*colour.corresponding.prediction...*, ...]

Return the corresponding chromaticities prediction for *Fairchild (1990)* chromatic adaptation model.

Parameters **experiment** (Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.CorrespondingColourDataset]) – *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [Fai91], [Fai13f]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_Fairchild1990(2)
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2089528...,  0.4724034...])),
 (array([ 0.449,  0.511]), array([ 0.4375652...,  0.5121030...])),
 (array([ 0.263,  0.505]), array([ 0.2621362...,  0.4972538...])),
 (array([ 0.322,  0.545]), array([ 0.3235312...,  0.5475665...])),
 (array([ 0.316,  0.537]), array([ 0.3151391...,  0.5398333...])),
 (array([ 0.265,  0.553]), array([ 0.2634745...,  0.5544335...])),
 (array([ 0.221,  0.538]), array([ 0.2211595...,  0.5324470...])),
 (array([ 0.135,  0.532]), array([ 0.1396949...,  0.5207234...])),
 (array([ 0.145,  0.472]), array([ 0.1512288...,  0.4533041...])),
 (array([ 0.163,  0.331]), array([ 0.1715691...,  0.3026264...])),
 (array([ 0.176,  0.431]), array([ 0.1825792...,  0.4077892...])),
 (array([ 0.244,  0.349]), array([ 0.2418905...,  0.3413401...]))]
```

CIE 1994

colour.corresponding

corresponding_chromaticities_prediction_CIE1994([...])	Return the corresponding chromaticities prediction for *CIE 1994* chromatic adaptation model.

colour.corresponding.corresponding_chromaticities_prediction_CIE1994

colour.corresponding.**corresponding_chromaticities_prediction_CIE1994**(*experiment: Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.Correspo = 1)* → Tuple[*colour.corresponding.prediction.Corres* ...]

Return the corresponding chromaticities prediction for *CIE 1994* chromatic adaptation model.

Parameters **experiment** (Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.CorrespondingColourDataset]) – *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [CIET13294]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CIE1994(2)
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2273130...,  0.5267609...])),
 (array([ 0.449,  0.511]), array([ 0.4612181...,  0.5191849...])),
 (array([ 0.263,  0.505]), array([ 0.2872404...,  0.5306938...])),
 (array([ 0.322,  0.545]), array([ 0.3489822...,  0.5454398...])),
 (array([ 0.316,  0.537]), array([ 0.3371612...,  0.5421567...])),
 (array([ 0.265,  0.553]), array([ 0.2889416...,  0.5534074...])),
 (array([ 0.221,  0.538]), array([ 0.2412195...,  0.5464301...])),
 (array([ 0.135,  0.532]), array([ 0.1530344...,  0.5488239...])),
 (array([ 0.145,  0.472]), array([ 0.1568709...,  0.5258835...])),
 (array([ 0.163,  0.331]), array([ 0.1499762...,  0.4401747...])),
 (array([ 0.176,  0.431]), array([ 0.1876711...,  0.5039627...])),
 (array([ 0.244,  0.349]), array([ 0.2560012...,  0.4546263...]))]
```

CMCCAT2000

colour.corresponding

corresponding_chromaticities_prediction_CMCCAT2000([...])	Return the corresponding chromaticities prediction for *CMCCAT2000* chromatic adaptation model.

colour.corresponding.corresponding_chromaticities_prediction_CMCCAT2000

colour.corresponding.**corresponding_chromaticities_prediction_CMCCAT2000**(*experiment: Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.Corre = 1)* → Tuple[*colour.corresponding.prediction.Co* ...]

Return the corresponding chromaticities prediction for *CMCCAT2000* chromatic adaptation model.

Parameters **experiment** (Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.CorrespondingColourDataset]) – *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [LLRH02], [WRC12d]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CMCCAT2000(2)
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2083210...,  0.4727168...])),
 (array([ 0.449,  0.511]), array([ 0.4459270...,  0.5077735...])),
 (array([ 0.263,  0.505]), array([ 0.2640262...,  0.4955361...])),
 (array([ 0.322,  0.545]), array([ 0.3316884...,  0.5431580...])),
 (array([ 0.316,  0.537]), array([ 0.3222624...,  0.5357624...])),
 (array([ 0.265,  0.553]), array([ 0.2710705...,  0.5501997...])),
 (array([ 0.221,  0.538]), array([ 0.2261826...,  0.5294740...])),
 (array([ 0.135,  0.532]), array([ 0.1439693...,  0.5190984...])),
 (array([ 0.145,  0.472]), array([ 0.1494835...,  0.4556760...])),
 (array([ 0.163,  0.331]), array([ 0.1563172...,  0.3164151...])),
 (array([ 0.176,  0.431]), array([ 0.1763199...,  0.4127589...])),
 (array([ 0.244,  0.349]), array([ 0.2287638...,  0.3499324...]))]
```

Von Kries

colour.corresponding

corresponding_chromaticities_prediction_VonKries(...)	Return 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: Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.Corresp = 1, transform: Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str] = 'CAT02'*) → Tuple[*colour.corresponding.prediction.Corre* ...]

Return the corresponding chromaticities prediction for *Von Kries* chromatic adaptation model using given transform.

Parameters

- **experiment** (Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.CorrespondingColourDataset]) – *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.
- **transform** (Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]) – Chromatic adaptation transform.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [Fai13a]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_VonKries(2, "Bradford")
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2082014...,  0.4722922...])),
 (array([ 0.449,  0.511]), array([ 0.4489102...,  0.5071602...])),
 (array([ 0.263,  0.505]), array([ 0.2643545...,  0.4959631...])),
 (array([ 0.322,  0.545]), array([ 0.3348730...,  0.5471220...])),
 (array([ 0.316,  0.537]), array([ 0.3248758...,  0.5390589...])),
 (array([ 0.265,  0.553]), array([ 0.2733105...,  0.5555028...])),
 (array([ 0.221,  0.538]), array([ 0.227148 ...,  0.5331318...])),
 (array([ 0.135,  0.532]), array([ 0.1442730...,  0.5226804...])),
 (array([ 0.145,  0.472]), array([ 0.1498745...,  0.4550785...])),
 (array([ 0.163,  0.331]), array([ 0.1564975...,  0.3148796...])),
```

(continues on next page)

(continued from previous page)

```
(array([ 0.176,  0.431]), array([ 0.1760593...,  0.4103772...])),
(array([ 0.244,  0.349]), array([ 0.2259805...,  0.3465291...]))]
```

Colour Difference

Delta E

colour

delta_E(a, b[, method])	Return the difference ΔE_{ab} between two given *CIE L*a*b**, IC_TC_P, or $J'a'b'$ colourspace arrays using given method.
DELTA_E_METHODS	Supported ΔE_{ab} computation methods.

colour.delta_E

colour.**delta_E**(*a: ArrayLike, b: ArrayLike, method: Union[Literal['CIE 1976', 'CIE 1994', 'CIE 2000', 'CMC', 'ITP', 'CAM02-LCD', 'CAM02-SCD', 'CAM02-UCS', 'CAM16-LCD', 'CAM16-SCD', 'CAM16-UCS', 'DIN99'], str] = 'CIE 2000', **kwargs: Any*) → NDArrayFloat

Return the difference ΔE_{ab} between two given *CIE L*a*b**, IC_TC_P, or $J'a'b'$ colourspace arrays using given method.

Parameters

- **a** (ArrayLike) – *CIE L*a*b**, IC_TC_P, or $J'a'b'$ colourspace array a.
- **b** (ArrayLike) – *CIE L*a*b**, IC_TC_P, or $J'a'b'$ colourspace array b.
- **method** (Union[Literal['CIE 1976', 'CIE 1994', 'CIE 2000', 'CMC', 'ITP', 'CAM02-LCD', 'CAM02-SCD', 'CAM02-UCS', 'CAM16-LCD', 'CAM16-SCD', 'CAM16-UCS', 'DIN99'], str]) – Computation method.
- **c** – {colour.difference.delta_E_CIE2000()}, Chroma weighting factor.
- **l** – {colour.difference.delta_E_CIE2000()}, Lightness weighting factor.
- **textiles** – {colour.difference.delta_E_CIE1994(), colour.difference.delta_E_CIE2000(), colour.difference.delta_E_DIN99()}, Textiles application specific parametric factors $k_L = 2$, $k_C = k_H = 1$, $k_1 = 0.048$, $k_2 = 0.014$, $k_E = 2$, $k_CH = 0.5$ weights are used instead of $k_L = k_C = k_H = 1$, $k_1 = 0.045$, $k_2 = 0.015$, $k_E = k_CH = 1.0$.
- **kwargs** (Any) –

Returns Colour difference ΔE_{ab}.

Return type numpy.ndarray

References

[ASTMInternational07], [InternationalTUnion19], [LLW+17], [Lin03a], [Lin11], [Lin09b], [LCL06], [Mel13], [Wikipedia08a]

Examples

```
>>> import numpy as np
>>> a = np.array([100.00000000, 21.57210357, 272.22819350])
>>> b = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E(a, b)
94.0356490...
>>> delta_E(a, b, method="CIE 2000")
94.0356490...
>>> delta_E(a, b, method="CIE 1976")
451.7133019...
>>> delta_E(a, b, method="CIE 1994")
83.7792255...
>>> delta_E(a, b, method="CIE 1994", textiles=False)
...
83.7792255...
>>> delta_E(a, b, method="DIN99")
66.1119282...
>>> a = np.array([0.4885468072, -0.04739350675, 0.07475401302])
>>> b = np.array([0.4899203231, -0.04567508203, 0.07361341775])
>>> delta_E(a, b, method="ITP")
1.42657228...
>>> 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 = CanonicalMapping({'CIE 1976': ..., 'CIE 1994': ..., 'CIE 2000': ..., 'CMC': ..., 'ITP': ..., 'CAM02-LCD': ..., 'CAM02-SCD': ..., 'CAM02-UCS': ..., 'CAM16-LCD': ..., 'CAM16-SCD': ..., 'CAM16-UCS': ..., 'DIN99': ..., 'cie1976': ..., 'cie1994': ..., 'cie2000': ...})**

Supported ΔE_{ab} computation methods.

References

[ASTMInternational07], [LLW+17], [Lin03a], [Lin11], [Lin09b], [LCL06], [Mel13], [Wikipedia08a]

Aliases:

- 'cie1976': 'CIE 1976'
- 'cie1994': 'CIE 1994'
- 'cie2000': 'CIE 2000'

CIE 1976

colour.difference

JND_CIE1976	Just Noticeable Difference (JND) according to *CIE 1976* colour difference formula, i.e. Euclidean distance in *CIE L*a*b** colourspace.
delta_E_CIE1976(Lab_1, Lab_2)	Return the difference ΔE_{76} between two given *CIE L*a*b** colourspace arrays using *CIE 1976* recommendation.

colour.difference.JND_CIE1976

colour.difference.**JND_CIE1976 = 2.3**

Just Noticeable Difference (JND) according to *CIE 1976* colour difference formula, i.e. Euclidean distance in *CIE L*a*b** colourspace.

Notes

A standard observer sees the difference in colour as follows:

- $0 < \Delta E^*_{ab} < 1$: Observer does not notice the difference.
- $1 < \Delta E^*_{ab} < 2$: Only experienced observer can notice the difference.
- $2 < \Delta E^*_{ab} < 3{:}5$: Unexperienced observer also notices the difference.
- $3{:}5 < \Delta E^*_{ab} < 5$: Clear difference in colour is noticed.
- $5 < \Delta E^*_{ab}$: Observer notices two different colours.

References

[MT11]

colour.difference.delta_E_CIE1976

colour.difference.**delta_E_CIE1976**(*Lab_1: ArrayLike*, *Lab_2: ArrayLike*) → NDArrayFloat

Return the difference ΔE_{76} between two given *CIE L*a*b** colourspace arrays using *CIE 1976* recommendation.

Parameters

- **Lab_1** (ArrayLike) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (ArrayLike) – *CIE L*a*b** colourspace array 2.

Returns Colour difference ΔE_{76}.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

References

[Lin03a]

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE1976(Lab_1, Lab_2)
451.7133019...
```

CIE 1994

`colour.difference`

`delta_E_CIE1994`(Lab_1, Lab_2[, textiles])	Return the difference ΔE_{94} between two given *CIE L*a*b** colourspace arrays using *CIE 1994* recommendation.

colour.difference.delta_E_CIE1994

`colour.difference.`**`delta_E_CIE1994`**(*Lab_1: ArrayLike*, *Lab_2: ArrayLike*, *textiles: bool = False*) → NDArrayFloat

Return the difference ΔE_{94} between two given *CIE L*a*b** colourspace arrays using *CIE 1994* recommendation.

Parameters

- **Lab_1** (ArrayLike) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (ArrayLike) – *CIE L*a*b** colourspace array 2.
- **textiles** (bool) – Textiles application specific parametric factors, $k_L = 2,\ k_C = k_H = 1,\ k_1 = 0.048,\ k_2 = 0.014$ weights are used instead of $k_L = k_C = k_H = 1,\ k_1 = 0.045,\ k_2 = 0.015$.

Returns Colour difference ΔE_{94}.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

- *CIE 1994* colour differences are not symmetrical: difference between Lab_1 and Lab_2 may not be the same as difference between Lab_2 and Lab_1 thus one colour must be understood to be the reference against which a sample colour is compared.

References

[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])	Return the difference ΔE_{00} between two given *CIE L*a*b** colourspace arrays using *CIE 2000* recommendation.

colour.difference.delta_E_CIE2000

colour.difference.**delta_E_CIE2000**(*Lab_1: ArrayLike, Lab_2: ArrayLike, textiles: bool = False*) → NDArrayFloat

Return the difference ΔE_{00} between two given *CIE L*a*b** colourspace arrays using *CIE 2000* recommendation.

Parameters

- **Lab_1** (ArrayLike) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (ArrayLike) – *CIE L*a*b** colourspace array 2.
- **textiles** (bool) – Textiles application specific parametric factors. $k_L = 2, \ k_C = k_H = 1$ weights are used instead of $k_L = k_C = k_H = 1$.

Returns Colour difference ΔE_{00}.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

- Parametric factors $k_L = k_C = k_H = 1$ weights under *reference conditions*:
  - Illumination: D65 source
  - Illuminance: 1000 lx
  - Observer: Normal colour vision
  - Background field: Uniform, neutral gray with $L^* = 50$
  - Viewing mode: Object
  - Sample size: Greater than 4 degrees
  - Sample separation: Direct edge contact
  - Sample colour-difference magnitude: Lower than 5.0 ΔE_{00}
  - Sample structure: Homogeneous (without texture)

References

[Mel13], [SWD05]

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])	Return the difference ΔE_{CMC} between two given *CIE L*a*b** colourspace arrays using *Colour Measurement Committee* recommendation.

colour.difference.delta_E_CMC

colour.difference.**delta_E_CMC**(*Lab_1: ArrayLike*, *Lab_2: ArrayLike*, *l: float = 2*, *c: float = 1*) → NDArrayFloat

Return the difference ΔE_{CMC} between two given *CIE L*a*b** colourspace arrays using *Colour Measurement Committee* recommendation.

The quasimetric has two parameters: *Lightness* (l) and *chroma* (c), allowing the users to weight the difference based on the ratio of l:c. Commonly used values are 2:1 for acceptability and 1:1 for the threshold of imperceptibility.

Parameters

- **Lab_1** (ArrayLike) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (ArrayLike) – *CIE L*a*b** colourspace array 2.
- **l** (float) – Lightness weighting factor.
- **c** (float) – Chroma weighting factor.

Returns Colour difference ΔE_{CMC}.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

References

[Lin09b]

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...
```

ITP

colour.difference

delta_E_ITP(ICtCp_1, ICtCp_2)	Return the difference ΔE_{ITP} between two given IC_TC_P colour encoding arrays using *Recommendation ITU-R BT.2124*.

colour.difference.delta_E_ITP

colour.difference.**delta_E_ITP**(*ICtCp_1: ArrayLike*, *ICtCp_2: ArrayLike*) → NDArrayFloat

Return the difference ΔE_{ITP} between two given IC_TC_P colour encoding arrays using *Recommendation ITU-R BT.2124*.

Parameters

- **ICtCp_1** (ArrayLike) – IC_TC_P colour encoding array 1.
- **ICtCp_2** (ArrayLike) – IC_TC_P colour encoding array 2.

Returns Colour difference ΔE_{ITP}.

Return type numpy.ndarray

Notes

- A value of 1 is equivalent to a just noticeable difference when viewed in the most critical adaptation state.

References

[InternationalTUnion19]

Examples

```
>>> ICtCp_1 = np.array([0.4885468072, -0.04739350675, 0.07475401302])
>>> ICtCp_2 = np.array([0.4899203231, -0.04567508203, 0.07361341775])
>>> delta_E_ITP(ICtCp_1, ICtCp_2)
1.42657228...
```

Luo, Cui and Li (2006)

colour.difference

delta_E_CAM02LCD(Jpapbp_1, Jpapbp_2)	Return the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-LCD* colourspaces $J'a'b'$ arrays.
delta_E_CAM02SCD(Jpapbp_1, Jpapbp_2)	Return the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-SCD* colourspaces $J'a'b'$ arrays.
delta_E_CAM02UCS(Jpapbp_1, Jpapbp_2)	Return the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-UCS* colourspaces $J'a'b'$ arrays.

colour.difference.delta_E_CAM02LCD

`colour.difference.delta_E_CAM02LCD`(*Jpapbp_1: ArrayLike*, *Jpapbp_2: ArrayLike*) → NDArrayFloat

Return the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-LCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (`ArrayLike`) – Standard / reference *Luo et al. (2006) CAM02-LCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (`ArrayLike`) – Sample / test *Luo et al. (2006) CAM02-LCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type `numpy.ndarray`

> **Warning:** The $J'a'b'$ array should have been computed with a *Luo et al. (2006) CAM02-LCD*, *CAM02-SCD*, or *CAM02-UCS* colourspace and not with the *CIE L*a*b** colourspace.

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LCL06]

Examples

```
>>> Jpapbp_1 = np.array([54.90433134, -0.08450395, -0.06854831])
>>> Jpapbp_2 = np.array([54.80352754, -3.96940084, -13.57591013])
>>> delta_E_CAM02LCD(Jpapbp_1, Jpapbp_2)
14.0555464...
```

colour.difference.delta_E_CAM02SCD

`colour.difference.delta_E_CAM02SCD`(*Jpapbp_1: ArrayLike*, *Jpapbp_2: ArrayLike*) → NDArrayFloat

Return the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-SCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (`ArrayLike`) – Standard / reference *Luo et al. (2006) CAM02-SCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (`ArrayLike`) – Sample / test *Luo et al. (2006) CAM02-SCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type `numpy.ndarray`

> **Warning:** The $J'a'b'$ array should have been computed with a *Luo et al. (2006) CAM02-LCD*, *CAM02-SCD*, or *CAM02-UCS* colourspace and not with the *CIE L*a*b** colourspace.

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LCL06]

Examples

```
>>> Jpapbp_1 = np.array([54.90433134, -0.08450395, -0.06854831])
>>> Jpapbp_2 = np.array([54.80352754, -3.96940084, -13.57591013])
>>> delta_E_CAM02SCD(Jpapbp_1, Jpapbp_2)
14.0551718...
```

colour.difference.delta_E_CAM02UCS

`colour.difference.`**`delta_E_CAM02UCS`**(*Jpapbp_1: ArrayLike*, *Jpapbp_2: ArrayLike*) → NDArrayFloat

Return the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-UCS* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (`ArrayLike`) – Standard / reference *Luo et al. (2006) CAM02-UCS* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (`ArrayLike`) – Sample / test *Luo et al. (2006) CAM02-UCS* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type `numpy.ndarray`

> **Warning:** The $J'a'b'$ array should have been computed with a *Luo et al. (2006) CAM02-LCD*, *CAM02-SCD*, or *CAM02-UCS* colourspace and not with the *CIE L*a*b** colourspace.

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LCL06]

Examples

```
>>> Jpapbp_1 = np.array([54.90433134, -0.08450395, -0.06854831])
>>> Jpapbp_2 = np.array([54.80352754, -3.96940084, -13.57591013])
>>> delta_E_CAM02UCS(Jpapbp_1, Jpapbp_2)
14.0552982...
```

Li, Li, Wang, Zu, Luo, Cui, Melgosa, Brill and Pointer (2017)

`colour.difference`

`delta_E_CAM16LCD`(Jpapbp_1, Jpapbp_2)	Return the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ arrays.
`delta_E_CAM16SCD`(Jpapbp_1, Jpapbp_2)	Return the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ arrays.
`delta_E_CAM16UCS`(Jpapbp_1, Jpapbp_2)	Return the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ arrays.

colour.difference.delta_E_CAM16LCD

`colour.difference.`**`delta_E_CAM16LCD`**(*Jpapbp_1*, *Jpapbp_2*)

Return the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** – Standard / reference *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** – Sample / test *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type `numpy.ndarray`

> **Warning:** The $J'a'b'$ array should have been computed with a *Li et al. (2017) CAM16-LCD*, *CAM16-SCD*, or *CAM16-UCS* colourspace and not with the *CIE L*a*b** colourspace.

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

colour.difference.delta_E_CAM16SCD

`colour.difference.delta_E_CAM16SCD(Jpapbp_1, Jpapbp_2)`

Return the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** – Standard / reference *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** – Sample / test *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type `numpy.ndarray`

> **Warning:** The $J'a'b'$ array should have been computed with a *Li et al. (2017) CAM16-LCD*, *CAM16-SCD*, or *CAM16-UCS* colourspace and not with the *CIE L*a*b** colourspace.

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

colour.difference.delta_E_CAM16UCS

colour.difference.**delta_E_CAM16UCS**(*Jpapbp_1*, *Jpapbp_2*)

Return the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** – Standard / reference *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.
- **Jpapbp_2** – Sample / test *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numpy.ndarray

Warning: The $J'a'b'$ array should have been computed with a *Li et al. (2017) CAM16-LCD*, *CAM16-SCD*, or *CAM16-UCS* colourspace and not with the *CIE L*a*b** colourspace.

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

DIN99

colour.difference

delta_E_DIN99(Lab_1, Lab_2[, textiles])	Return the difference ΔE_{DIN99} between two given *CIE L*a*b** colourspace arrays using *DIN99* formula.

colour.difference.delta_E_DIN99

`colour.difference.delta_E_DIN99`(*Lab_1: ArrayLike*, *Lab_2: ArrayLike*, *textiles: bool = False*) → NDArrayFloat

Return the difference ΔE_{DIN99} between two given *CIE L*a*b** colourspace arrays using *DIN99* formula.

Parameters

- **Lab_1** (ArrayLike) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (ArrayLike) – *CIE L*a*b** colourspace array 2.
- **textiles** (bool) – Textiles application specific parametric factors, $k_E = 2,\ k_{CH} = 0.5$ weights are used instead of $k_E = 1,\ k_{CH} = 1$.

Returns Colour difference ΔE_{DIN99}.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

References

[ASTMInternational07]

Examples

```
>>> import numpy as np
>>> Lab_1 = np.array([60.2574, -34.0099, 36.2677])
>>> Lab_2 = np.array([60.4626, -34.1751, 39.4387])
>>> delta_E_DIN99(Lab_1, Lab_2)
1.1772166...
```

Standardized Residual Sum of Squares (STRESS) Index

`colour`

index_stress(d_E, d_V[, method])	Compute the *Kruskal's Standardized Residual Sum of Squares (:math:`STRESS`)* index according to given method.
INDEX_STRESS_METHODS	Supported $STRESS$ index computation methods.

colour.index_stress

colour.**index_stress**(*d_E: ArrayLike, d_V: ArrayLike, method: Union[Literal['Garcia 2007'], str] = 'Garcia 2007'*) → NDArrayFloat

Compute the *Kruskal's Standardized Residual Sum of Squares (:math: `STRESS`)* index according to given method.

Parameters

- **d_E** (ArrayLike) – Computed colour difference array ΔE.
- **d_V** (ArrayLike) – Computed colour difference array ΔV.
- **method** (Union[Literal['Garcia 2007'], str]) – Computation method.

Returns $STRESS$ index.

Return type numpy.ndarray

References

[GarciaHMC07]

Examples

```
>>> d_E = np.array([2.0425, 2.8615, 3.4412])
>>> d_V = np.array([1.2644, 1.2630, 1.8731])
>>> index_stress(d_E, d_V)
0.1211709...
```

colour.INDEX_STRESS_METHODS

colour.**INDEX_STRESS_METHODS = CanonicalMapping({'Garcia 2007': ...})**

Supported $STRESS$ index computation methods.

References

[GarciaHMC07]

colour.difference

index_stress_Garcia2007(d_E, d_V)	Compute the *Kruskal's Standardized Residual Sum of Squares (:math: `STRESS`)* index according to *Garci?a, Huertas, Melgosa and Cui (2007)* method.

colour.difference.index_stress_Garcia2007

colour.difference.**index_stress_Garcia2007**(*d_E: ArrayLike*, *d_V: ArrayLike*) → NDArrayFloat

Compute the *Kruskal's Standardized Residual Sum of Squares (:math: `STRESS `)* index according to *Garci?a, Huertas, Melgosa and Cui (2007)* method.

Parameters

- **d_E** (ArrayLike) – Computed colour difference array ΔE.
- **d_V** (ArrayLike) – Computed colour difference array ΔV.

Returns $STRESS$ index.

Return type numpy.ndarray

References

[GarciaHMC07]

Examples

```
>>> d_E = np.array([2.0425, 2.8615, 3.4412])
>>> d_V = np.array([1.2644, 1.2630, 1.8731])
>>> index_stress_Garcia2007(d_E, d_V)
0.1211709...
```

Huang et al. (2015) Power-Functions

colour.difference

power_function_Huang2015(d_E[, coefficients])	Improve the performance of the ΔE value for given coefficients using *Huang, Cui, Melgosa, Sanchez-Maranon, Li, Luo and Liu (2015)* power-function: $d'_E = a * d_{E^b}$.

colour.difference.power_function_Huang2015

colour.difference.**power_function_Huang2015**(*d_E: ArrayLike*, *coefficients: Union[Literal['CIE 1976', 'CIE 1994', 'CIE 2000', 'CMC', 'CAM02-LCD', 'CAM02-SCD', 'CAM16-UCS', 'DIN99d', 'OSA', 'OSA-GP-Euclidean', 'ULAB'], str] = 'CIE 2000'*) → NDArrayFloat

Improve the performance of the ΔE value for given coefficients using *Huang, Cui, Melgosa, Sanchez-Maranon, Li, Luo and Liu (2015)* power-function: $d'_E = a * d_{E^b}$.

Parameters

- **d_E** (ArrayLike) – Computed colour difference array ΔE.
- **coefficients** (Union[Literal['CIE 1976', 'CIE 1994', 'CIE 2000', 'CMC', 'CAM02-LCD', 'CAM02-SCD', 'CAM16-UCS', 'DIN99d', 'OSA', 'OSA-GP-Euclidean', 'ULAB'], str]) – Coefficients for the power-function.

Returns Improved math:*Delta E* value.

Return type numpy.ndarray

References

[HCM+15], [LLW+17]

Examples

```
>>> d_E = np.array([2.0425, 2.8615, 3.4412])
>>> power_function_Huang2015(d_E)
array([ 2.3574879...,  2.9850503...,  3.3965106...])
```

Geometry Computations

Ellipse

`colour.geometry`

`ellipse_coefficients_general_form`(coefficients)	Return the general form ellipse coefficients from given canonical form ellipse coefficients.
`ellipse_coefficients_canonical_form`(coefficients)	Return the canonical form ellipse coefficients from given general form ellipse coefficients.
`point_at_angle_on_ellipse`(phi, coefficients)	Return the coordinates of the point at angle ϕ in degrees on the ellipse with given canonical form coefficients.
`ELLIPSE_FITTING_METHODS`	Supported ellipse fitting methods.
`ellipse_fitting`(a[, method])	Return the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using given method.

colour.geometry.ellipse_coefficients_general_form

`colour.geometry.`**`ellipse_coefficients_general_form`**(*coefficients: ArrayLike*) → NDArrayFloat

Return the general form ellipse coefficients from given canonical form ellipse coefficients.

The canonical form ellipse coefficients are as follows: the center coordinates x_c and y_c, semi-major axis length a_a, semi-minor axis length a_b and rotation angle θ in degrees of its semi-major axis a_a.

Parameters **`coefficients`** (`ArrayLike`) – Canonical form ellipse coefficients.

Returns General form ellipse coefficients.

Return type `numpy.ndarray`

References

[Wikipedia]

Examples

```
>>> coefficients = np.array([0.5, 0.5, 2, 1, 45])
>>> ellipse_coefficients_general_form(coefficients)
array([ 2.5, -3. ,  2.5, -1. , -1. , -3.5])
```

colour.geometry.ellipse_coefficients_canonical_form

colour.geometry.**ellipse_coefficients_canonical_form**(*coefficients: ArrayLike*) → NDArrayFloat

Return the canonical form ellipse coefficients from given general form ellipse coefficients.

The general form ellipse coefficients are the coefficients of the implicit second-order polynomial/quadratic curve expressed as follows:

$F(x,y)$ = ax^2 + bxy + cy^2 + dx + ey + f = 0`

with an ellipse-specific constraint such as $b^2 - 4ac < 0$ and where a, b, c, d, e, f are coefficients of the ellipse and $F(x,y)$ are coordinates of points lying on it.

Parameters **coefficients** (ArrayLike) – General form ellipse coefficients.

Returns Canonical form ellipse coefficients.

Return type numpy.ndarray

References

[Wikipedia]

Examples

```
>>> coefficients = np.array([2.5, -3.0, 2.5, -1.0, -1.0, -3.5])
>>> ellipse_coefficients_canonical_form(coefficients)
array([  0.5,   0.5,   2. ,   1. ,  45. ])
```

colour.geometry.point_at_angle_on_ellipse

colour.geometry.**point_at_angle_on_ellipse**(*phi: ArrayLike*, *coefficients: ArrayLike*) → NDArrayFloat

Return the coordinates of the point at angle ϕ in degrees on the ellipse with given canonical form coefficients.

Parameters

- **phi** (ArrayLike) – Point at angle ϕ in degrees to retrieve the coordinates of.
- **coefficients** (ArrayLike) – General form ellipse coefficients as follows: the center coordinates x_c and y_c, semi-major axis length a_a, semi-minor axis length a_b and rotation angle θ in degrees of its semi-major axis a_a.

Returns Coordinates of the point at angle ϕ

Return type numpy.ndarray

Examples

```
>>> coefficients = np.array([0.5, 0.5, 2, 1, 45])
>>> point_at_angle_on_ellipse(45, coefficients)
array([ 1.,  2.])
```

colour.geometry.ELLIPSE_FITTING_METHODS

colour.geometry.**ELLIPSE_FITTING_METHODS = CanonicalMapping({'Halir 1998': ...})**

Supported ellipse fitting methods.

References

[HF98]

colour.geometry.ellipse_fitting

colour.geometry.**ellipse_fitting**(*a: ArrayLike, method: Union[Literal['Halir 1998'], str] = 'Halir 1998'*) → NDArrayFloat

Return the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using given method.

The implicit second-order polynomial is expressed as follows:

$F(x, y)$ = ax^2 + bxy + cy^2 + dx + ey + f = 0`

with an ellipse-specific constraint such as $b^2 - 4ac < 0$ and where a, b, c, d, e, f are coefficients of the ellipse and $F(x, y)$ are coordinates of points lying on it.

Parameters

- **a** (ArrayLike) – Point array a to be fitted.
- **method** (Union[Literal['Halir 1998'], str]) – Computation method.

Returns Coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a.

Return type numpy.ndarray

References

[HF98]

Examples

```
>>> a = np.array([[2, 0], [0, 1], [-2, 0], [0, -1]])
>>> ellipse_fitting(a)
array([ 0.2425356...,  0.        ,  0.9701425...,  0.        ,  0.        ,
       -0.9701425...])
>>> ellipse_coefficients_canonical_form(ellipse_fitting(a))
array([-0., -0.,  2.,  1.,  0.])
```

Ancillary Objects

colour.geometry

`ellipse_fitting_Halir1998`(a)	Return the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using *Halir and Flusser (1998)* method.

colour.geometry.ellipse_fitting_Halir1998

colour.geometry.**ellipse_fitting_Halir1998**(*a: ArrayLike*) → NDArrayFloat

Return the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using *Halir and Flusser (1998)* method.

The implicit second-order polynomial is expressed as follows:

$F(x, y)$ = ax^2 + bxy + cy^2 + dx + ey + f = 0`

with an ellipse-specific constraint such as $b^2 - 4ac < 0$ and where a, b, c, d, e, f are coefficients of the ellipse and $F(x, y)$ are coordinates of points lying on it.

Parameters a (ArrayLike) – Point array a to be fitted.

Returns Coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a.

Return type numpy.ndarray

References

[HF98]

Examples

```
>>> a = np.array([[2, 0], [0, 1], [-2, 0], [0, -1]])
>>> ellipse_fitting_Halir1998(a)
array([ 0.2425356...,  0.        ,  0.9701425...,  0.        ,  0.        ,
       -0.9701425...])
>>> ellipse_coefficients_canonical_form(ellipse_fitting_Halir1998(a))
array([-0., -0.,  2.,  1.,  0.])
```

Intersection

colour.geometry

`extend_line_segment`(a, b[, distance])	Extend 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)	Compute l_1 line segments intersections with l_2 line segments.

colour.geometry.extend_line_segment

colour.geometry.**extend_line_segment**(*a: ArrayLike, b: ArrayLike, distance: float = 1*) → NDArrayFloat

Extend the line segment defined by point arrays a and b by given distance and return the new end point.

Parameters

- **a** (ArrayLike) – Point array a.
- **b** (ArrayLike) – Point array b.
- **distance** (float) – Distance to extend the line segment.

Returns New end point.

Return type numpy.ndarray

References

[Saeedn]

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.geometry.intersect_line_segments

colour.geometry.**intersect_line_segments**(*l_1: ArrayLike, l_2: ArrayLike*) → *colour.geometry.intersection.LineSegmentsIntersections_Specification*

Compute l_1 line segments intersections with l_2 line segments.

Parameters

- **l_1** (ArrayLike) – 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** (ArrayLike) – 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 colour.algebra.LineSegmentsIntersections_Specification

References

[Boua], [Erda]

Notes

- Input line segments points coordinates are 2d coordinates.

Examples

```
>>> l_1 = np.array(
...     [
...         [[0.15416284, 0.7400497], [0.26331502, 0.53373939]],
...         [[0.01457496, 0.91874701], [0.90071485, 0.03342143]],
...     ]
... )
>>> l_2 = np.array(
...     [
...         [[0.95694934, 0.13720932], [0.28382835, 0.60608318]],
...         [[0.94422514, 0.85273554], [0.00225923, 0.52122603]],
...         [[0.55203763, 0.48537741], [0.76813415, 0.16071675]],
...     ]
... )
>>> s = intersect_line_segments(l_1, l_2)
>>> s.xy
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.geometry

LineSegmentsIntersections_Specification(xy, ...)	Define the specification for intersection of line segments l_1 and l_2 returned by colour.algebra.intersect_line_segments() definition.

colour.geometry.LineSegmentsIntersections_Specification

class colour.geometry.**LineSegmentsIntersections_Specification**(*xy: NDArrayFloat*, *intersect: NDArrayFloat*, *parallel: NDArrayFloat*, *coincident: NDArrayFloat*)

Define the specification for intersection of line segments l_1 and l_2 returned by `colour.algebra.intersect_line_segments()` definition.

Parameters

- **xy** (NDArrayFloat) – Array of l_1 and l_2 line segments intersections coordinates. Non existing segments intersections coordinates are set with *np.nan*.
- **intersect** (NDArrayFloat) – Array of *bool* indicating if line segments l_1 and l_2 intersect.
- **parallel** (NDArrayFloat) – Array of `bool` indicating if line segments l_1 and l_2 are parallel.
- **coincident** (NDArrayFloat) – Array of `bool` indicating if line segments l_1 and l_2 are coincident.

Return type None

__init__(*xy: NDArrayFloat*, *intersect: NDArrayFloat*, *parallel: NDArrayFloat*, *coincident: NDArrayFloat*) → None

Parameters

- **xy** (NDArrayFloat) –
- **intersect** (NDArrayFloat) –
- **parallel** (NDArrayFloat) –
- **coincident** (NDArrayFloat) –

Return type None

Methods

`__init__`(xy, intersect, parallel, coincident)	

Attributes

`xy`
`intersect`
`parallel`
`coincident`

Primitives

colour

PRIMITIVE_METHODS	Supported geometry primitive generation methods.
primitive([method])	Return a geometry primitive using given method.

colour.PRIMITIVE_METHODS

colour.**PRIMITIVE_METHODS = CanonicalMapping({'Grid': ..., 'Cube': ...})**

Supported geometry primitive generation methods.

colour.primitive

colour.**primitive**(*method: Union[Literal['Cube', 'Grid'], str] = 'Cube', **kwargs: Any*) → Tuple[numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]], numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]], numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]]

Return a geometry primitive using given method.

Parameters

- **method** (Union[Literal['Cube', 'Grid'], str]) – Generation method.
- **axis** – {colour.geometry.primitive_grid()}, Axis the primitive will be normal to, or plane the primitive will be co-planar with.
- **depth** – {colour.geometry.primitive_grid(), colour.geometry.primitive_cube()}, Primitive depth.
- **depth_segments** – {colour.geometry.primitive_grid(), colour.geometry.primitive_cube()}, Primitive segments count along the depth.
- **dtype_indexes** – {colour.geometry.primitive_grid(), colour.geometry.primitive_cube()}, numpy.dtype to use for the grid indexes, default to the numpy.dtype defined by the colour.constant.DEFAULT_INT_DTYPE attribute.
- **dtype_vertices** – {colour.geometry.primitive_grid(), colour.geometry.primitive_cube()}, numpy.dtype to use for the grid vertices, default to the numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.
- **height** – {colour.geometry.primitive_grid(), colour.geometry.primitive_cube()}, Primitive height.
- **planes** – {colour.geometry.primitive_cube()}, Included grid primitives in the cube construction.
- **width** – {colour.geometry.primitive_grid(), colour.geometry.primitive_cube()}, Primitive width.
- **width_segments** – {colour.geometry.primitive_grid(), colour.geometry.primitive_cube()}, Primitive segments count along the width.
- **height_segments** – {colour.geometry.primitive_grid(), colour.geometry.primitive_cube()}, Primitive segments count along the height.
- **kwargs** (Any) –

Returns Tuple of primitive vertices, face indexes to produce a filled primitive and outline indexes to produce an outline of the faces of the primitive.

Return type tuple

References

[Cab]

Examples

```
>>> vertices, faces, outline = primitive()
>>> print(vertices)
[([-0.5,  0.5, -0.5], [ 0.,  1.], [-0., -0., -1.], [ 0.,  1.,  0.,  1.])
 ([ 0.5,  0.5, -0.5], [ 1.,  1.], [-0., -0., -1.], [ 1.,  1.,  0.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-0., -0., -1.], [ 0.,  0.,  0.,  1.])
 ([ 0.5, -0.5, -0.5], [ 1.,  0.], [-0., -0., -1.], [ 1.,  0.,  0.,  1.])
 ([-0.5,  0.5,  0.5], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  1.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  1.,  1.])
 ([-0.5, -0.5,  0.5], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  1.,  1.])
 ([ 0.5, -0.5,  0.5], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  1.,  1.])
 ([ 0.5, -0.5, -0.5], [ 0.,  1.], [-0., -1., -0.], [ 1.,  0.,  0.,  1.])
 ([ 0.5, -0.5,  0.5], [ 1.,  1.], [-0., -1., -0.], [ 1.,  0.,  1.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-0., -1., -0.], [ 0.,  0.,  0.,  1.])
 ([-0.5, -0.5,  0.5], [ 1.,  0.], [-0., -1., -0.], [ 0.,  0.,  1.,  1.])
 ([ 0.5,  0.5, -0.5], [ 0.,  1.], [ 0.,  1.,  0.], [ 1.,  1.,  0.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 0.,  1.,  0.], [ 1.,  1.,  1.,  1.])
 ([-0.5,  0.5, -0.5], [ 0.,  0.], [ 0.,  1.,  0.], [ 0.,  1.,  0.,  1.])
 ([-0.5,  0.5,  0.5], [ 1.,  0.], [ 0.,  1.,  0.], [ 0.,  1.,  1.,  1.])
 ([-0.5, -0.5,  0.5], [ 0.,  1.], [-1., -0., -0.], [ 0.,  0.,  1.,  1.])
 ([-0.5,  0.5,  0.5], [ 1.,  1.], [-1., -0., -0.], [ 0.,  1.,  1.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-1., -0., -0.], [ 0.,  0.,  0.,  1.])
 ([-0.5,  0.5, -0.5], [ 1.,  0.], [-1., -0., -0.], [ 0.,  1.,  0.,  1.])
 ([ 0.5, -0.5,  0.5], [ 0.,  1.], [ 1.,  0.,  0.], [ 1.,  0.,  1.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 1.,  0.,  0.], [ 1.,  1.,  1.,  1.])
 ([ 0.5, -0.5, -0.5], [ 0.,  0.], [ 1.,  0.,  0.], [ 1.,  0.,  0.,  1.])
 ([ 0.5,  0.5, -0.5], [ 1.,  0.], [ 1.,  0.,  0.], [ 1.,  1.,  0.,  1.])]
>>> print(faces)
[[ 1  2  0]
 [ 1  3  2]
 [ 4  6  5]
 [ 6  7  5]
 [ 9 10  8]
 [ 9 11 10]
 [12 14 13]
 [14 15 13]
 [17 18 16]
 [17 19 18]
 [20 22 21]
 [22 23 21]]
>>> print(outline)
[[ 0  2]
 [ 2  3]
 [ 3  1]
 [ 1  0]
 [ 4  6]
 [ 6  7]
 [ 7  5]
 [ 5  4]
```

(continues on next page)

(continued from previous page)

```
 [ 8 10]
 [10 11]
 [11  9]
 [ 9  8]
 [12 14]
 [14 15]
 [15 13]
 [13 12]
 [16 18]
 [18 19]
 [19 17]
 [17 16]
 [20 22]
 [22 23]
 [23 21]
 [21 20]]
>>> vertices, faces, outline = primitive("Grid")
>>> print(vertices)
[([-0.5,  0.5,  0. ], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  0.,  1.])
 ([ 0.5,  0.5,  0. ], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  0.,  1.])
 ([-0.5, -0.5,  0. ], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  0.,  1.])
 ([ 0.5, -0.5,  0. ], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  0.,  1.])]
>>> print(faces)
[[0 2 1]
 [2 3 1]]
>>> print(outline)
[[0 2]
 [2 3]
 [3 1]
 [1 0]]
```

Ancillary Objects

`colour.geometry`

MAPPING_PLANE_TO_AXIS	Plane to axis mapping.
primitive_grid([width, height, ...])	Generate vertices and indexes for a filled and outlined grid primitive.
primitive_cube([width, height, depth, ...])	Generate vertices and indexes for a filled and outlined cube primitive.

colour.geometry.MAPPING_PLANE_TO_AXIS

`colour.geometry.MAPPING_PLANE_TO_AXIS = CanonicalMapping({'yz': ..., 'zy': ..., 'xz': ..., 'zx': ..., 'xy': ..., 'yx': ...})`

Plane to axis mapping.

colour.geometry.primitive_grid

colour.geometry.**primitive_grid**(*width: float = 1, height: float = 1, width_segments: int = 1, height_segments: int = 1, axis: Literal['-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy'] = '+z', dtype_vertices: Type[DTypeFloat] | None = None, dtype_indexes: Type[DTypeInt] | None = None*) → Tuple[NDArray, NDArray, NDArray]

Generate vertices and indexes for a filled and outlined grid primitive.

Parameters

- **width** (float) – Grid width.
- **height** (float) – Grid height.
- **width_segments** (int) – Grid segments count along the width.
- **height_segments** (int) – Grid segments count along the height.
- **axis** (Literal['-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy']) – Axis the primitive will be normal to, or plane the primitive will be co-planar with.
- **dtype_vertices** (Type[DTypeFloat] | None) – numpy.dtype to use for the grid vertices, default to the numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.
- **dtype_indexes** (Type[DTypeInt] | None) – numpy.dtype to use for the grid indexes, default to the numpy.dtype defined by the colour.constant.DEFAULT_INT_DTYPE attribute.

Returns Tuple of grid vertices, face indexes to produce a filled grid and outline indexes to produce an outline of the faces of the grid.

Return type tuple

References

[Cab]

Examples

```
>>> vertices, faces, outline = primitive_grid()
>>> print(vertices)
[([-0.5,  0.5,  0. ], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  0.,  1.])
 ([ 0.5,  0.5,  0. ], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  0.,  1.])
 ([-0.5, -0.5,  0. ], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  0.,  1.])
 ([ 0.5, -0.5,  0. ], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  0.,  1.])]
>>> print(faces)
[[0 2 1]
 [2 3 1]]
>>> print(outline)
[[0 2]
 [2 3]
 [3 1]
 [1 0]]
```

colour.geometry.primitive_cube

colour.geometry.**primitive_cube**(*width: float = 1, height: float = 1, depth: float = 1, width_segments: int = 1, height_segments: int = 1, depth_segments: int = 1, planes: Literal['-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy'] | None = None, dtype_vertices: Type[DTypeFloat] | None = None, dtype_indexes: Type[DTypeInt] | None = None*) → Tuple[NDArray, NDArray, NDArray]

Generate vertices and indexes for a filled and outlined cube primitive.

Parameters

- **width** (float) – Cube width.
- **height** (float) – Cube height.
- **depth** (float) – Cube depth.
- **width_segments** (int) – Cube segments count along the width.
- **height_segments** (int) – Cube segments count along the height.
- **depth_segments** (int) – Cube segments count along the depth.
- **planes** (Literal['-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy'] | None) – Grid primitives to include in the cube construction.
- **dtype_vertices** (Type[DTypeFloat] | None) – numpy.dtype to use for the grid vertices, default to the numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.
- **dtype_indexes** (Type[DTypeInt] | None) – numpy.dtype to use for the grid indexes, default to the numpy.dtype defined by the colour.constant.DEFAULT_INT_DTYPE attribute.

Returns Tuple of cube vertices, face indexes to produce a filled cube and outline indexes to produce an outline of the faces of the cube.

Return type tuple

Examples

```
>>> vertices, faces, outline = primitive_cube()
>>> print(vertices)
[([-0.5,  0.5, -0.5], [ 0.,  1.], [-0., -0., -1.], [ 0.,  1.,  0.,  1.])
 ([ 0.5,  0.5, -0.5], [ 1.,  1.], [-0., -0., -1.], [ 1.,  1.,  0.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-0., -0., -1.], [ 0.,  0.,  0.,  1.])
 ([ 0.5, -0.5, -0.5], [ 1.,  0.], [-0., -0., -1.], [ 1.,  0.,  0.,  1.])
 ([-0.5,  0.5,  0.5], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  1.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  1.,  1.])
 ([-0.5, -0.5,  0.5], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  1.,  1.])
 ([ 0.5, -0.5,  0.5], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  1.,  1.])
 ([ 0.5, -0.5, -0.5], [ 0.,  1.], [-0., -1., -0.], [ 1.,  0.,  0.,  1.])
 ([ 0.5, -0.5,  0.5], [ 1.,  1.], [-0., -1., -0.], [ 1.,  0.,  1.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-0., -1., -0.], [ 0.,  0.,  0.,  1.])
 ([-0.5, -0.5,  0.5], [ 1.,  0.], [-0., -1., -0.], [ 0.,  0.,  1.,  1.])
 ([ 0.5,  0.5, -0.5], [ 0.,  1.], [ 0.,  1.,  0.], [ 1.,  1.,  0.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 0.,  1.,  0.], [ 1.,  1.,  1.,  1.])
 ([-0.5,  0.5, -0.5], [ 0.,  0.], [ 0.,  1.,  0.], [ 0.,  1.,  0.,  1.])
 ([-0.5,  0.5,  0.5], [ 1.,  0.], [ 0.,  1.,  0.], [ 0.,  1.,  1.,  1.])
 ([-0.5, -0.5,  0.5], [ 0.,  1.], [-1., -0., -0.], [ 0.,  0.,  1.,  1.])
 ([-0.5,  0.5,  0.5], [ 1.,  1.], [-1., -0., -0.], [ 0.,  1.,  1.,  1.])
```

(continues on next page)

(continued from previous page)

```
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-1., -0., -0.], [ 0.,  0.,  0.,  1.])
 ([-0.5,  0.5, -0.5], [ 1.,  0.], [-1., -0., -0.], [ 0.,  1.,  0.,  1.])
 ([ 0.5, -0.5,  0.5], [ 0.,  1.], [ 1.,  0.,  0.], [ 1.,  0.,  1.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 1.,  0.,  0.], [ 1.,  1.,  1.,  1.])
 ([ 0.5, -0.5, -0.5], [ 0.,  0.], [ 1.,  0.,  0.], [ 1.,  0.,  0.,  1.])
 ([ 0.5,  0.5, -0.5], [ 1.,  0.], [ 1.,  0.,  0.], [ 1.,  1.,  0.,  1.])]
>>> print(faces)
[[ 1  2  0]
 [ 1  3  2]
 [ 4  6  5]
 [ 6  7  5]
 [ 9 10  8]
 [ 9 11 10]
 [12 14 13]
 [14 15 13]
 [17 18 16]
 [17 19 18]
 [20 22 21]
 [22 23 21]]
>>> print(outline)
[[ 0  2]
 [ 2  3]
 [ 3  1]
 [ 1  0]
 [ 4  6]
 [ 6  7]
 [ 7  5]
 [ 5  4]
 [ 8 10]
 [10 11]
 [11  9]
 [ 9  8]
 [12 14]
 [14 15]
 [15 13]
 [13 12]
 [16 18]
 [18 19]
 [19 17]
 [17 16]
 [20 22]
 [22 23]
 [23 21]
 [21 20]]
```

Primitive Vertices

colour

PRIMITIVE_VERTICES_METHODS	Supported geometry primitive vertices generation methods.
primitive_vertices([method])	Return the vertices of a geometry primitive using given method.

colour.PRIMITIVE_VERTICES_METHODS

colour.**PRIMITIVE_VERTICES_METHODS = CanonicalMapping({'Quad MPL': ..., 'Grid MPL': ..., 'Cube MPL': ..., 'Sphere': ...})**

Supported geometry primitive vertices generation methods.

colour.primitive_vertices

colour.**primitive_vertices**(*method: Union[Literal['Cube MPL', 'Quad MPL', 'Grid MPL', 'Sphere'], str] = 'Cube MPL', **kwargs: Any*) → NDArrayFloat

Return the vertices of a geometry primitive using given method.

Parameters

- **method** (Union[Literal['Cube MPL', 'Quad MPL', 'Grid MPL', 'Sphere'], str]) – Vertices generation method.
- **axis** – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_sphere()}, **{'+z', '+x', '+y', 'yz', 'xz', 'xy'}**, Axis the primitive will be normal to, or plane the primitive will be co-planar with.
- **depth** – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive depth.
- **depth_segments** – {colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive depth segments, quad primitive counts along the depth.
- **height** – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive height.
- **height_segments** – {colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive height segments, quad primitive counts along the height.
- **intermediate** – {colour.geometry.primitive_vertices_sphere()}, Whether to generate the sphere vertices at the center of the faces outlined by the segments of a regular sphere generated without the intermediate argument set to *True*. The resulting sphere is inscribed on the regular sphere faces but possesses the same poles.
- **origin** – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl(), colour.geometry.primitive_vertices_sphere()}, Primitive origin on the construction plane.

- **planes** – {colour.geometry.primitive_vertices_cube_mpl()}, {'-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy'}, Included grid primitives in the cube construction.
- **radius** – {colour.geometry.primitive_vertices_sphere()}, Sphere radius.
- **segments** – {colour.geometry.primitive_vertices_sphere()}, Latitude-longitude segments, if the intermediate argument is *True*, then the sphere will have one less segment along its longitude.
- **width** – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive width.
- **width_segments** – {colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive width segments, quad primitive counts along the width.
- **kwargs** (Any) –

Returns Primitive vertices.

Return type numpy.ndarray

Examples

```
>>> primitive_vertices()
array([[[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
        [ 0.,  1.,  0.]],

       [[ 0.,  0.,  1.],
        [ 1.,  0.,  1.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  0.,  1.],
        [ 0.,  0.,  1.]],

       [[ 0.,  1.,  0.],
        [ 1.,  1.,  0.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 0.,  1.,  0.],
        [ 0.,  1.,  1.],
        [ 0.,  0.,  1.]],

       [[ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
        [ 1.,  1.,  1.],
        [ 1.,  0.,  1.]]])
>>> primitive_vertices("Quad MPL")
array([[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  0.],
```

(continues on next page)

(continued from previous page)

```
       [ 0.,   1.,   0.]])
>>> primitive_vertices("Sphere", segments=4)
array([[[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [ -3.5355339...e-01,  -4.3297802...e-17,   3.5355339...e-01],
        [ -5.0000000...e-01,  -6.1232340...e-17,   3.0616170...e-17],
        [ -3.5355339...e-01,  -4.3297802...e-17,  -3.5355339...e-01],
        [ -6.1232340...e-17,  -7.4987989...e-33,  -5.0000000...e-01]],

       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  2.1648901...e-17,  -3.5355339...e-01,   3.5355339...e-01],
        [  3.0616170...e-17,  -5.0000000...e-01,   3.0616170...e-17],
        [  2.1648901...e-17,  -3.5355339...e-01,  -3.5355339...e-01],
        [  3.7493994...e-33,  -6.1232340...e-17,  -5.0000000...e-01]],

       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  3.5355339...e-01,   0.0000000...e+00,   3.5355339...e-01],
        [  5.0000000...e-01,   0.0000000...e+00,   3.0616170...e-17],
        [  3.5355339...e-01,   0.0000000...e+00,  -3.5355339...e-01],
        [  6.1232340...e-17,   0.0000000...e+00,  -5.0000000...e-01]],

       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  2.1648901...e-17,   3.5355339...e-01,   3.5355339...e-01],
        [  3.0616170...e-17,   5.0000000...e-01,   3.0616170...e-17],
        [  2.1648901...e-17,   3.5355339...e-01,  -3.5355339...e-01],
        [  3.7493994...e-33,   6.1232340...e-17,  -5.0000000...e-01]]])
```

`colour.geometry`

`primitive_vertices_quad_mpl`([width, height, ...])	Return the vertices of a quad primitive for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.
`primitive_vertices_grid_mpl`([width, height, ...])	Return the vertices of a grid primitive made of quad primitives for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.
`primitive_vertices_cube_mpl`([width, height, ...])	Return the vertices of a cube primitive made of grid primitives for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.
`primitive_vertices_sphere`([radius, ...])	Return the vertices of a latitude-longitude sphere primitive.

colour.geometry.primitive_vertices_quad_mpl

`colour.geometry.`**`primitive_vertices_quad_mpl`**(*width: float = 1, height: float = 1, depth: float = 0, origin: ArrayLike = np.array([0, 0]), axis: Union[Literal['+z', '+x', '+y', 'yz', 'xz', 'xy'], str] = '+z'*) → NDArrayFloat

Return the vertices of a quad primitive for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.

Parameters

- **`width`** (`float`) – Quad width.
- **`height`** (`float`) – Quad height.

- **depth** (float) – Quad depth.
- **origin** (ArrayLike) – Quad origin on the construction plane.
- **axis** (Union[Literal['+z', '+x', '+y', 'yz', 'xz', 'xy'], str]) – Axis the quad will be normal to, or plane the quad will be co-planar with.

Returns Quad primitive vertices.

Return type numpy.ndarray

Examples

```
>>> primitive_vertices_quad_mpl()
array([[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  0.],
       [ 0.,  1.,  0.]])
```

colour.geometry.primitive_vertices_grid_mpl

colour.geometry.**primitive_vertices_grid_mpl**(*width: float = 1, height: float = 1, depth: float = 0, width_segments: int = 1, height_segments: int = 1, origin: ArrayLike = np.array([0, 0]), axis: Union[Literal['+z', '+x', '+y', 'yz', 'xz', 'xy'], str] = '+z'*) → NDArrayFloat

Return the vertices of a grid primitive made of quad primitives for use with *Matplotlib* mpl_toolkits.mplot3d.art3d.Poly3DCollection class.

Parameters

- **width** (float) – Grid width.
- **height** (float) – Grid height.
- **depth** (float) – Grid depth.
- **width_segments** (int) – Grid width segments, quad primitive counts along the width.
- **height_segments** (int) – Grid height segments, quad primitive counts along the height.
- **origin** (ArrayLike) – Grid origin on the construction plane.
- **axis** (Union[Literal['+z', '+x', '+y', 'yz', 'xz', 'xy'], str]) – Axis the grid will be normal to, or plane the grid will be co-planar with.

Returns Grid primitive vertices.

Return type numpy.ndarray

Examples

```
>>> primitive_vertices_grid_mpl(width_segments=2, height_segments=2)
array([[[ 0. ,  0. ,  0. ],
        [ 0.5,  0. ,  0. ],
        [ 0.5,  0.5,  0. ],
        [ 0. ,  0.5,  0. ]],

       [[ 0. ,  0.5,  0. ],
        [ 0.5,  0.5,  0. ],
        [ 0.5,  1. ,  0. ],
        [ 0. ,  1. ,  0. ]],

       [[ 0.5,  0. ,  0. ],
        [ 1. ,  0. ,  0. ],
        [ 1. ,  0.5,  0. ],
        [ 0.5,  0.5,  0. ]],

       [[ 0.5,  0.5,  0. ],
        [ 1. ,  0.5,  0. ],
        [ 1. ,  1. ,  0. ],
        [ 0.5,  1. ,  0. ]]])
```

colour.geometry.primitive_vertices_cube_mpl

`colour.geometry.`**`primitive_vertices_cube_mpl`**(*width: float = 1, height: float = 1, depth: float = 1, width_segments: int = 1, height_segments: int = 1, depth_segments: int = 1, origin: ArrayLike = np.array([0, 0, 0]), planes: Optional[Literal['-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy']] = None*) → NDArrayFloat

Return the vertices of a cube primitive made of grid primitives for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.

Parameters

- **`width`** (`float`) – Cube width.
- **`height`** (`float`) – Cube height.
- **`depth`** (`float`) – Cube depth.
- **`width_segments`** (`int`) – Cube segments count along the width.
- **`height_segments`** (`int`) – Cube segments count along the height.
- **`depth_segments`** (`int`) – Cube segments count along the depth.
- **`origin`** (`ArrayLike`) – Cube origin.
- **`planes`** (`Optional[Literal['-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy']]`) – Grid primitives to include in the cube construction.

Returns Cube primitive vertices.

Return type `numpy.ndarray`

Examples

```
>>> primitive_vertices_cube_mpl()
array([[[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
        [ 0.,  1.,  0.]],

       [[ 0.,  0.,  1.],
        [ 1.,  0.,  1.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  0.,  1.],
        [ 0.,  0.,  1.]],

       [[ 0.,  1.,  0.],
        [ 1.,  1.,  0.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 0.,  1.,  0.],
        [ 0.,  1.,  1.],
        [ 0.,  0.,  1.]],

       [[ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
        [ 1.,  1.,  1.],
        [ 1.,  0.,  1.]]])
```

colour.geometry.primitive_vertices_sphere

`colour.geometry.`**`primitive_vertices_sphere`**(*radius: float = 0.5, segments: int = 8, intermediate: bool = False, origin: ArrayLike = np.array([0, 0, 0]), axis: Union[Literal['+z', '+x', '+y', 'yz', 'xz', 'xy'], str] = '+z'*) → NDArrayFloat

Return the vertices of a latitude-longitude sphere primitive.

Parameters

- **`radius`** (`float`) – Sphere radius.
- **`segments`** (`int`) – Latitude-longitude segments, if the `intermediate` argument is *True*, then the sphere will have one less segment along its longitude.
- **`intermediate`** (`bool`) – Whether to generate the sphere vertices at the center of the faces outlined by the segments of a regular sphere generated without the `intermediate` argument set to *True*. The resulting sphere is inscribed on the regular sphere faces but possesses the same poles.
- **`origin`** (`ArrayLike`) – Sphere origin on the construction plane.
- **`axis`** (`Union[Literal['+z', '+x', '+y', 'yz', 'xz', 'xy'], str]`) – Axis (or normal of the plane) the poles of the sphere will be aligned with.

Returns Sphere primitive vertices.

Return type numpy.ndarray

Notes

- The sphere poles have latitude segments count - 1 co-located vertices.

Examples

```
>>> primitive_vertices_sphere(segments=4)
array([[[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [ -3.5355339...e-01,  -4.3297802...e-17,   3.5355339...e-01],
        [ -5.0000000...e-01,  -6.1232340...e-17,   3.0616170...e-17],
        [ -3.5355339...e-01,  -4.3297802...e-17,  -3.5355339...e-01],
        [ -6.1232340...e-17,  -7.4987989...e-33,  -5.0000000...e-01]],

       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  2.1648901...e-17,  -3.5355339...e-01,   3.5355339...e-01],
        [  3.0616170...e-17,  -5.0000000...e-01,   3.0616170...e-17],
        [  2.1648901...e-17,  -3.5355339...e-01,  -3.5355339...e-01],
        [  3.7493994...e-33,  -6.1232340...e-17,  -5.0000000...e-01]],

       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  3.5355339...e-01,   0.0000000...e+00,   3.5355339...e-01],
        [  5.0000000...e-01,   0.0000000...e+00,   3.0616170...e-17],
        [  3.5355339...e-01,   0.0000000...e+00,  -3.5355339...e-01],
        [  6.1232340...e-17,   0.0000000...e+00,  -5.0000000...e-01]],

       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  2.1648901...e-17,   3.5355339...e-01,   3.5355339...e-01],
        [  3.0616170...e-17,   5.0000000...e-01,   3.0616170...e-17],
        [  2.1648901...e-17,   3.5355339...e-01,  -3.5355339...e-01],
        [  3.7493994...e-33,   6.1232340...e-17,  -5.0000000...e-01]]])
```

Hull Section

colour.geometry

hull_section(hull[, axis, origin, normalise])	Compute the hull section for given axis at given origin.

colour.geometry.hull_section

colour.geometry.**hull_section**(*hull: trimesh.Trimesh*, *axis: Literal['+z', '+x', '+y'] | str = '+z'*, *origin: float = 0.5*, *normalise: bool = False*) → NDArrayFloat

Compute the hull section for given axis at given origin.

Parameters

- **hull** (trimesh.Trimesh) – *Trimesh* hull.
- **axis** (Literal['+z', '+x', '+y'] | str) – Axis the hull section will be normal to.
- **origin** (float) – Coordinate along axis at which to plot the hull section.
- **normalise** (bool) – Whether to normalise axis to the extent of the hull along it.

Returns Hull section vertices.

Return type numpy.ndarray

Examples

```
>>> from colour.geometry import primitive_cube
>>> from colour.utilities import is_trimesh_installed
>>> vertices, faces, outline = primitive_cube(1, 1, 1, 2, 2, 2)
>>> if is_trimesh_installed:
...     import trimesh
...
...     hull = trimesh.Trimesh(vertices["position"], faces, process=False)
...     hull_section(hull, origin=0)
...
array([[-0. , -0.5,  0. ],
       [ 0.5, -0.5,  0. ],
       [ 0.5,  0. , -0. ],
       [ 0.5,  0.5, -0. ],
       [-0. ,  0.5,  0. ],
       [-0.5,  0.5,  0. ],
       [-0.5,  0. , -0. ],
       [-0.5, -0.5, -0. ],
       [-0. , -0.5,  0. ]])
```

Automatic Colour Conversion Graph

Conversion

colour

convert(a, source, target, **kwargs)	Convert given object a from source colour representation to target colour representation using the automatic colour conversion graph.
describe_conversion_path(source, target[, ...])	Describe the conversion path from source colour representation to target colour representation using the automatic colour conversion graph.

colour.convert

colour.**convert**(*a: Any*, *source: str*, *target: str*, ***kwargs: Any*) → *Any*

Convert given object a from source colour representation to target colour representation using the automatic colour conversion graph.

The conversion is performed by finding the shortest path in a NetworkX DiGraph class instance.

The conversion path adopts the **'1'** domain-range scale and the object a is expected to be *soft* normalised accordingly. For example, *CIE XYZ* tristimulus values arguments for use with the *CAM16* colour appearance model should be in domain *[0, 1]* instead of the domain *[0, 100]* used with the **'Reference'** domain-range scale. The arguments are typically converted as follows:

- *Scalars* in domain-range *[0, 10]*, e.g *Munsell Value* are scaled by *10*.
- *Percentages* in domain-range *[0, 100]* are scaled by *100*.
- *Degrees* in domain-range *[0, 360]* are scaled by *360*.

- *Integers* in domain-range *[0, 2**n -1]* where *n* is the bit depth are scaled by *2**n -1*.

See the Domain-Range Scales page for more information.

Parameters

- **a** (Any) – Object a to convert. If a represents a reflectance, transmittance or absorptance value, the expectation is that it is viewed under *CIE Standard Illuminant D Series D65*. The illuminant can be changed on a per-definition basis along the conversion path.
- **source** (str) – Source colour representation, i.e. the source node in the automatic colour conversion graph.
- **target** (str) – Target colour representation, i.e. the target node in the automatic colour conversion graph.
- **kwargs** (Any) – See the documentation of the supported conversion definitions.

  Arguments for the conversion definitions are passed as keyword arguments whose names is those of the conversion definitions and values set as dictionaries. For example, in the conversion from spectral distribution to *sRGB* colourspace, passing arguments to the `colour.sd_to_XYZ()` definition is done as follows:

```
convert(sd, "Spectral Distribution", "sRGB", sd_to_XYZ={"illuminant
↪": SDS_ILLUMINANTS["FL2"]})
```

  It is also possible to pass keyword arguments directly to the various conversion definitions irrespective of their name. This is `dangerous` and could cause unexpected behaviour, consider the following conversion:

```
convert(sd, "Spectral Distribution", "sRGB", "illuminant": SDS_
↪ILLUMINANTS["FL2"])
```

  Because both the `colour.sd_to_XYZ()` and `colour.XYZ_to_sRGB()` definitions have an *illuminant* argument, *SDS_ILLUMINANTS["FL2"]* will be passed to both of them and will raise an exception in the `colour.XYZ_to_sRGB()` definition. This will be addressed in the future by either catching the exception and trying a new time without the keyword argument or more elegantly via type checking.

  With that in mind, this mechanism offers some good benefits: For example, it allows defining a conversion from *CIE XYZ* colourspace to *n* different colour models while passing an illuminant argument but without having to explicitly define all the explicit conversion definition arguments:

```
a = np.array([0.20654008, 0.12197225, 0.05136952])
illuminant = CCS_ILLUMINANTS["CIE 1931 2 Degree Standard Observer"][
↪"D65"]
for model in ("CIE xyY", "CIE Lab"):
    convert(a, "CIE XYZ", model, illuminant=illuminant)
```

  Instead of:

```
for model in ("CIE xyY", "CIE Lab"):
    convert(a, "CIE XYZ", model, XYZ_to_xyY={"illuminant":␣
↪illuminant}, XYZ_to_Lab={"illuminant": illuminant})
```

  Mixing both approaches is possible for the brevity benefits. It is made possible because the keyword arguments directly passed are filtered first and then the resulting dict is updated with the explicit conversion definition arguments:

```
illuminant = CCS_ILLUMINANTS["CIE 1931 2 Degree Standard Observer"][
↪"D65"]
 convert(sd, "Spectral Distribution", "sRGB", "illuminant": SDS_
↪ILLUMINANTS["FL2"], XYZ_to_sRGB={"illuminant": illuminant})
```

For inspection purposes, verbose is enabled by passing arguments to the `colour.describe_conversion_path()` definition via the `verbose` keyword argument as follows:

```
convert(sd, "Spectral Distribution", "sRGB", verbose={"mode": "Long"}
↪)
```

Returns Converted object a.

Return type *Any*

Warning: The domain-range scale is **'1'** and cannot be changed.

Notes

- The **RGB** colour representation is assumed to be linear and representing *scene-referred* imagery, i.e. **Scene-Referred RGB** representation. To encode such *RGB* values as *output-referred* (*display-referred*) imagery, i.e. encode the *RGB* values using an encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF), the **Output-Referred RGB** representation must be used:

```
convert(RGB, "Scene-Referred RGB", "Output-Referred RGB")
```

  Likewise, encoded *output-referred RGB* values can be decoded with the **Scene-Referred RGB** representation:

```
convert(RGB, "Output-Referred RGB", "Scene-Referred RGB")
```

- The following defaults have been adopted:
  - The default illuminant for the computation is *CIE Standard Illuminant D Series D65*. It can be changed on a per-definition basis along the conversion path. Note that the conversion from spectral to *CIE XYZ* tristimulus values remains unchanged.
  - The default *RGB* colourspace primaries and whitepoint are that of the *BT.709/sRGB* colourspace. They can be changed on a per-definition basis along the conversion path.
  - When using **sRGB** as a source or target colour representation, the convenient `colour.sRGB_to_XYZ()` and `colour.XYZ_to_sRGB()` definitions are used, respectively. Thus, decoding and encoding using the sRGB electro-optical transfer function (EOTF) and its inverse will be applied by default.
  - Most of the colour appearance models have defaults set according to *IEC 61966-2-1:1999* viewing conditions, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.

Examples

```
>>> import numpy as np
>>> from colour import SDS_COLOURCHECKERS, SDS_ILLUMINANTS
>>> sd = SDS_COLOURCHECKERS["ColorChecker N Ohta"]["dark skin"]
>>> convert(
...     sd,
...     "Spectral Distribution",
...     "sRGB",
...     verbose={"mode": "Short", "width": 75},
... )
...
===========================================================================
*                                                                         *
*   [ Conversion Path ]                                                   *
*                                                                         *
*   "sd_to_XYZ" --> "XYZ_to_sRGB"                                         *
*                                                                         *
===========================================================================
array([ 0.4903477...,  0.3018587...,  0.2358768...])
>>> illuminant = SDS_ILLUMINANTS["FL2"]
>>> convert(
...     sd,
...     "Spectral Distribution",
...     "sRGB",
...     sd_to_XYZ={"illuminant": illuminant},
... )
...
array([ 0.4792457...,  0.3167696...,  0.1736272...])
>>> a = np.array([0.45675795, 0.30986982, 0.24861924])
>>> convert(a, "Output-Referred RGB", "CAM16UCS")
...
array([ 0.3999481...,  0.0920655...,  0.0812752...])
>>> a = np.array([0.39994811, 0.09206558, 0.08127526])
>>> convert(a, "CAM16UCS", "sRGB", verbose={"mode": "Short", "width": 75})
...
===========================================================================
*                                                                         *
*   [ Conversion Path ]                                                   *
*                                                                         *
*   "UCS_Li2017_to_JMh_CAM16" --> "JMh_CAM16_to_CAM16" -->                *
*   "CAM16_to_XYZ" --> "XYZ_to_sRGB"                                      *
*                                                                         *
===========================================================================
array([ 0.4567576...,  0.3098826...,  0.2486222...])
```

colour.describe_conversion_path

colour.**describe_conversion_path**(*source: str, target: str, mode: Union[Literal['Short', 'Long', 'Extended'], str] = 'Short', width: int = 79, padding: int = 3, print_callable: Callable = print, **kwargs: Any*)

Describe the conversion path from source colour representation to target colour representation using the automatic colour conversion graph.

Parameters

- **source** (str) – Source colour representation, i.e. the source node in the automatic colour conversion graph.

- **target** (str) – Target colour representation, i.e. the target node in the automatic colour conversion graph.
- **mode** (Union[Literal['Short', 'Long', 'Extended'], str]) – Verbose mode: *Short* describes the conversion path, *Long* provides details about the arguments, definitions signatures and output values, *Extended* appends the definitions' documentation.
- **width** (int) – Message box width.
- **padding** (int) – Padding on each side of the message.
- **print_callable** (Callable) – Callable used to print the message box.
- **kwargs** (Any) – {colour.convert()}, See the documentation of the previously listed definition.

Examples

```
>>> describe_conversion_path("Spectral Distribution", "sRGB", width=75)
===========================================================================
*                                                                         *
*   [ Conversion Path ]                                                   *
*                                                                         *
*   "sd_to_XYZ" --> "XYZ_to_sRGB"                                         *
*                                                                         *
===========================================================================
```

Annotation Type Hints

colour.hints

ArrayLike	alias of Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]
NDArray	ndarray(shape, dtype=float, buffer=None, offset=0,
ModuleType	alias of module
Any(*args, **kwargs)	Special type indicating an unconstrained type.
Callable	Deprecated alias to collections.abc.Callable.
Dict	A generic version of dict.
Generator()	
Iterable()	
Iterator()	
List	A generic version of list.
Literal	Special typing form to define literal types (a.k.a.

continues on next page

Table 1 – continued from previous page

Mapping()	A Mapping is a generic container for associating key/value pairs.
NewType(name, tp)	NewType creates simple unique types with almost zero runtime overhead.
Optional	Optional[X] is equivalent to Union[X, None].
Protocol()	Base class for protocol classes.
Sequence()	All the operations on a read-only sequence.
SupportsIndex(*args, **kwargs)	An ABC with one abstract method __index__.
TYPE_CHECKING	bool(x) -> bool
TextIO()	Typed version of the return of open() in text mode.
Tuple	Deprecated alias to builtins.tuple.
Type	Deprecated alias to builtins.type.
TypeVar(name, *constraints[, bound, ...])	Type variable.
TypedDict(typename[, fields, total])	A simple typed namespace.
Union	Union type; Union[X, Y] means either X or Y.
cast(typ, val)	Cast a value to a type.
overload(func)	Decorator for overloaded functions/methods.
runtime_checkable(cls)	Mark a protocol class as a runtime protocol.
Self	Used to spell the type of "self" in classes.
RegexFlag	NewType creates simple unique types with almost zero runtime overhead.
DTypeInt	alias of Union[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64]
DTypeFloat	alias of Union[numpy.float16, numpy.float32, numpy.float64]
DTypeReal	alias of Union[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64, numpy.float16, numpy.float32, numpy.float64]
DTypeComplex	alias of Union[numpy.complex64, numpy.complex128]
DTypeBoolean	alias of numpy.bool_
DType	alias of Union[numpy.bool_, numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64, numpy.float16, numpy.float32, numpy.float64, numpy.complex64, numpy.complex128]
Real	alias of Union[int, float]
Dataclass	alias of Any
NDArrayInt	ndarray(shape, dtype=float, buffer=None, offset=0,
NDArrayFloat	ndarray(shape, dtype=float, buffer=None, offset=0,
NDArrayReal	ndarray(shape, dtype=float, buffer=None, offset=0,
NDArrayComplex	ndarray(shape, dtype=float, buffer=None, offset=0,
NDArrayBoolean	ndarray(shape, dtype=float, buffer=None, offset=0,
NDArrayStr	ndarray(shape, dtype=float, buffer=None, offset=0,
ProtocolInterpolator(*args, **kwargs)	

continues on next page

Table 1 – continued from previous page

ProtocolExtrapolator(*args, **kwargs)	
ProtocolLUTSequenceItem(*args, **kwargs)	
LiteralWarning	alias of Literal['default', 'error', 'ignore', 'always', 'module', 'once']

colour.hints.ArrayLike

colour.hints.**ArrayLike**

alias of Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]

colour.hints.NDArray

colour.hints.**NDArray**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(. . .)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type* <*numpy.dtype.type*>.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

alias of `numpy.ndarray`[`Any`, `numpy.dtype`[`numpy._typing._array_like._ScalarType_co`]]

colour.hints.ModuleType

`colour.hints.`**`ModuleType`**

alias of `module`

colour.hints.Any

class `colour.hints.`**`Any`**(**args*, ***kwargs*)

Special type indicating an unconstrained type.

- Any is compatible with every type.
- Any assumed to have all methods.
- All values assumed to be instances of Any.

Note that all the above statements are true from the point of view of static type checkers. At runtime, Any should not be used with instance checks.

`__init__`()

Methods

`__init__`()

colour.hints.Callable

`colour.hints.`**`Callable`**

Deprecated alias to collections.abc.Callable.

Callable[[int], str] signifies a function of (int) -> str. The subscription syntax must always be used with exactly two values: the argument list and the return type. The argument list must be a list of types, a ParamSpec or ellipsis. The return type must be a single type.

There is no syntax to indicate optional or keyword arguments; such function types are rarely used as callback types.

alias of `Callable`

colour.hints.Dict

`colour.hints.Dict`

A generic version of dict.

alias of `Dict`

colour.hints.Generator

`class colour.hints.Generator`

`__init__()`

Methods

`__init__()`	
`close()`	Raise GeneratorExit inside generator.
`send(value)`	Send a value into the generator.
`throw(typ[, val, tb])`	Raise an exception in the generator.

colour.hints.Iterable

`class colour.hints.Iterable`

`__init__()`

Methods

`__init__()`	

colour.hints.Iterator

`class colour.hints.Iterator`

`__init__()`

Methods

`__init__()`	

colour.hints.List

colour.hints.**List**

A generic version of list.

alias of List

colour.hints.Literal

colour.hints.**Literal = typing.Literal**

Special typing form to define literal types (a.k.a. value types).

This form can be used to indicate to type checkers that the corresponding variable or function parameter has a value equivalent to the provided literal (or one of several literals):

```
def validate_simple(data: Any) -> Literal[True]:  # always returns True
    ...

MODE = Literal['r', 'rb', 'w', 'wb']
def open_helper(file: str, mode: MODE) -> str:
    ...

open_helper('/some/path', 'r')  # Passes type check
open_helper('/other/path', 'typo')  # Error in type checker
```

Literal[...] cannot be subclassed. At runtime, an arbitrary value is allowed as type argument to Literal[...], but type checkers may impose restrictions.

colour.hints.Mapping

class colour.hints.**Mapping**

A Mapping is a generic container for associating key/value pairs.

This class provides concrete generic implementations of all methods except for __getitem__, __iter__, and __len__.

__init__()

Methods

__init__()
get(k[,d])
items()
keys()
values()

colour.hints.NewType

class colour.hints.**NewType**(*name*, *tp*)

NewType creates simple unique types with almost zero runtime overhead.

NewType(name, tp) is considered a subtype of tp by static type checkers. At runtime, NewType(name, tp) returns a dummy callable that simply returns its argument. Usage:

```
UserId = NewType('UserId', int)

def name_by_id(user_id: UserId) -> str:
    ...

UserId('user')          # Fails type check

name_by_id(42)          # Fails type check
name_by_id(UserId(42))  # OK

num = UserId(5) + 1     # type: int
```

__init__(*name*, *tp*)

Methods

__init__(name, tp)	

colour.hints.Optional

colour.hints.**Optional = typing.Optional**

Optional[X] is equivalent to Union[X, None].

colour.hints.Protocol

class colour.hints.**Protocol**

Base class for protocol classes.

Protocol classes are defined as:

```
class Proto(Protocol):
    def meth(self) -> int:
        ...
```

Such classes are primarily used with static type checkers that recognize structural subtyping (static duck-typing), for example:

```
class C:
    def meth(self) -> int:
        return 0

def func(x: Proto) -> int:
    return x.meth()

func(C())  # Passes static type check
```

See PEP 544 for details. Protocol classes decorated with @typing.runtime_checkable act as simple-minded runtime protocols that check only the presence of given attributes, ignoring their type signatures. Protocol classes can be generic, they are defined as:

```
class GenProto(Protocol[T]):
    def meth(self) -> T:
        ...
```

__init__()

Methods

__init__()	

colour.hints.Sequence

class colour.hints.**Sequence**

All the operations on a read-only sequence.

Concrete subclasses must override __new__ or __init__, __getitem__, and __len__.

__init__()

Methods

__init__()	
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

colour.hints.SupportsIndex

class colour.hints.**SupportsIndex**(**args*, ***kwargs*)

An ABC with one abstract method __index__.

__init__(**args*, ***kwargs*)

Methods

__init__(*args, **kwargs)	

colour.hints.TYPE_CHECKING

`colour.hints.TYPE_CHECKING = False`

bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

colour.hints.TextIO

`class colour.hints.TextIO`

Typed version of the return of open() in text mode.

`__init__()`

Methods

`__init__`()
`close`()
`fileno`()
`flush`()
`isatty`()
`read`([n])
`readable`()
`readline`([limit])
`readlines`([hint])
`seek`(offset[, whence])
`seekable`()
`tell`()
`truncate`([size])
`writable`()
`write`(s)
`writelines`(lines)

Attributes

buffer
closed
encoding
errors
line_buffering
mode
name
newlines

colour.hints.Tuple

colour.hints.**Tuple**

Deprecated alias to builtins.tuple.

Tuple[X, Y] is the cross-product type of X and Y.

Example: Tuple[T1, T2] is a tuple of two elements corresponding to type variables T1 and T2. Tuple[int, float, str] is a tuple of an int, a float and a string.

To specify a variable-length tuple of homogeneous type, use Tuple[T, ...].

alias of Tuple

colour.hints.Type

colour.hints.**Type**

Deprecated alias to builtins.type.

builtins.type or typing.Type can be used to annotate class objects. For example, suppose we have the following classes:

```
class User: ...  # Abstract base for User classes
class BasicUser(User): ...
class ProUser(User): ...
class TeamUser(User): ...
```

And a function that takes a class argument that's a subclass of User and returns an instance of the corresponding class:

```
U = TypeVar('U', bound=User)
def new_user(user_class: Type[U]) -> U:
    user = user_class()
    # (Here we could write the user object to a database)
    return user

joe = new_user(BasicUser)
```

At this point the type checker knows that joe has type BasicUser.

alias of Type

colour.hints.TypeVar

class colour.hints.**TypeVar**(*name*, **constraints*, *bound=None*, *covariant=False*, *contravariant=False*)

Type variable.

Usage:

```
T = TypeVar('T')  # Can be anything
A = TypeVar('A', str, bytes)  # Must be str or bytes
```

Type variables exist primarily for the benefit of static type checkers. They serve as the parameters for generic types as well as for generic function definitions. See class Generic for more information on generic types. Generic functions work as follows:

def repeat(x: T, n: int) -> List[T]: '"Return a list containing n references to x."' return [x]*n

def longest(x: A, y: A) -> A: '"Return the longest of two strings."' return x if len(x) >= len(y) else y

The latter example's signature is essentially the overloading of (str, str) -> str and (bytes, bytes) -> bytes. Also note that if the arguments are instances of some subclass of str, the return type is still plain str.

At runtime, isinstance(x, T) and issubclass(C, T) will raise TypeError.

Type variables defined with covariant=True or contravariant=True can be used to declare covariant or contravariant generic types. See PEP 484 for more details. By default generic types are invariant in all type variables.

Type variables can be introspected. e.g.:

T.__name__ == 'T' T.__constraints__ == () T.__covariant__ == False T.__contravariant__ = False A.__constraints__ == (str, bytes)

Note that only type variables defined in global scope can be pickled.

__init__(*name*, **constraints*, *bound=None*, *covariant=False*, *contravariant=False*)

Methods

__init__(name, *constraints[, bound, ...])	

colour.hints.TypedDict

colour.hints.**TypedDict**(*typename*, *fields=None*, */*, ***, *total=True*, ***kwargs*)

A simple typed namespace. At runtime it is equivalent to a plain dict.

TypedDict creates a dictionary type such that a type checker will expect all instances to have a certain set of keys, where each key is associated with a value of a consistent type. This expectation is not checked at runtime.

Usage:

```
class Point2D(TypedDict):
    x: int
    y: int
    label: str

a: Point2D = {'x': 1, 'y': 2, 'label': 'good'}  # OK
b: Point2D = {'z': 3, 'label': 'bad'}           # Fails type check

assert Point2D(x=1, y=2, label='first') == dict(x=1, y=2, label='first')
```

The type info can be accessed via the Point2D.__annotations__ dict, and the Point2D.__required_keys__ and Point2D.__optional_keys__ frozensets. TypedDict supports an additional equivalent form:

```
Point2D = TypedDict('Point2D', {'x': int, 'y': int, 'label': str})
```

By default, all keys must be present in a TypedDict. It is possible to override this by specifying totality:

```
class Point2D(TypedDict, total=False):
    x: int
    y: int
```

This means that a Point2D TypedDict can have any of the keys omitted. A type checker is only expected to support a literal False or True as the value of the total argument. True is the default, and makes all items defined in the class body be required.

The Required and NotRequired special forms can also be used to mark individual keys as being required or not required:

```
class Point2D(TypedDict):
    x: int               # the "x" key must always be present (Required is the␣
↪default)
    y: NotRequired[int]  # the "y" key can be omitted
```

See PEP 655 for more details on Required and NotRequired.

colour.hints.Union

colour.hints.**Union = typing.Union**

Union type; Union[X, Y] means either X or Y.

On Python 3.10 and higher, the | operator can also be used to denote unions; X | Y means the same thing to the type checker as Union[X, Y].

To define a union, use e.g. Union[int, str]. Details: - The arguments must be types and there must be at least one. - None as an argument is a special case and is replaced by

type(None).

- Unions of unions are flattened, e.g.:

```
assert Union[Union[int, str], float] == Union[int, str, float]
```

- Unions of a single argument vanish, e.g.:

```
assert Union[int] == int  # The constructor actually returns int
```

- Redundant arguments are skipped, e.g.:

```
assert Union[int, str, int] == Union[int, str]
```

- When comparing unions, the argument order is ignored, e.g.:

```
assert Union[int, str] == Union[str, int]
```

- You cannot subclass or instantiate a union.
- You can use Optional[X] as a shorthand for Union[X, None].

colour.hints.cast

`colour.hints.`**`cast`**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

colour.hints.overload

`colour.hints.`**`overload`**(*func*)

Decorator for overloaded functions/methods.

In a stub file, place two or more stub definitions for the same function in a row, each decorated with @overload. For example:

```
@overload
def utf8(value: None) -> None: ...
@overload
def utf8(value: bytes) -> bytes: ...
@overload
def utf8(value: str) -> bytes: ...
```

In a non-stub file (i.e. a regular .py file), do the same but follow it with an implementation. The implementation should *not* be decorated with @overload. For example:

```
@overload
def utf8(value: None) -> None: ...
@overload
def utf8(value: bytes) -> bytes: ...
@overload
def utf8(value: str) -> bytes: ...
def utf8(value):
    ...  # implementation goes here
```

The overloads for a function can be retrieved at runtime using the get_overloads() function.

colour.hints.runtime_checkable

colour.hints.**runtime_checkable**(*cls*)

Mark a protocol class as a runtime protocol.

Such protocol can be used with isinstance() and issubclass(). Raise TypeError if applied to a non-protocol class. This allows a simple-minded structural check very similar to one trick ponies in collections.abc such as Iterable.

For example:

```
@runtime_checkable
class Closable(Protocol):
    def close(self): ...

assert isinstance(open('/some/file'), Closable)
```

Warning: this will check only the presence of the required methods, not their type signatures!

colour.hints.Self

colour.hints.**Self = typing.Self**

Used to spell the type of "self" in classes.

Example:

```
from typing import Self

class Foo:
    def return_self(self) -> Self:
        ...
        return self
```

This is especially useful for:

- classmethods that are used as alternative constructors
- annotating an __*enter*__ method which returns self

colour.hints.RegexFlag

colour.hints.**RegexFlag**

NewType creates simple unique types with almost zero runtime overhead.

NewType(name, tp) is considered a subtype of tp by static type checkers. At runtime, NewType(name, tp) returns a dummy callable that simply returns its argument. Usage:

```
UserId = NewType('UserId', int)

def name_by_id(user_id: UserId) -> str:
    ...

UserId('user')          # Fails type check

name_by_id(42)          # Fails type check
name_by_id(UserId(42))  # OK

num = UserId(5) + 1     # type: int
```

alias of re.RegexFlag

colour.hints.DTypeInt

colour.hints.**DTypeInt**

alias of Union[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64]

colour.hints.DTypeFloat

colour.hints.**DTypeFloat**

alias of Union[numpy.float16, numpy.float32, numpy.float64]

colour.hints.DTypeReal

colour.hints.**DTypeReal**

alias of Union[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64, numpy.float16, numpy.float32, numpy.float64]

colour.hints.DTypeComplex

colour.hints.**DTypeComplex**

alias of Union[numpy.complex64, numpy.complex128]

colour.hints.DTypeBoolean

colour.hints.**DTypeBoolean**

alias of numpy.bool_

colour.hints.DType

colour.hints.**DType**

alias of Union[numpy.bool_, numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64, numpy.float16, numpy.float32, numpy.float64, numpy.complex64, numpy.complex128]

colour.hints.Real

colour.hints.**Real**

alias of Union[int, float]

colour.hints.Dataclass

`colour.hints.Dataclass`

alias of `Any`

colour.hints.NDArrayInt

`colour.hints.NDArrayInt`

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(. . .)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** (`(for the __new__ method; see Notes)`) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.
- **offset** (`int, optional`) – Offset of array data in buffer.
- **strides** (`tuple of ints, optional`) – Strides of data in memory.
- **order** (`{'C', 'F'}, optional`) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

alias of `numpy.ndarray[Any, numpy.dtype[Union[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64]]]`

colour.hints.NDArrayFloat

colour.hints.**NDArrayFloat**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** (`(for the __new__ method; see Notes)`) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.
- **offset** (`int, optional`) – Offset of array data in buffer.
- **strides** (`tuple of ints, optional`) – Strides of data in memory.
- **order** (`{'C', 'F'}, optional`) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time `(2 * 4)`.

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type* <*numpy.dtype.type*>.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

alias of `numpy.ndarray[Any, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]`

colour.hints.NDArrayReal

`colour.hints.NDArrayReal`

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** (`(for the __new__ method; see Notes)`) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.
- **offset** (`int, optional`) – Offset of array data in buffer.
- **strides** (`tuple of ints, optional`) – Strides of data in memory.
- **order** (`{'C', 'F'}, optional`) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

alias of `numpy.ndarray[Any, numpy.dtype[Union[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64, numpy.float16, numpy.float32, numpy.float64]]]`

colour.hints.NDArrayComplex

`colour.hints.`**`NDArrayComplex`**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(…)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** (`(for the __new__ method; see Notes)`) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.
- **offset** (`int, optional`) – Offset of array data in buffer.
- **strides** (`tuple of ints, optional`) – Strides of data in memory.

- **order** (`{'C', 'F'}, optional`) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

alias of `numpy.ndarray[Any, numpy.dtype[Union[numpy.complex64, numpy.complex128]]]`

colour.hints.NDArrayBoolean

`colour.hints.`**`NDArrayBoolean`**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(. . .)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** (`(for the __new__ method; see Notes)`) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.
- **offset** (`int, optional`) – Offset of array data in buffer.
- **strides** (`tuple of ints, optional`) – Strides of data in memory.
- **order** (`{'C', 'F'}, optional`) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type* <*numpy.dtype.type*>.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

alias of `numpy.ndarray[Any, numpy.dtype[numpy.bool_]]`

colour.hints.NDArrayStr

`colour.hints.`**`NDArrayStr`**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** (`(for the __new__ method; see Notes)`) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.
- **offset** (`int, optional`) – Offset of array data in buffer.
- **strides** (`tuple of ints, optional`) – Strides of data in memory.
- **order** (`{'C', 'F'}, optional`) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

alias of `numpy.ndarray`[`Any`, `numpy.dtype`[`numpy.str_`]]

colour.hints.ProtocolInterpolator

class colour.hints.**ProtocolInterpolator**(**args: Any*, ***kwargs: Any*)

Parameters

- **args** (Any) –
- **kwargs** (Any) –

Return type None

__init__(**args: Any*, ***kwargs: Any*) → None

Parameters

- **args** (Any) –
- **kwargs** (Any) –

Return type None

Methods

`__init__`(*args, **kwargs)

Attributes

x
y

colour.hints.ProtocolExtrapolator

class colour.hints.**ProtocolExtrapolator**(**args: Any*, ***kwargs: Any*)

Parameters
- **args** (Any) –
- **kwargs** (Any) –

Return type None

__init__(**args: Any*, ***kwargs: Any*) → None

Parameters
- **args** (Any) –
- **kwargs** (Any) –

Return type None

Methods

`__init__`(*args, **kwargs)	

Attributes

`interpolator`	

colour.hints.ProtocolLUTSequenceItem

class colour.hints.**ProtocolLUTSequenceItem**(**args*, ***kwargs*)

__init__(**args*, ***kwargs*)

Methods

`__init__`(*args, **kwargs)	
`apply`(RGB, **kwargs)	

colour.hints.LiteralWarning

`colour.hints.`**`LiteralWarning`**

alias of `Literal`[‘default’, ‘error’, ‘ignore’, ‘always’, ‘module’, ‘once’]

Input and Output

Image Data

`colour`

`READ_IMAGE_METHODS`	Supported image read methods.
`read_image`(path[, bit_depth, method])	Read the image data at given path using given method.
`WRITE_IMAGE_METHODS`	Supported image write methods.
`write_image`(image, path[, bit_depth, method])	Write given image data at given path using given method.

colour.READ_IMAGE_METHODS

`colour.`**`READ_IMAGE_METHODS = CanonicalMapping({'Imageio': ..., 'OpenImageIO': ...})`**

Supported image read methods.

colour.read_image

`colour.`**`read_image`**(*path: str, bit_depth: Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128'] = 'float32', method: Union[Literal['Imageio', 'OpenImageIO'], str] = 'OpenImageIO', **kwargs: Any*) → NDArrayReal

Read the image data at given path using given method.

Parameters

- **`path`** (`str`) – Image path.
- **`bit_depth`** (`Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128']`) – Returned image bit-depth, for the *Imageio* method, the image data is converted with `colour.io.convert_bit_depth()` definition after reading the image, for the *OpenImageIO* method, the bit-depth conversion behaviour is driven directly by the library, this definition only converts to the relevant data type after reading.
- **`method`** (`Union[Literal['Imageio', 'OpenImageIO'], str]`) – Read method, i.e. the image library used for reading images.
- **`attributes`** – {`colour.io.read_image_OpenImageIO()`}, Whether to return the image attributes.
- **`kwargs`** (`Any`) –

Returns Image data.

Return type class`numpy.ndarray`

Notes

- If the given method is *OpenImageIO* but the library is not available writing will be performed by *Imageio*.
- If the given method is *Imageio*, `kwargs` is passed directly to the wrapped definition.
- For convenience, single channel images are squeezed to 2D arrays.

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     colour.__path__[0],
...     "io",
...     "tests",
...     "resources",
...     "CMS_Test_Pattern.exr",
... )
>>> image = read_image(path)
>>> image.shape
(1267, 1274, 3)
>>> image.dtype
dtype('float32')
```

colour.WRITE_IMAGE_METHODS

`colour.WRITE_IMAGE_METHODS = CanonicalMapping({'Imageio': ..., 'OpenImageIO': ...})`

Supported image write methods.

colour.write_image

`colour.write_image`(*image: ArrayLike, path: str, bit_depth: Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128'] = 'float32', method: Union[Literal['Imageio', 'OpenImageIO'], str] = 'OpenImageIO', **kwargs: Any*) → bool

Write given image data at given path using given method.

Parameters

- **image** (`ArrayLike`) – Image data.
- **path** (`str`) – Image path.
- **bit_depth** (`Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128']`) – Bit-depth to write the image at, for the *Imageio* method, the image data is converted with `colour.io.convert_bit_depth()` definition prior to writing the image.
- **method** (`Union[Literal['Imageio', 'OpenImageIO'], str]`) – Write method, i.e. the image library used for writing images.
- **attributes** – {`colour.io.write_image_OpenImageIO()`}, An array of `colour.io.ImageAttribute_Specification` class instances used to set attributes of the image.
- **kwargs** (`Any`) –

Returns Definition success.

Return type bool

Notes

- If the given method is *OpenImageIO* but the library is not available writing will be performed by *Imageio*.
- If the given method is *Imageio*, kwargs is passed directly to the wrapped definition.
- It is possible to control how the image are saved by the *Freeimage* backend by using the flags keyword argument and passing a desired value. See the *Load / Save flag constants* section in https://sourceforge.net/p/freeimage/svn/HEAD/tree/FreeImage/trunk/Source/FreeImage.h

Examples

Basic image writing:

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     colour.__path__[0],
...     "io",
...     "tests",
...     "resources",
...     "CMS_Test_Pattern.exr",
... )
>>> image = read_image(path)
>>> path = os.path.join(
...     colour.__path__[0],
...     "io",
...     "tests",
...     "resources",
...     "CMSTestPattern.tif",
... )
>>> write_image(image, path)
True
```

Advanced image writing while setting attributes using *OpenImageIO*:

```
>>> compression = ImageAttribute_Specification("Compression", "none")
>>> write_image(image, path, bit_depth="uint8", attributes=[compression])
...
True
```

Ancillary Objects

colour.io

ImageAttribute_Specification(name, value, type_)	Define an image specification attribute.
convert_bit_depth(a[, bit_depth])	Convert given array to given bit-depth, the current bit-depth of the array is used to determine the appropriate conversion path.
read_image_OpenImageIO(path[, bit_depth, ...])	Read the image data at given path using *OpenImageIO*.
write_image_OpenImageIO(image, path[, ...])	Write given image data at given path using *OpenImageIO*.
read_image_Imageio(path[, bit_depth])	Read the image data at given path using *Imageio*.
write_image_Imageio(image, path[, bit_depth])	Write given image data at given path using *Imageio*.
as_3_channels_image(a)	Convert given array a to a 3-channels image-like representation.

colour.io.ImageAttribute_Specification

class colour.io.**ImageAttribute_Specification**(*name: str*, *value: Any*, *type_: Optional[OpenImageIO.TypeDesc] = <factory>*)

Define an image specification attribute.

Parameters

- **name** (str) – Attribute name.
- **value** (Any) – Attribute value.
- **type** (Optional[OpenImageIO.TypeDesc]) – Attribute type as an *OpenImageIO* TypeDesc class instance.
- **type_** (Optional[OpenImageIO.TypeDesc]) –

Return type None

__init__(*name: str*, *value: Any*, *type_: Optional[OpenImageIO.TypeDesc] = <factory>*) → None

Parameters

- **name** (str) –
- **value** (Any) –
- **type_** (Optional[OpenImageIO.TypeDesc]) –

Return type None

Methods

__init__(name, value[, type_])	

Attributes

name
value
type_

colour.io.convert_bit_depth

colour.io.**convert_bit_depth**(*a: ArrayLike*, *bit_depth: Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128'] = 'float32'*) → NDArrayReal

Convert given array to given bit-depth, the current bit-depth of the array is used to determine the appropriate conversion path.

Parameters

- **a** (ArrayLike) – Array to convert to given bit-depth.
- **bit_depth** (Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128']) – Bit-depth.

Returns Converted array.

Return type class`numpy.ndarray`

Examples

```
>>> a = np.array([0.0, 0.5, 1.0])
>>> convert_bit_depth(a, "uint8")
array([  0, 128, 255], dtype=uint8)
>>> convert_bit_depth(a, "uint16")
array([    0, 32768, 65535], dtype=uint16)
>>> convert_bit_depth(a, "float16")
array([ 0. ,  0.5,  1. ], dtype=float16)
>>> a = np.array([0, 128, 255], dtype=np.uint8)
>>> convert_bit_depth(a, "uint16")
array([    0, 32896, 65535], dtype=uint16)
>>> convert_bit_depth(a, "float32")
array([ 0.        ,  0.501960...,  1.        ], dtype=float32)
```

colour.io.read_image_OpenImageIO

colour.io.**read_image_OpenImageIO**(*path: str*, *bit_depth: Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128'] = 'float32'*, *attributes: bool = False*) → NDArrayReal | Tuple[NDArrayReal, list]

Read the image data at given path using *OpenImageIO*.

Parameters

- **path** (str) – Image path.
- **bit_depth** (Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128']) – Returned image bit-depth, the bit-depth conversion behaviour is driven directly by *OpenImageIO*, this definition only converts to the relevant data type after reading.

- **attributes** (bool) – Whether to return the image attributes.

Returns Image data or tuple of image data and list of colour.io.ImageAttribute_Specification class instances.

Return type class`numpy.ndarray` or tuple

Notes

- For convenience, single channel images are squeezed to 2D arrays.

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     colour.__path__[0],
...     "io",
...     "tests",
...     "resources",
...     "CMS_Test_Pattern.exr",
... )
>>> image = read_image_OpenImageIO(path)
```

colour.io.write_image_OpenImageIO

colour.io.**write_image_OpenImageIO**(*image: ArrayLike, path: str, bit_depth: Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128'] = 'float32', attributes: collections.abc.Sequence | None = None*) → bool

Write given image data at given path using *OpenImageIO*.

Parameters

- **image** (ArrayLike) – Image data.
- **path** (str) – Image path.
- **bit_depth** (Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128']) – Bit-depth to write the image at, the bit-depth conversion behaviour is ruled directly by *OpenImageIO*.
- **attributes** (collections.abc.Sequence | None) – An array of colour.io.ImageAttribute_Specification class instances used to set attributes of the image.

Returns Definition success.

Return type bool

Examples

Basic image writing:

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     colour.__path__[0],
...     "io",
...     "tests",
...     "resources",
...     "CMS_Test_Pattern.exr",
... )
>>> image = read_image(path)
>>> path = os.path.join(
...     colour.__path__[0],
...     "io",
...     "tests",
...     "resources",
...     "CMSTestPattern.tif",
... )
>>> write_image_OpenImageIO(image, path)
True
```

Advanced image writing while setting attributes:

```
>>> compression = ImageAttribute_Specification("Compression", "none")
>>> write_image_OpenImageIO(image, path, "uint8", [compression])
...
True
```

Writing an "ACES" compliant "EXR" file:

```
>>> if is_openimageio_installed():
...     from OpenImageIO import TypeDesc
...
...     chromaticities = (
...         0.7347,
...         0.2653,
...         0.0,
...         1.0,
...         0.0001,
...         -0.077,
...         0.32168,
...         0.33767,
...     )
...     attributes = [
...         ImageAttribute_Specification("acesImageContainerFlag", True),
...         ImageAttribute_Specification(
...             "chromaticities", chromaticities, TypeDesc("float[8]")
...         ),
...         ImageAttribute_Specification("compression", "none"),
...     ]
...     write_image_OpenImageIO(image, path, attributes=attributes)
...
```

colour.io.read_image_Imageio

colour.io.**read_image_Imageio**(*path: str*, *bit_depth: Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128'] = 'float32'*, ***kwargs: Any*) → NDArrayReal

Read the image data at given path using *Imageio*.

Parameters

- **path** (str) – Image path.
- **bit_depth** (Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128']) – Returned image bit-depth, the image data is converted with colour.io.convert_bit_depth() definition after reading the image.
- **kwargs** (Any) – Keywords arguments.

Returns Image data.

Return type class`numpy.ndarray`

Notes

- For convenience, single channel images are squeezed to 2D arrays.

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     colour.__path__[0],
...     "io",
...     "tests",
...     "resources",
...     "CMS_Test_Pattern.exr",
... )
>>> image = read_image_Imageio(path)
>>> image.shape
(1267, 1274, 3)
>>> image.dtype
dtype('float32')
```

colour.io.write_image_Imageio

colour.io.**write_image_Imageio**(*image: ArrayLike*, *path: str*, *bit_depth: Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128'] = 'float32'*, ***kwargs: Any*) → bytes | None

Write given image data at given path using *Imageio*.

Parameters

- **image** (ArrayLike) – Image data.
- **path** (str) – Image path.
- **bit_depth** (Literal['uint8', 'uint16', 'float16', 'float32', 'float64', 'float128']) – Bit-depth to write the image at, the image data is converted with colour.io.convert_bit_depth() definition prior to writing the image.
- **kwargs** (Any) – Keywords arguments.

Returns Definition success.

Return type bool

Notes

- It is possible to control how the image are saved by the *Freeimage* backend by using the flags keyword argument and passing a desired value. See the *Load / Save flag constants* section in https://sourceforge.net/p/freeimage/svn/HEAD/tree/FreeImage/trunk/Source/FreeImage.h

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     colour.__path__[0],
...     "io",
...     "tests",
...     "resources",
...     "CMS_Test_Pattern.exr",
... )
>>> image = read_image(path)
>>> path = os.path.join(
...     colour.__path__[0],
...     "io",
...     "tests",
...     "resources",
...     "CMSTestPattern.tif",
... )
>>> write_image_Imageio(image, path)
True
```

colour.io.as_3_channels_image

colour.io.**as_3_channels_image**(*a: ArrayLike*) → NDArrayFloat

Convert given array a to a 3-channels image-like representation.

Parameters **a** (ArrayLike) – Array a to convert to a 3-channels image-like representation.

Returns 3-channels image-like representation of array a.

Return type class`numpy.ndarray`

Examples

```
>>> as_3_channels_image(0.18)
array([[[ 0.18,  0.18,  0.18]]])
>>> as_3_channels_image([0.18])
array([[[ 0.18,  0.18,  0.18]]])
>>> as_3_channels_image([0.18, 0.18, 0.18])
array([[[ 0.18,  0.18,  0.18]]])
>>> as_3_channels_image([[0.18, 0.18, 0.18]])
array([[[ 0.18,  0.18,  0.18]]])
```

(continues on next page)

(continued from previous page)

```
>>> as_3_channels_image([[[0.18, 0.18, 0.18]]])
array([[[ 0.18,  0.18,  0.18]]])
```

OpenColorIO Processing

`colour.io`

process_image_OpenColorIO(a, *args, **kwargs)	Process given image data with *OpenColorIO*.

colour.io.process_image_OpenColorIO

colour.io.**process_image_OpenColorIO**(*a: ArrayLike*, **args: Any*, ***kwargs: Any*) → NDArrayFloat

Process given image data with *OpenColorIO*.

Parameters

- **a** (ArrayLike) – Image data to process with *OpenColorIO*.
- **config** – *OpenColorIO* config to use for processing. If not defined, the *OpenColorIO* set defined by the $OCIO environment variable is used.
- **args** (Any) – Arguments for *Config.getProcessor* method. See https://opencolorio.readthedocs.io/en/latest/api/config.html for more information.
- **kwargs** (Any) –

Returns Processed image data.

Return type numpy.ndarray

Examples

```
>>> import os
>>> import PyOpenColorIO as ocio
>>> from colour.utilities import full
>>> config = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "config-aces-reference.ocio.yaml",
... )
>>> a = 0.18
>>> process_image_OpenColorIO(
...     a, "ACES - ACES2065-1", "ACES - ACEScct", config=config
... )
0.4135884...
>>> a = np.array([0.18])
>>> process_image_OpenColorIO(
...     a, "ACES - ACES2065-1", "ACES - ACEScct", config=config
... )
array([ 0.4135884...])
>>> a = np.array([0.18, 0.18, 0.18])
>>> process_image_OpenColorIO(
...     a, "ACES - ACES2065-1", "ACES - ACEScct", config=config
... )
```

(continues on next page)

(continued from previous page)

```
array([ 0.4135884...,  0.4135884...,  0.4135884...])
>>> a = np.array([[0.18, 0.18, 0.18]])
>>> process_image_OpenColorIO(
...     a, "ACES - ACES2065-1", "ACES - ACEScct", config=config
... )
array([[ 0.4135884...,  0.4135884...,  0.4135884...]])
>>> a = np.array([[[0.18, 0.18, 0.18]]])
>>> process_image_OpenColorIO(
...     a, "ACES - ACES2065-1", "ACES - ACEScct", config=config
... )
array([[[ 0.4135884...,  0.4135884...,  0.4135884...]]])
>>> a = full([4, 2, 3], 0.18)
>>> process_image_OpenColorIO(
...     a, "ACES - ACES2065-1", "ACES - ACEScct", config=config
... )
array([[[ 0.4135884...,  0.4135884...,  0.4135884...],
        [ 0.4135884...,  0.4135884...,  0.4135884...]],

       [[ 0.4135884...,  0.4135884...,  0.4135884...],
        [ 0.4135884...,  0.4135884...,  0.4135884...]],

       [[ 0.4135884...,  0.4135884...,  0.4135884...],
        [ 0.4135884...,  0.4135884...,  0.4135884...]],

       [[ 0.4135884...,  0.4135884...,  0.4135884...],
        [ 0.4135884...,  0.4135884...,  0.4135884...]]])
>>> process_image_OpenColorIO(
...     a,
...     "ACES - ACES2065-1",
...     "Display - sRGB",
...     "Output - SDR Video - ACES 1.0",
...     ocio.TRANSFORM_DIR_FORWARD,
...     config=config,
... )
array([[[ 0.3559542...,  0.3559542...,  0.3559542...],
        [ 0.3559542...,  0.3559542...,  0.3559542...]],

       [[ 0.3559542...,  0.3559542...,  0.3559542...],
        [ 0.3559542...,  0.3559542...,  0.3559542...]],

       [[ 0.3559542...,  0.3559542...,  0.3559542...],
        [ 0.3559542...,  0.3559542...,  0.3559542...]],

       [[ 0.3559542...,  0.3559542...,  0.3559542...],
        [ 0.3559542...,  0.3559542...,  0.3559542...]]])
```

Look Up Table (LUT) Data

colour

LUT1D([table, name, domain, size, comments])	Define the base class for a 1D *LUT*.
LUT3x1D([table, name, domain, size, comments])	Define the base class for a 3x1D *LUT*.
LUT3D([table, name, domain, size, comments])	Define the base class for a 3D *LUT*.
LUTOperatorMatrix([matrix, offset])	Define the *LUT* operator supporting a 3x3 or 4x4 matrix and an offset vector.
LUTSequence(*args)	Define the base class for a *LUT* sequence, i.e. a series of *LUTs*, *LUT* operators or objects implementing the colour.hints.ProtocolLUTSequenceItem protocol.

colour.LUT1D

class colour.**LUT1D**(*table: ArrayLike | None = None, name: str | None = None, domain: ArrayLike | None = None, size: ArrayLike | None = None, comments: Sequence | None = None*)

Bases: colour.io.luts.lut.AbstractLUT

Define the base class for a 1D *LUT*.

Parameters

- **table** (ArrayLike | None) – Underlying *LUT* table.
- **name** (str | None) – *LUT* name.
- **domain** (ArrayLike | None) – *LUT* domain, also used to define the instantiation time default table domain.
- **size** (ArrayLike | None) – Size of the instantiation time default table, default to 10.
- **comments** (Sequence | None) – Comments to add to the *LUT*.

Return type None

Methods

- __init__()
- is_domain_explicit()
- linear_table()
- invert()
- apply()
- convert()

Examples

Instantiating a unity LUT with a table with 16 elements:

```
>>> print(LUT1D(size=16))
LUT1D - Unity 16
----------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (16,)
```

Instantiating a LUT using a custom table with 16 elements:

```
>>> print(LUT1D(LUT1D.linear_table(16) ** (1 / 2.2)))
LUT1D - ...
--------...

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (16,)
```

Instantiating a LUT using a custom table with 16 elements, custom name, custom domain and comments:

```
>>> from colour.algebra import spow
>>> domain = np.array([-0.1, 1.5])
>>> print(
...     LUT1D(
...         spow(LUT1D.linear_table(16, domain), 1 / 2.2),
...         "My LUT",
...         domain,
...         comments=["A first comment.", "A second comment."],
...     )
... )
LUT1D - My LUT
--------------

Dimensions : 1
Domain     : [-0.1  1.5]
Size       : (16,)
Comment 01 : A first comment.
Comment 02 : A second comment.
```

__init__(*table: ArrayLike | None = None, name: str | None = None, domain: ArrayLike | None = None, size: ArrayLike | None = None, comments: Sequence | None = None*) → None

Parameters

- **table** (ArrayLike | None) –
- **name** (str | None) –
- **domain** (ArrayLike | None) –
- **size** (ArrayLike | None) –
- **comments** (Sequence | None) –

Return type None

is_domain_explicit() → bool

Return whether the *LUT* domain is explicit (or implicit).

An implicit domain is defined by its shape only:

```
[0 1]
```

While an explicit domain defines every single discrete samples:

```
[0.0 0.1 0.2 0.4 0.8 1.0]
```

Returns Is *LUT* domain explicit.

Return type bool

Examples

```
>>> LUT1D().is_domain_explicit()
False
>>> table = domain = np.linspace(0, 1, 10)
>>> LUT1D(table, domain=domain).is_domain_explicit()
True
```

static linear_table(*size: ArrayLike | None = None, domain: ArrayLike | None = None*) → NDArrayFloat

Return a linear table, the number of output samples n is equal to `size`.

Parameters

- **size** (ArrayLike | None) – Expected table size, default to 10.
- **domain** (ArrayLike | None) – Domain of the table.

Returns Linear table with `size` samples.

Return type numpy.ndarray

Examples

```
>>> LUT1D.linear_table(5, np.array([-0.1, 1.5]))
array([-0.1,  0.3,  0.7,  1.1,  1.5])
>>> LUT1D.linear_table(domain=np.linspace(-0.1, 1.5, 5))
array([-0.1,  0.3,  0.7,  1.1,  1.5])
```

invert(***kwargs: Any*) → Self

Compute and returns an inverse copy of the *LUT*.

Parameters **kwargs** (Any) – Keywords arguments, only given for signature compatibility with the `AbstractLUT.invert()` method.

Returns Inverse *LUT* class instance.

Return type colour.LUT1D

Examples

```
>>> LUT = LUT1D(LUT1D.linear_table() ** (1 / 2.2))
>>> print(LUT.table)
[ 0.        ...  0.3683438...  0.5047603...  0.6069133...  0.6916988...  0.
↪7655385...
  0.8316843...  0.8920493...  0.9478701...  1.        ]
>>> print(LUT.invert())
LUT1D - ... - Inverse
--------...----------

Dimensions : 1
Domain     : [ 0.          0.3683438...  0.5047603...  0.6069133...  0.6916988...
↪  0.7655385...
               0.8316843...  0.8920493...  0.9478701...  1.        ]
Size       : (10,)
>>> print(LUT.invert().table)
[ 0.        ...  0.1111111...  0.2222222...  0.3333333...  0.4444444...  0.
↪5555555...
  0.6666666...  0.7777777...  0.8888888...  1.        ]
```

apply(*RGB: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Apply the *LUT* to given *RGB* colourspace array using given method.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to apply the *LUT* onto.
- **direction** – Whether the *LUT* should be applied in the forward or inverse direction.
- **extrapolator** – Extrapolator class type or object to use as extrapolating function.
- **extrapolator_kwargs** – Arguments to use when instantiating or calling the extrapolating function.
- **interpolator** – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function.
- **kwargs** (Any) –

Returns Interpolated *RGB* colourspace array.

Return type numpy.ndarray

Examples

```
>>> LUT = LUT1D(LUT1D.linear_table() ** (1 / 2.2))
>>> RGB = np.array([0.18, 0.18, 0.18])
```

LUT applied to the given *RGB* colourspace in the forward direction:

```
>>> LUT.apply(RGB)
array([ 0.4529220...,  0.4529220...,  0.4529220...])
```

LUT applied to the modified *RGB* colourspace in the inverse direction:

```
>>> LUT.apply(LUT.apply(RGB), direction="Inverse")
...
array([ 0.18...,  0.18...,  0.18...])
```

convert(*cls: Type[colour.io.luts.lut.AbstractLUT], force_conversion: bool = False, **kwargs: Any*) → colour.io.luts.lut.AbstractLUT

Convert the *LUT* to given `cls` class instance.

Parameters

- **cls** (`Type[colour.io.luts.lut.AbstractLUT]`) – *LUT* class instance.
- **force_conversion** (`bool`) – Whether to force the conversion as it might be destructive.
- **interpolator** – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function.
- **size** – Expected table size in case of an upcast to a `LUT3D` class instance.
- **kwargs** (`Any`) –

Returns Converted *LUT* class instance.

Return type `colour.LUT1D` or `colour.LUT3x1D` or `colour.LUT3D`

> **Warning:** Some conversions are destructive and raise a `ValueError` exception by default.

Raises `ValueError` – If the conversion is destructive.

Parameters

- **cls** (`Type[colour.io.luts.lut.AbstractLUT]`) –
- **force_conversion** (`bool`) –
- **kwargs** (`Any`) –

Return type colour.io.luts.lut.AbstractLUT

Examples

```
>>> LUT = LUT1D()
>>> print(LUT.convert(LUT1D))
LUT1D - Unity 10 - Converted 1D to 1D
-------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT.convert(LUT3x1D))
LUT3x1D - Unity 10 - Converted 1D to 3x1D
-----------------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
>>> print(LUT.convert(LUT3D, force_conversion=True))
```

(continues on next page)

(continued from previous page)

```
LUT3D - Unity 10 - Converted 1D to 3D
-------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
```

colour.LUT3x1D

class colour.**LUT3x1D**(*table: ArrayLike | None = None, name: str | None = None, domain: ArrayLike | None = None, size: ArrayLike | None = None, comments: Sequence | None = None*)

Bases: `colour.io.luts.lut.AbstractLUT`

Define the base class for a 3x1D *LUT*.

Parameters

- **table** (`ArrayLike | None`) – Underlying *LUT* table.
- **name** (`str | None`) – *LUT* name.
- **domain** (`ArrayLike | None`) – *LUT* domain, also used to define the instantiation time default table domain.
- **size** (`ArrayLike | None`) – Size of the instantiation time default table, default to 10.
- **comments** (`Sequence | None`) – Comments to add to the *LUT*.

Return type None

Methods

- `__init__()`
- `is_domain_explicit()`
- `linear_table()`
- `invert()`
- `apply()`
- `convert()`

Examples

Instantiating a unity LUT with a table with 16x3 elements:

```
>>> print(LUT3x1D(size=16))
LUT3x1D - Unity 16
------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (16, 3)
```

Instantiating a LUT using a custom table with 16x3 elements:

```
>>> print(LUT3x1D(LUT3x1D.linear_table(16) ** (1 / 2.2)))
...
LUT3x1D - ...
----------...

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (16, 3)
```

Instantiating a LUT using a custom table with 16x3 elements, custom name, custom domain and comments:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> print(
...     LUT3x1D(
...         spow(LUT3x1D.linear_table(16), 1 / 2.2),
...         "My LUT",
...         domain,
...         comments=["A first comment.", "A second comment."],
...     )
... )
LUT3x1D - My LUT
----------------

Dimensions : 2
Domain     : [[-0.1 -0.2 -0.4]
              [ 1.5  3.   6. ]]
Size       : (16, 3)
Comment 01 : A first comment.
Comment 02 : A second comment.
```

`__init__`(*table: ArrayLike | None = None, name: str | None = None, domain: ArrayLike | None = None, size: ArrayLike | None = None, comments: Sequence | None = None*) → None

Parameters

- **table** (ArrayLike | None) –
- **name** (str | None) –
- **domain** (ArrayLike | None) –
- **size** (ArrayLike | None) –
- **comments** (Sequence | None) –

Return type None

`is_domain_explicit`() → bool

Return whether the *LUT* domain is explicit (or implicit).

An implicit domain is defined by its shape only:

```
[[0 1]
 [0 1]
 [0 1]]
```

While an explicit domain defines every single discrete samples:

```
[[0.0 0.0 0.0]
 [0.1 0.1 0.1]
 [0.2 0.2 0.2]
 [0.3 0.3 0.3]
 [0.4 0.4 0.4]
 [0.8 0.8 0.8]
 [1.0 1.0 1.0]]
```

Returns Is *LUT* domain explicit.

Return type `bool`

Examples

```
>>> LUT3x1D().is_domain_explicit()
False
>>> samples = np.linspace(0, 1, 10)
>>> table = domain = tstack([samples, samples, samples])
>>> LUT3x1D(table, domain=domain).is_domain_explicit()
True
```

static linear_table(*size: ArrayLike* | *None* = *None*, *domain: ArrayLike* | *None* = *None*) → NDArrayFloat

Return a linear table, the number of output samples n is equal to `size * 3` or `size[0] + size[1] + size[2]`.

Parameters

- **size** (`ArrayLike | None`) – Expected table size, default to 10.
- **domain** (`ArrayLike | None`) – Domain of the table.

Returns Linear table with `size * 3` or `size[0] + size[1] + size[2]` samples.

Return type `numpy.ndarray`

> **Warning:** If `size` is non uniform, the linear table will be padded accordingly.

Examples

```
>>> LUT3x1D.linear_table(
...     5, np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
... )
array([[-0.1, -0.2, -0.4],
       [ 0.3,  0.6,  1.2],
       [ 0.7,  1.4,  2.8],
       [ 1.1,  2.2,  4.4],
       [ 1.5,  3. ,  6. ]])
>>> LUT3x1D.linear_table(
...     np.array([5, 3, 2]),
...     np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]),
... )
array([[-0.1, -0.2, -0.4],
       [ 0.3,  1.4,  6. ],
       [ 0.7,  3. ,  nan],
```

(continues on next page)

(continued from previous page)

```
       [ 1.1,   nan,   nan],
       [ 1.5,   nan,   nan]])
>>> domain = np.array(
...     [
...         [-0.1, -0.2, -0.4],
...         [0.3, 1.4, 6.0],
...         [0.7, 3.0, np.nan],
...         [1.1, np.nan, np.nan],
...         [1.5, np.nan, np.nan],
...     ]
... )
>>> LUT3x1D.linear_table(domain=domain)
array([[-0.1, -0.2, -0.4],
       [ 0.3,   1.4,   6. ],
       [ 0.7,   3. ,   nan],
       [ 1.1,   nan,   nan],
       [ 1.5,   nan,   nan]])
```

invert(***kwargs: Any*) → Self

Compute and returns an inverse copy of the *LUT*.

Parameters **kwargs** (Any) – Keywords arguments, only given for signature compatibility with the `AbstractLUT.invert()` method.

Returns Inverse *LUT* class instance.

Return type `colour.LUT3x1D`

Examples

```
>>> LUT = LUT3x1D(LUT3x1D.linear_table() ** (1 / 2.2))
>>> print(LUT.table)
[[ 0.          0.          0.        ]
 [ 0.36834383  0.36834383  0.36834383]
 [ 0.50476034  0.50476034  0.50476034]
 [ 0.60691337  0.60691337  0.60691337]
 [ 0.69169882  0.69169882  0.69169882]
 [ 0.76553851  0.76553851  0.76553851]
 [ 0.83168433  0.83168433  0.83168433]
 [ 0.89204934  0.89204934  0.89204934]
 [ 0.94787016  0.94787016  0.94787016]
 [ 1.          1.          1.        ]]
>>> print(LUT.invert())
LUT3x1D - ... - Inverse
----------...----------

Dimensions : 2
Domain     : [[ 0.         ...  0.         ...  0.         ...]
              [ 0.3683438...  0.3683438...  0.3683438...]
              [ 0.5047603...  0.5047603...  0.5047603...]
              [ 0.6069133...  0.6069133...  0.6069133...]
              [ 0.6916988...  0.6916988...  0.6916988...]
              [ 0.7655385...  0.7655385...  0.7655385...]
              [ 0.8316843...  0.8316843...  0.8316843...]
              [ 0.8920493...  0.8920493...  0.8920493...]
              [ 0.9478701...  0.9478701...  0.9478701...]
```

(continues on next page)

(continued from previous page)

```
              [ 1.         ...  1.         ...  1.         ...]]
Size        : (10, 3)
>>> print(LUT.invert().table)
[[ 0.         ...  0.         ...  0.         ...]
 [ 0.1111111...  0.1111111...  0.1111111...]
 [ 0.2222222...  0.2222222...  0.2222222...]
 [ 0.3333333...  0.3333333...  0.3333333...]
 [ 0.4444444...  0.4444444...  0.4444444...]
 [ 0.5555555...  0.5555555...  0.5555555...]
 [ 0.6666666...  0.6666666...  0.6666666...]
 [ 0.7777777...  0.7777777...  0.7777777...]
 [ 0.8888888...  0.8888888...  0.8888888...]
 [ 1.         ...  1.         ...  1.         ...]]
```

apply(*RGB: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Apply the *LUT* to given *RGB* colourspace array using given method.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to apply the *LUT* onto.
- **direction** – Whether the *LUT* should be applied in the forward or inverse direction.
- **extrapolator** – Extrapolator class type or object to use as extrapolating function.
- **extrapolator_kwargs** – Arguments to use when instantiating or calling the extrapolating function.
- **interpolator** – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function.
- **kwargs** (Any) –

Returns Interpolated *RGB* colourspace array.

Return type numpy.ndarray

Examples

```
>>> LUT = LUT3x1D(LUT3x1D.linear_table() ** (1 / 2.2))
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4529220...,  0.4529220...,  0.4529220...])
>>> LUT.apply(LUT.apply(RGB), direction="Inverse")
...
array([ 0.18...,  0.18...,  0.18...])
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> table = spow(LUT3x1D.linear_table(domain=domain), 1 / 2.2)
>>> LUT = LUT3x1D(table, domain=domain)
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4423903...,  0.4503801...,  0.3581625...])
>>> domain = np.array(
...     [
...         [-0.1, -0.2, -0.4],
```

(continues on next page)

(continued from previous page)

```
...         [0.3, 1.4, 6.0],
...         [0.7, 3.0, np.nan],
...         [1.1, np.nan, np.nan],
...         [1.5, np.nan, np.nan],
...     ]
... )
>>> table = spow(LUT3x1D.linear_table(domain=domain), 1 / 2.2)
>>> LUT = LUT3x1D(table, domain=domain)
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.2996370..., -0.0901332..., -0.3949770...])
```

convert(*cls: Type[colour.io.luts.lut.AbstractLUT]*, *force_conversion: bool = False*, ***kwargs: Any*) → colour.io.luts.lut.AbstractLUT

Convert the *LUT* to given `cls` class instance.

Parameters

- **cls** (`Type[colour.io.luts.lut.AbstractLUT]`) – *LUT* class instance.
- **force_conversion** (`bool`) – Whether to force the conversion as it might be destructive.
- **interpolator** – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function.
- **size** – Expected table size in case of an upcast to a `LUT3D` class instance.
- **kwargs** (`Any`) –

Returns Converted *LUT* class instance.

Return type `colour.LUT1D` or `colour.LUT3x1D` or `colour.LUT3D`

> **Warning:** Some conversions are destructive and raise a `ValueError` exception by default.

Raises `ValueError` – If the conversion is destructive.

Parameters

- **cls** (`Type[colour.io.luts.lut.AbstractLUT]`) –
- **force_conversion** (`bool`) –
- **kwargs** (`Any`) –

Return type colour.io.luts.lut.AbstractLUT

Examples

```
>>> LUT = LUT3x1D()
>>> print(LUT.convert(LUT1D, force_conversion=True))
LUT1D - Unity 10 - Converted 3x1D to 1D
---------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
```

(continues on next page)

(continued from previous page)

```
>>> print(LUT.convert(LUT3x1D))
LUT3x1D - Unity 10 - Converted 3x1D to 3x1D
-------------------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
>>> print(LUT.convert(LUT3D, force_conversion=True))
LUT3D - Unity 10 - Converted 3x1D to 3D
---------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
```

colour.LUT3D

class colour.**LUT3D**(*table: ArrayLike | None = None, name: str | None = None, domain: ArrayLike | None = None, size: ArrayLike | None = None, comments: Sequence | None = None*)

Bases: `colour.io.luts.lut.AbstractLUT`

Define the base class for a 3D *LUT*.

Parameters

- **table** (ArrayLike | None) – Underlying *LUT* table.
- **name** (str | None) – *LUT* name.
- **domain** (ArrayLike | None) – *LUT* domain, also used to define the instantiation time default table domain.
- **size** (ArrayLike | None) – Size of the instantiation time default table, default to 33.
- **comments** (Sequence | None) – Comments to add to the *LUT*.

Return type None

Methods

- `__init__()`
- `is_domain_explicit()`
- `linear_table()`
- `invert()`
- `apply()`
- `convert()`

Examples

Instantiating a unity LUT with a table with 16x16x16x3 elements:

```
>>> print(LUT3D(size=16))
LUT3D - Unity 16
----------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (16, 16, 16, 3)
```

Instantiating a LUT using a custom table with 16x16x16x3 elements:

```
>>> print(LUT3D(LUT3D.linear_table(16) ** (1 / 2.2)))
LUT3D - ...
--------...

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (16, 16, 16, 3)
```

Instantiating a LUT using a custom table with 16x16x16x3 elements, custom name, custom domain and comments:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> print(
...     LUT3D(
...         spow(LUT3D.linear_table(16), 1 / 2.2),
...         "My LUT",
...         domain,
...         comments=["A first comment.", "A second comment."],
...     )
... )
LUT3D - My LUT
--------------

Dimensions : 3
Domain     : [[-0.1 -0.2 -0.4]
              [ 1.5  3.   6. ]]
Size       : (16, 16, 16, 3)
Comment 01 : A first comment.
Comment 02 : A second comment.
```

__init__(*table: ArrayLike | None = None, name: str | None = None, domain: ArrayLike | None = None, size: ArrayLike | None = None, comments: Sequence | None = None*) → None

Parameters

- **table** (ArrayLike | None) –
- **name** (str | None) –
- **domain** (ArrayLike | None) –
- **size** (ArrayLike | None) –
- **comments** (Sequence | None) –

Return type None

is_domain_explicit() → bool

Return whether the *LUT* domain is explicit (or implicit).

An implicit domain is defined by its shape only:

```
[[0 0 0]
 [1 1 1]]
```

While an explicit domain defines every single discrete samples:

```
[[0.0 0.0 0.0]
 [0.1 0.1 0.1]
 [0.2 0.2 0.2]
 [0.3 0.3 0.3]
 [0.4 0.4 0.4]
 [0.8 0.8 0.8]
 [1.0 1.0 1.0]]
```

Returns Is *LUT* domain explicit.

Return type bool

Examples

```
>>> LUT3D().is_domain_explicit()
False
>>> domain = np.array(
...     [[-0.1, -0.2, -0.4], [0.7, 1.4, 6.0], [1.5, 3.0, np.nan]]
... )
>>> LUT3D(domain=domain).is_domain_explicit()
True
```

static linear_table(*size: ArrayLike | None = None, domain: ArrayLike | None = None*) → NDArrayFloat

Return a linear table, the number of output samples n is equal to `size**3 * 3` or `size[0] * size[1] * size[2] * 3`.

Parameters

- **size** (`ArrayLike | None`) – Expected table size, default to 33.
- **domain** (`ArrayLike | None`) – Domain of the table.

Returns Linear table with `size**3 * 3` or `size[0] * size[1] * size[2] * 3` samples.

Return type numpy.ndarray

Examples

```
>>> LUT3D.linear_table(
...     3, np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
... )
array([[[[-0.1, -0.2, -0.4],
         [-0.1, -0.2,  2.8],
         [-0.1, -0.2,  6. ]],

        [[-0.1,  1.4, -0.4],
```

(continues on next page)

(continued from previous page)

```
         [-0.1,  1.4,  2.8],
         [-0.1,  1.4,  6. ]],

        [[-0.1,  3. , -0.4],
         [-0.1,  3. ,  2.8],
         [-0.1,  3. ,  6. ]]],


       [[[ 0.7, -0.2, -0.4],
         [ 0.7, -0.2,  2.8],
         [ 0.7, -0.2,  6. ]],

        [[ 0.7,  1.4, -0.4],
         [ 0.7,  1.4,  2.8],
         [ 0.7,  1.4,  6. ]],

        [[ 0.7,  3. , -0.4],
         [ 0.7,  3. ,  2.8],
         [ 0.7,  3. ,  6. ]]],


       [[[ 1.5, -0.2, -0.4],
         [ 1.5, -0.2,  2.8],
         [ 1.5, -0.2,  6. ]],

        [[ 1.5,  1.4, -0.4],
         [ 1.5,  1.4,  2.8],
         [ 1.5,  1.4,  6. ]],

        [[ 1.5,  3. , -0.4],
         [ 1.5,  3. ,  2.8],
         [ 1.5,  3. ,  6. ]]]])
>>> LUT3D.linear_table(
...     np.array([3, 3, 2]),
...     np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]),
... )
array([[[[-0.1, -0.2, -0.4],
         [-0.1, -0.2,  6. ]],

        [[-0.1,  1.4, -0.4],
         [-0.1,  1.4,  6. ]],

        [[-0.1,  3. , -0.4],
         [-0.1,  3. ,  6. ]]],


       [[[ 0.7, -0.2, -0.4],
         [ 0.7, -0.2,  6. ]],

        [[ 0.7,  1.4, -0.4],
         [ 0.7,  1.4,  6. ]],

        [[ 0.7,  3. , -0.4],
         [ 0.7,  3. ,  6. ]]],
```

(continues on next page)

(continued from previous page)

```
       [[[ 1.5, -0.2, -0.4],
         [ 1.5, -0.2,  6. ]],

        [[ 1.5,  1.4, -0.4],
         [ 1.5,  1.4,  6. ]],

        [[ 1.5,  3. , -0.4],
         [ 1.5,  3. ,  6. ]]]])
>>> domain = np.array(
...     [[-0.1, -0.2, -0.4], [0.7, 1.4, 6.0], [1.5, 3.0, np.nan]]
... )
>>> LUT3D.linear_table(domain=domain)
array([[[[-0.1, -0.2, -0.4],
         [-0.1, -0.2,  6. ]],

        [[-0.1,  1.4, -0.4],
         [-0.1,  1.4,  6. ]],

        [[-0.1,  3. , -0.4],
         [-0.1,  3. ,  6. ]]],


       [[[ 0.7, -0.2, -0.4],
         [ 0.7, -0.2,  6. ]],

        [[ 0.7,  1.4, -0.4],
         [ 0.7,  1.4,  6. ]],

        [[ 0.7,  3. , -0.4],
         [ 0.7,  3. ,  6. ]]],


       [[[ 1.5, -0.2, -0.4],
         [ 1.5, -0.2,  6. ]],

        [[ 1.5,  1.4, -0.4],
         [ 1.5,  1.4,  6. ]],

        [[ 1.5,  3. , -0.4],
         [ 1.5,  3. ,  6. ]]]])
```

invert(***kwargs: Any*) → Self

Compute and returns an inverse copy of the *LUT*.

Parameters

- **extrapolate** – Whether to extrapolate the *LUT* when computing its inverse. Extrapolation is performed by reflecting the *LUT* cube along its 8 faces. Note that the domain is extended beyond [0, 1], thus the *LUT* might not be handled properly in other software.
- **interpolator** – Interpolator class type or object to use as interpolating function.
- **query_size** – Number of points to query in the KDTree, their mean is computed, resulting in a smoother result.
- **size** – Size of the inverse *LUT*. With the given implementation, it is good practise to double the size of the inverse *LUT* to provide a smoother result. If `size`

is not given, $2^{\sqrt{size_{LUT}}+1} + 1$ will be used instead.

- **kwargs** (Any) –

Returns Inverse *LUT* class instance.

Return type colour.LUT3D

Examples

```
>>> LUT = LUT3D()
>>> print(LUT)
LUT3D - Unity 33
----------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
>>> print(LUT.invert())
LUT3D - Unity 33 - Inverse
--------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (108, 108, 108, 3)
```

apply(*RGB: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Apply the *LUT* to given *RGB* colourspace array using given method.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to apply the *LUT* onto.
- **direction** – Whether the *LUT* should be applied in the forward or inverse direction.
- **extrapolate** – Whether to extrapolate the *LUT* when computing its inverse. Extrapolation is performed by reflecting the *LUT* cube along its 8 faces.
- **interpolator** – Interpolator object to use as interpolating function.
- **interpolator_kwargs** – Arguments to use when calling the interpolating function.
- **query_size** – Number of points to query in the KDTree, their mean is computed, resulting in a smoother result.
- **size** – Size of the inverse *LUT*. With the given implementation, it is good practise to double the size of the inverse *LUT* to provide a smoother result. If size is not given, $2^{\sqrt{size_{LUT}}+1} + 1$ will be used instead.
- **kwargs** (Any) –

Returns Interpolated *RGB* colourspace array.

Return type numpy.ndarray

Examples

```
>>> LUT = LUT3D(LUT3D.linear_table() ** (1 / 2.2))
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4583277...,  0.4583277...,  0.4583277...])
>>> LUT.apply(LUT.apply(RGB), direction="Inverse")
...
array([ 0.1781995...,  0.1809414...,  0.1809513...])
>>> from colour.algebra import spow
>>> domain = np.array(
...     [
...         [-0.1, -0.2, -0.4],
...         [0.3, 1.4, 6.0],
...         [0.7, 3.0, np.nan],
...         [1.1, np.nan, np.nan],
...         [1.5, np.nan, np.nan],
...     ]
... )
>>> table = spow(LUT3D.linear_table(domain=domain), 1 / 2.2)
>>> LUT = LUT3D(table, domain=domain)
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.2996370..., -0.0901332..., -0.3949770...])
```

convert(*cls: Type[colour.io.luts.lut.AbstractLUT], force_conversion: bool = False, **kwargs: Any*) → colour.io.luts.lut.AbstractLUT

Convert the *LUT* to given `cls` class instance.

Parameters

- **cls** (Type[`colour.io.luts.lut.AbstractLUT`]) – *LUT* class instance.
- **force_conversion** (bool) – Whether to force the conversion as it might be destructive.
- **interpolator** – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function.
- **size** – Expected table size in case of a downcast from a `LUT3D` class instance.
- **kwargs** (Any) –

Returns Converted *LUT* class instance.

Return type `colour.LUT1D` or `colour.LUT3x1D` or `colour.LUT3D`

> **Warning:** Some conversions are destructive and raise a `ValueError` exception by default.

Raises `ValueError` – If the conversion is destructive.

Parameters

- **cls** (Type[`colour.io.luts.lut.AbstractLUT`]) –
- **force_conversion** (bool) –
- **kwargs** (Any) –

Return type colour.io.luts.lut.AbstractLUT

Examples

```
>>> LUT = LUT3D()
>>> print(LUT.convert(LUT1D, force_conversion=True))
LUT1D - Unity 33 - Converted 3D to 1D
-------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT.convert(LUT3x1D, force_conversion=True))
LUT3x1D - Unity 33 - Converted 3D to 3x1D
-----------------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
>>> print(LUT.convert(LUT3D))
LUT3D - Unity 33 - Converted 3D to 3D
-------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
```

colour.LUTOperatorMatrix

class colour.**LUTOperatorMatrix**(*matrix: ArrayLike | None = None*, *offset: ArrayLike | None = None*, **args: Any*, ***kwargs: Any*)

Bases: `colour.io.luts.operator.AbstractLUTSequenceOperator`

Define the *LUT* operator supporting a 3x3 or 4x4 matrix and an offset vector.

Parameters

- **matrix** (`ArrayLike | None`) – 3x3 or 4x4 matrix for the operator.
- **offset** (`ArrayLike | None`) – Offset for the operator.
- **name** – *LUT* operator name.
- **comments** – Comments to add to the *LUT* operator.
- **args** (`Any`) –
- **kwargs** (`Any`) –

Return type None

Attributes

- matrix()
- offset()

Methods

- __str__()
- __repr__()
- __eq__()
- __ne__()
- apply()

Notes

- The internal colour.io.Matrix.matrix and colour.io.Matrix.offset properties are reshaped to (4, 4) and (4,) respectively.

Examples

Instantiating an identity matrix:

```
>>> print(LUTOperatorMatrix(name="Identity"))
LUTOperatorMatrix - Identity
----------------------------

Matrix     : [[ 1.  0.  0.  0.]
              [ 0.  1.  0.  0.]
              [ 0.  0.  1.  0.]
              [ 0.  0.  0.  1.]]
Offset     : [ 0.  0.  0.  0.]
```

Instantiating a matrix with comments:

```
>>> matrix = np.array(
...     [
...         [1.45143932, -0.23651075, -0.21492857],
...         [-0.07655377, 1.1762297, -0.09967593],
...         [0.00831615, -0.00603245, 0.9977163],
...     ]
... )
>>> print(
...     LUTOperatorMatrix(
...         matrix,
...         name="AP0 to AP1",
...         comments=["A first comment.", "A second comment."],
...     )
... )
LUTOperatorMatrix - AP0 to AP1
------------------------------

Matrix     : [[ 1.45143932 -0.23651075 -0.21492857  0.        ]
              [-0.07655377  1.1762297  -0.09967593  0.        ]
```

(continues on next page)

(continued from previous page)

```
              [ 0.00831615 -0.00603245  0.9977163   0.        ]
              [ 0.          0.          0.          1.        ]]
Offset     : [ 0.  0.  0.  0.]

A first comment.
A second comment.
```

__init__(*matrix: ArrayLike | None = None*, *offset: ArrayLike | None = None*, **args: Any*, ***kwargs: Any*) → None

Parameters

- **matrix** (ArrayLike | None) –
- **offset** (ArrayLike | None) –
- **args** (Any) –
- **kwargs** (Any) –

Return type None

__hash__ = None

property matrix: NDArrayFloat

Getter and setter property for the *LUT* operator matrix.

Parameters value – Value to set the *LUT* operator matrix with.

Returns Operator matrix.

Return type numpy.ndarray

property offset: NDArrayFloat

Getter and setter property for the *LUT* operator offset.

Parameters value – Value to set the *LUT* operator offset with.

Returns Operator offset.

Return type numpy.ndarray

__str__() → str

Return a formatted string representation of the *LUT* operator.

Returns Formatted string representation.

Return type str

Examples

```
>>> print(LUTOperatorMatrix())
LUTOperatorMatrix - LUT Sequence Operator ...
------------------------------------------...

Matrix     : [[ 1.  0.  0.  0.]
              [ 0.  1.  0.  0.]
              [ 0.  0.  1.  0.]
              [ 0.  0.  0.  1.]]
Offset     : [ 0.  0.  0.  0.]
```

__repr__() → str

Return an evaluable string representation of the *LUT* operator.

Returns Evaluable string representation.

Return type str

Examples

```
>>> LUTOperatorMatrix(
...     comments=["A first comment.", "A second comment."]
... )
...
LUTOperatorMatrix([[ 1.,  0.,  0.,  0.],
                   [ 0.,  1.,  0.,  0.],
                   [ 0.,  0.,  1.,  0.],
                   [ 0.,  0.,  0.,  1.]],
                  [ 0.,  0.,  0.,  0.],
                  name='LUT Sequence Operator ...',
                  comments=['A first comment.', 'A second comment.'])
```

__eq__(*other: Any*) → bool

Return whether the *LUT* operator is equal to given other object.

Parameters **other** (Any) – Object to test whether it is equal to the *LUT* operator.

Returns Whether given object equal to the *LUT* operator.

Return type bool

Examples

```
>>> LUTOperatorMatrix() == LUTOperatorMatrix()
True
```

__ne__(*other: Any*) → bool

Return whether the *LUT* operator is not equal to given other object.

Parameters **other** (Any) – Object to test whether it is not equal to the *LUT* operator.

Returns Whether given object is not equal to the *LUT* operator.

Return type bool

Examples

```
>>> LUTOperatorMatrix() != LUTOperatorMatrix(
...     np.linspace(0, 1, 16).reshape([4, 4])
... )
True
```

apply(*RGB: ArrayLike*, **args: Any*, ***kwargs: Any*) → NDArrayFloat

Apply the *LUT* operator to given *RGB* array.

Parameters

- **RGB** (ArrayLike) – *RGB* array to apply the *LUT* operator transform to.
- **apply_offset_first** – Whether to apply the offset first and then the matrix.

- **args** (Any) –
- **kwargs** (Any) –

Returns Transformed *RGB* array.

Return type numpy.ndarray

Examples

```
>>> matrix = np.array(
...     [
...         [1.45143932, -0.23651075, -0.21492857],
...         [-0.07655377, 1.1762297, -0.09967593],
...         [0.00831615, -0.00603245, 0.9977163],
...     ]
... )
>>> M = LUTOperatorMatrix(matrix)
>>> RGB = np.array([0.3, 0.4, 0.5])
>>> M.apply(RGB)
array([ 0.2333632...,  0.3976877...,  0.4989400...])
```

colour.LUTSequence

class colour.**LUTSequence**(**args: colour.hints.ProtocolLUTSequenceItem*)

Bases: collections.abc.MutableSequence

Define the base class for a *LUT* sequence, i.e. a series of *LUTs*, *LUT* operators or objects implementing the colour.hints.ProtocolLUTSequenceItem protocol.

The *colour.LUTSequence* class can be used to model series of *LUTs* such as when a shaper *LUT* is combined with a 3D *LUT*.

Parameters **args** (ProtocolLUTSequenceItem) – Sequence of objects implementing the colour.hints.ProtocolLUTSequenceItem protocol.

Return type None

Attributes

- sequence

Methods

- __init__()
- __getitem__()
- __setitem__()
- __delitem__()
- __len__()
- __str__()
- __repr__()
- __eq__()
- __ne__()

- insert()
- apply()
- copy()

Examples

```
>>> from colour.io.luts import LUT1D, LUT3x1D, LUT3D
>>> LUT_1 = LUT1D()
>>> LUT_2 = LUT3D(size=3)
>>> LUT_3 = LUT3x1D()
>>> print(LUTSequence(LUT_1, LUT_2, LUT_3))
LUT Sequence
------------

Overview

    LUT1D --> LUT3D --> LUT3x1D

Operations

    LUT1D - Unity 10
    ----------------

    Dimensions : 1
    Domain     : [ 0.  1.]
    Size       : (10,)

    LUT3D - Unity 3
    ---------------

    Dimensions : 3
    Domain     : [[ 0.  0.  0.]
                  [ 1.  1.  1.]]
    Size       : (3, 3, 3, 3)

    LUT3x1D - Unity 10
    ------------------

    Dimensions : 2
    Domain     : [[ 0.  0.  0.]
                  [ 1.  1.  1.]]
    Size       : (10, 3)
```

__init__(**args: colour.hints.ProtocolLUTSequenceItem*) → None

Parameters **args** (colour.hints.ProtocolLUTSequenceItem) –

Return type None

property sequence: List[colour.hints.ProtocolLUTSequenceItem]

Getter and setter property for the underlying *LUT* sequence.

Parameters **value** – Value to set the underlying *LUT* sequence with.

Returns Underlying *LUT* sequence.

Return type list

`__getitem__`(*index: int | slice*) → *Any*

Return the *LUT* sequence item(s) at given index (or slice).

Parameters **`index`** (`int | slice`) – Index (or slice) to return the *LUT* sequence item(s) at.

Returns *LUT* sequence item(s) at given index (or slice).

Return type *ProtocolLUTSequenceItem*

`__setitem__`(*index: int | slice*, *value: Any*)

Set the *LUT* sequence at given index (or slice) with given value.

Parameters

- **`index`** (`int | slice`) – Index (or slice) to set the *LUT* sequence value at.
- **`value`** (`Any`) – Value to set the *LUT* sequence with.

`__delitem__`(*index: int | slice*)

Delete the *LUT* sequence item(s) at given index (or slice).

Parameters **`index`** (`int | slice`) – Index (or slice) to delete the *LUT* sequence items at.

`__len__`() → int

Return the *LUT* sequence items count.

Returns *LUT* sequence items count.

Return type `int`

`__str__`() → str

Return a formatted string representation of the *LUT* sequence.

Returns Formatted string representation.

Return type `str`

`__repr__`() → str

Return an evaluable string representation of the *LUT* sequence.

Returns Evaluable string representation.

Return type `str`

`__eq__`(*other*) → bool

Return whether the *LUT* sequence is equal to given other object.

Parameters **`other`** – Object to test whether it is equal to the *LUT* sequence.

Returns Whether given object is equal to the *LUT* sequence.

Return type `bool`

`__ne__`(*other*) → bool

Return whether the *LUT* sequence is not equal to given other object.

Parameters **`other`** – Object to test whether it is not equal to the *LUT* sequence.

Returns Whether given object is not equal to the *LUT* sequence.

Return type `bool`

`insert`(*index: int*, *item: colour.hints.ProtocolLUTSequenceItem*)

Insert given *LUT* at given index into the *LUT* sequence.

Parameters

- **`index`** (`int`) – Index to insert the item at into the *LUT* sequence.

- **item** (colour.hints.ProtocolLUTSequenceItem) – *LUT* to insert into the *LUT* sequence.

__hash__ = None

__weakref__

list of weak references to the object (if defined)

apply(*RGB: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Apply the *LUT* sequence sequentially to given *RGB* colourspace array.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to apply the *LUT* sequence sequentially onto.
- **kwargs** (Any) – Keywords arguments, the keys must be the class type names for which they are intended to be used with. There is no implemented way to discriminate which class instance the keyword arguments should be used with, thus if many class instances of the same type are members of the sequence, any matching keyword arguments will be used with all the class instances.

Returns Processed *RGB* colourspace array.

Return type numpy.ndarray

Examples

```
>>> import numpy as np
>>> from colour.io.luts import LUT1D, LUT3x1D, LUT3D
>>> from colour.utilities import tstack
>>> LUT_1 = LUT1D(LUT1D.linear_table(16) + 0.125)
>>> LUT_2 = LUT3D(LUT3D.linear_table(16) ** (1 / 2.2))
>>> LUT_3 = LUT3x1D(LUT3x1D.linear_table(16) * 0.750)
>>> LUT_sequence = LUTSequence(LUT_1, LUT_2, LUT_3)
>>> samples = np.linspace(0, 1, 5)
>>> RGB = tstack([samples, samples, samples])
>>> LUT_sequence.apply(RGB, LUT1D={"direction": "Inverse"})
...
array([[ 0.        ...,  0.        ...,  0.        ...],
       [ 0.2899886...,  0.2899886...,  0.2899886...],
       [ 0.4797662...,  0.4797662...,  0.4797662...],
       [ 0.6055328...,  0.6055328...,  0.6055328...],
       [ 0.7057779...,  0.7057779...,  0.7057779...]])
```

copy() → *colour.io.luts.sequence.LUTSequence*

Return a copy of the *LUT* sequence.

Returns *LUT* sequence copy.

Return type colour.LUTSequence

read_LUT(path[, method])	Read given *LUT* file using given method.
write_LUT(LUT, path[, decimals, method])	Write given *LUT* to given file using given method.

colour.read_LUT

colour.**read_LUT**(*path: str, method: Optional[Union[Literal['Cinespace', 'Iridas Cube', 'Resolve Cube', 'Sony SPI1D', 'Sony SPI3D', 'Sony SPImtx'], str]] = None, **kwargs: Any*) → *colour.io.luts.lut.LUT1D* | *colour.io.luts.lut.LUT3x1D* | *colour.io.luts.lut.LUT3D* | *colour.io.luts.sequence.LUTSequence* | *colour.io.luts.operator.LUTOperatorMatrix*

Read given *LUT* file using given method.

Parameters

- **path** (str) – *LUT* path.
- **method** (Optional[Union[Literal['Cinespace', 'Iridas Cube', 'Resolve Cube', 'Sony SPI1D', 'Sony SPI3D', 'Sony SPImtx'], str]]) – Reading method, if *None*, the method will be auto-detected according to extension.
- **kwargs** (Any) –

Returns colour.LUT1D or colour.LUT3x1D or colour.LUT3D or colour.LUTSequence or colour.LUTOperatorMatrix class instance.

Return type colour.LUT1D or colour.LUT3x1D or colour.LUT3D or colour.LUTSequence or colour.LUTOperatorMatrix

References

[AdobeSystems13c], [Cha15], [RisingSResearch]

Examples

Reading a 3x1D *Iridas .cube LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "iridas_cube",
...     "ACES_Proxy_10_to_ACES.cube",
... )
>>> print(read_LUT(path))
LUT3x1D - ACES Proxy 10 to ACES
-------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (32, 3)
```

Reading a 1D *Sony .spi1d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "sony_spi1d",
...     "eotf_sRGB_1D.spi1d",
... )
>>> print(read_LUT(path))
LUT1D - eotf sRGB 1D
```

(continues on next page)

(continued from previous page)

```
--------------------

Dimensions : 1
Domain     : [-0.1  1.5]
Size       : (16,)
Comment 01 : Generated by "Colour 0.3.11".
Comment 02 : "colour.models.eotf_sRGB".
```

Reading a 3D *Sony .spi3d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "sony_spi3d",
...     "Colour_Correct.spi3d",
... )
>>> print(read_LUT(path))
LUT3D - Colour Correct
----------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (4, 4, 4, 3)
Comment 01 : Adapted from a LUT generated by Foundry::LUT.
```

Reading a *Sony .spimtx LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "sony_spimtx",
...     "dt.spimtx",
... )
>>> print(read_LUT(path))
LUTOperatorMatrix - dt
----------------------

Matrix     : [[ 0.864274  0.        0.        0.       ]
              [ 0.        0.864274  0.        0.       ]
              [ 0.        0.        0.864274  0.       ]
              [ 0.        0.        0.        1.       ]]
Offset     : [ 0.  0.  0.  0.]
```

colour.write_LUT

colour.**write_LUT**(*LUT: colour.io.luts.lut.LUT1D | colour.io.luts.lut.LUT3x1D | colour.io.luts.lut.LUT3D | colour.io.luts.sequence.LUTSequence | colour.io.luts.operator.LUTOperatorMatrix, path: str, decimals: int = 7, method: Optional[Union[Literal['Cinespace', 'Iridas Cube', 'Resolve Cube', 'Sony SPI1D', 'Sony SPI3D', 'Sony SPImtx'], str]] = None, **kwargs: Any*) → bool

Write given *LUT* to given file using given method.

Parameters

- **LUT** (colour.io.luts.lut.LUT1D | colour.io.luts.lut.LUT3x1D | colour.io.luts.lut.LUT3D | colour.io.luts.sequence.LUTSequence | colour.io.luts.operator.LUTOperatorMatrix) – colour.LUT1D or colour.LUT3x1D or colour.LUT3D or colour.LUTSequence or colour.LUTOperatorMatrix class instance to write at given path.
- **path** (str) – *LUT* path.
- **decimals** (int) – Formatting decimals.
- **method** (Optional[Union[Literal['Cinespace', 'Iridas Cube', 'Resolve Cube', 'Sony SPI1D', 'Sony SPI3D', 'Sony SPImtx'], str]]) – Writing method, if *None*, the method will be auto-detected according to extension.
- **kwargs** (Any) –

Returns Definition success.

Return type bool

References

[AdobeSystems13c], [Cha15], [RisingSResearch]

Examples

Writing a 3x1D *Iridas .cube LUT*:

```
>>> import numpy as np
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16, domain), 1 / 2.2),
...     "My LUT",
...     domain,
...     comments=["A first comment.", "A second comment."],
... )
>>> write_LUT(LUT, "My_LUT.cube")
```

Writing a 1D *Sony .spi1d LUT*:

```
>>> domain = np.array([-0.1, 1.5])
>>> LUT = LUT1D(
...     spow(LUT1D.linear_table(16, domain), 1 / 2.2),
...     "My LUT",
...     domain,
...     comments=["A first comment.", "A second comment."],
... )
>>> write_LUT(LUT, "My_LUT.spi1d")
```

Writing a 3D *Sony .spi3d LUT*:

```
>>> LUT = LUT3D(
...     LUT3D.linear_table(16) ** (1 / 2.2),
...     "My LUT",
...     np.array([[0, 0, 0], [1, 1, 1]]),
...     comments=["A first comment.", "A second comment."],
... )
>>> write_LUT(LUT, "My_LUT.cube")
```

Ancillary Objects

colour.io

AbstractLUTSequenceOperator([name, comments])	Define the base class for *LUT* sequence operators.

colour.io.AbstractLUTSequenceOperator

class colour.io.**AbstractLUTSequenceOperator**(*name: str | None = None, comments: collections.abc.Sequence[str] | None = None*)

Bases: abc.ABC

Define the base class for *LUT* sequence operators.

This is an ABCMeta abstract class that must be inherited by sub-classes.

Parameters

- **name** (str | None) – *LUT* sequence operator name.
- **comments** (Sequence[str] | None) – Comments to add to the *LUT* sequence operator.

Return type None

Attributes

- name
- comments

Methods

- apply()

__init__(*name: str | None = None, comments: collections.abc.Sequence[str] | None = None*) → None

Parameters

- **name** (str | None) –
- **comments** (collections.abc.Sequence[str] | None) –

Return type None

property name: str

Getter and setter property for the *LUT* name.

Parameters **value** – Value to set the *LUT* name with.

Returns *LUT* name.

Return type str

property comments: List[str]

Getter and setter property for the *LUT* comments.

Parameters **value** – Value to set the *LUT* comments with.

Returns *LUT* comments.

Return type list

abstract apply(*RGB: ArrayLike*, **args: Any*, ***kwargs: Any*) → NDArrayFloat

Apply the *LUT* sequence operator to given *RGB* colourspace array.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array to apply the *LUT* sequence operator onto.
- **args** (Any) – Arguments.
- **kwargs** (Any) – Keywords arguments.

Returns Processed *RGB* colourspace array.

Return type numpy.ndarray

__weakref__

list of weak references to the object (if defined)

LUT_to_LUT(LUT, cls[, force_conversion])	Convert given *LUT* to given cls class instance.
read_LUT_Cinespace(path)	Read given *Cinespace .csp LUT* file.
write_LUT_Cinespace(LUT, path[, decimals])	Write given *LUT* to given *Cinespace .csp LUT* file.
read_LUT_IridasCube(path)	Read given *Iridas .cube LUT* file.
write_LUT_IridasCube(LUT, path[, decimals])	Write given *LUT* to given *Iridas .cube LUT* file.
read_LUT_SonySPI1D(path)	Read given *Sony .spi1d LUT* file.
write_LUT_SonySPI1D(LUT, path[, decimals])	Write given *LUT* to given *Sony .spi1d LUT* file.
read_LUT_SonySPI3D(path)	Read given *Sony .spi3d LUT* file.
write_LUT_SonySPI3D(LUT, path[, decimals])	Write given *LUT* to given *Sony .spi3d LUT* file.

colour.io.LUT_to_LUT

colour.io.**LUT_to_LUT**(*LUT*, *cls: colour.io.luts.lut.LUT1D | colour.io.luts.lut.LUT3x1D | colour.io.luts.lut.LUT3D*, *force_conversion: bool = False*, ***kwargs: Any*) → colour.io.luts.lut.AbstractLUT

Convert given *LUT* to given cls class instance.

Parameters

- **cls** (colour.io.luts.lut.LUT1D | colour.io.luts.lut.LUT3x1D | colour.io.luts.lut.LUT3D) – *LUT* class instance.
- **force_conversion** (bool) – Whether to force the conversion if destructive.
- **channel_weights** – Channel weights in case of a downcast from a LUT3x1D or LUT3D class instance.
- **interpolator** – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function.

- **size** – Expected table size in case of an upcast to or a downcast from a LUT3D class instance.
- **kwargs** (Any) –

Returns Converted *LUT* class instance.

Return type colour.LUT1D or colour.LUT3x1D or colour.LUT3D

> **Warning:** Some conversions are destructive and raise a ValueError exception by default.

Raises **ValueError** – If the conversion is destructive.

Parameters

- **cls** (colour.io.luts.lut.LUT1D | colour.io.luts.lut.LUT3x1D | colour.io.luts.lut.LUT3D) –
- **force_conversion** (bool) –
- **kwargs** (Any) –

Return type colour.io.luts.lut.AbstractLUT

Examples

```
>>> print(LUT_to_LUT(LUT1D(), LUT3D, force_conversion=True))
LUT3D - Unity 10 - Converted 1D to 3D
-------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
>>> print(LUT_to_LUT(LUT3x1D(), LUT1D, force_conversion=True))
LUT1D - Unity 10 - Converted 3x1D to 1D
---------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT_to_LUT(LUT3D(), LUT1D, force_conversion=True))
LUT1D - Unity 33 - Converted 3D to 1D
-------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
```

colour.io.read_LUT_Cinespace

colour.io.**read_LUT_Cinespace**(*path: str*) → *colour.io.luts.lut.LUT3x1D* | *colour.io.luts.lut.LUT3D* | *colour.io.luts.sequence.LUTSequence*

Read given *Cinespace .csp LUT* file.

Parameters **path** (str) – *LUT* path.

Returns LUT3x1D or LUT3D or LUTSequence class instance.

Return type colour.LUT3x1D or colour.LUT3D or colour.LUTSequence

References

[RisingSResearch]

Examples

Reading a 3x1D *Cinespace .csp LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "cinespace",
...     "ACES_Proxy_10_to_ACES.csp",
... )
>>> print(read_LUT_Cinespace(path))
LUT3x1D - ACES Proxy 10 to ACES
-------------------------------

Dimensions : 2
Domain     : [[ 0.   0.   0.]
              [ 1.   1.   1.]]
Size       : (32, 3)
```

Reading a 3D *Cinespace .csp LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "cinespace",
...     "Colour_Correct.csp",
... )
>>> print(read_LUT_Cinespace(path))
LUT3D - Generated by Foundry::LUT
---------------------------------

Dimensions : 3
Domain     : [[ 0.   0.   0.]
              [ 1.   1.   1.]]
Size       : (4, 4, 4, 3)
```

colour.io.write_LUT_Cinespace

colour.io.**write_LUT_Cinespace**(*LUT: colour.io.luts.lut.LUT3x1D | colour.io.luts.lut.LUT3D | colour.io.luts.sequence.LUTSequence*, *path: str*, *decimals: int = 7*) → bool

Write given *LUT* to given *Cinespace .csp LUT* file.

Parameters

- **LUT** (colour.io.luts.lut.LUT3x1D | colour.io.luts.lut.LUT3D | colour.io.luts.sequence.LUTSequence) – LUT1D, LUT3x1D or LUT3D or LUTSequence class instance to write at given path.
- **path** (str) – *LUT* path.
- **decimals** (int) – Formatting decimals.

Returns Definition success.

Return type bool

References

[RisingSResearch]

Examples

Writing a 3x1D *Cinespace .csp LUT*:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16, domain), 1 / 2.2),
...     "My LUT",
...     domain,
...     comments=["A first comment.", "A second comment."],
... )
>>> write_LUT_Cinespace(LUT, "My_LUT.cube")
```

Writing a 3D *Cinespace .csp LUT*:

```
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3D(
...     spow(LUT3D.linear_table(16, domain), 1 / 2.2),
...     "My LUT",
...     domain,
...     comments=["A first comment.", "A second comment."],
... )
>>> write_LUT_Cinespace(LUT, "My_LUT.cube")
```

colour.io.read_LUT_IridasCube

colour.io.**read_LUT_IridasCube**(*path: str*) → *colour.io.luts.lut.LUT3x1D* | *colour.io.luts.lut.LUT3D*

Read given *Iridas .cube LUT* file.

Parameters **path** (str) – *LUT* path.

Returns LUT3x1D or LUT3D class instance.

Return type LUT3x1D or LUT3D.

References

[AdobeSystems13c]

Examples

Reading a 3x1D *Iridas .cube LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "iridas_cube",
...     "ACES_Proxy_10_to_ACES.cube",
... )
>>> print(read_LUT_IridasCube(path))
LUT3x1D - ACES Proxy 10 to ACES
-------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (32, 3)
```

Reading a 3D *Iridas .cube LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "iridas_cube",
...     "Colour_Correct.cube",
... )
>>> print(read_LUT_IridasCube(path))
LUT3D - Generated by Foundry::LUT
---------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (4, 4, 4, 3)
```

Reading a 3D *Iridas .cube LUT* with comments:

```
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "iridas_cube",
...     "Demo.cube",
... )
>>> print(read_LUT_IridasCube(path))
LUT3x1D - Demo
--------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  2.  3.]]
Size       : (3, 3)
Comment 01 : Comments can go anywhere
```

colour.io.write_LUT_IridasCube

colour.io.**write_LUT_IridasCube**(*LUT: colour.io.luts.lut.LUT3x1D | colour.io.luts.lut.LUT3D | colour.io.luts.sequence.LUTSequence*, *path: str*, *decimals: int = 7*) → bool

Write given *LUT* to given *Iridas .cube LUT* file.

Parameters

- **LUT** (colour.io.luts.lut.LUT3x1D | colour.io.luts.lut.LUT3D | colour.io.luts.sequence.LUTSequence) – LUT3x1D, LUT3D or LUTSequence class instance to write at given path.
- **path** (str) – *LUT* path.
- **decimals** (int) – Formatting decimals.

Returns Definition success.

Return type bool

Warning:

- If a LUTSequence class instance is passed as LUT, the first *LUT* in the *LUT* sequence will be used.

References

[AdobeSystems13c]

Examples

Writing a 3x1D *Iridas .cube LUT*:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16, domain), 1 / 2.2),
...     "My LUT",
...     domain,
...     comments=["A first comment.", "A second comment."],
... )
>>> write_LUT_IridasCube(LUTxD, "My_LUT.cube")
```

Writing a 3D *Iridas .cube LUT*:

```
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3D(
...     spow(LUT3D.linear_table(16, domain), 1 / 2.2),
...     "My LUT",
...     np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]),
...     comments=["A first comment.", "A second comment."],
... )
>>> write_LUT_IridasCube(LUTxD, "My_LUT.cube")
```

colour.io.read_LUT_SonySPI1D

colour.io.**read_LUT_SonySPI1D**(*path: str*) → *colour.io.luts.lut.LUT1D* | *colour.io.luts.lut.LUT3x1D*

Read given *Sony .spi1d LUT* file.

Parameters **path** (str) – *LUT* path.

Returns LUT1D or LUT3x1D class instance.

Return type colour.LUT1D or colour.LUT3x1D

Examples

Reading a 1D *Sony .spi1d LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "sony_spi1d",
...     "eotf_sRGB_1D.spi1d",
... )
>>> print(read_LUT_SonySPI1D(path))
LUT1D - eotf sRGB 1D
--------------------

Dimensions : 1
Domain     : [-0.1  1.5]
Size       : (16,)
Comment 01 : Generated by "Colour 0.3.11".
Comment 02 : "colour.models.eotf_sRGB".
```

Reading a 3x1D *Sony .spi1d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "sony_spi1d",
...     "eotf_sRGB_3x1D.spi1d",
... )
>>> print(read_LUT_SonySPI1D(path))
LUT3x1D - eotf sRGB 3x1D
------------------------

Dimensions : 2
Domain     : [[-0.1 -0.1 -0.1]
              [ 1.5  1.5  1.5]]
Size       : (16, 3)
Comment 01 : Generated by "Colour 0.3.11".
Comment 02 : "colour.models.eotf_sRGB".
```

colour.io.write_LUT_SonySPI1D

colour.io.**write_LUT_SonySPI1D**(*LUT: colour.io.luts.lut.LUT1D | colour.io.luts.lut.LUT3x1D | colour.io.luts.sequence.LUTSequence, path: str, decimals: int = 7*) → bool

Write given *LUT* to given *Sony .spi1d LUT* file.

Parameters

- **LUT** (colour.io.luts.lut.LUT1D | colour.io.luts.lut.LUT3x1D | colour.io.luts.sequence.LUTSequence) – LUT1D, LUT3x1D or LUTSequence class instance to write at given path.
- **path** (str) – *LUT* path.
- **decimals** (int) – Formatting decimals.

Returns Definition success.

Return type bool

Warning:

- If a LUTSequence class instance is passed as LUT, the first *LUT* in the *LUT* sequence will be used.

Examples

Writing a 1D *Sony .spi1d LUT*:

```
>>> from colour.algebra import spow
>>> domain = np.array([-0.1, 1.5])
>>> LUT = LUT1D(
...     spow(LUT1D.linear_table(16), 1 / 2.2),
...     "My LUT",
...     domain,
...     comments=["A first comment.", "A second comment."],
```

(continues on next page)

(continued from previous page)

```
... )
>>> write_LUT_SonySPI1D(LUT, "My_LUT.cube")
```

Writing a 3x1D *Sony .spi1d LUT*:

```
>>> domain = np.array([[-0.1, -0.1, -0.1], [1.5, 1.5, 1.5]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16), 1 / 2.2),
...     "My LUT",
...     domain,
...     comments=["A first comment.", "A second comment."],
... )
>>> write_LUT_SonySPI1D(LUT, "My_LUT.cube")
```

colour.io.read_LUT_SonySPI3D

colour.io.**read_LUT_SonySPI3D**(*path: str*) → *colour.io.luts.lut.LUT3D*

Read given *Sony .spi3d LUT* file.

Parameters **path** (str) – *LUT* path.

Returns LUT3D class instance.

Return type colour.LUT3D

Examples

Reading an ordered and an unordered 3D *Sony .spi3d LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "sony_spi3d",
...     "Colour_Correct.spi3d",
... )
>>> print(read_LUT_SonySPI3D(path))
LUT3D - Colour Correct
----------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (4, 4, 4, 3)
Comment 01 : Adapted from a LUT generated by Foundry::LUT.
>>> path = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "sony_spi3d",
...     "Colour_Correct_Unordered.spi3d",
... )
>>> print(read_LUT_SonySPI3D(path))
LUT3D - Colour Correct Unordered
```

(continues on next page)

(continued from previous page)

```
-------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (4, 4, 4, 3)
Comment 01 : Adapted from a LUT generated by Foundry::LUT.
```

colour.io.write_LUT_SonySPI3D

colour.io.**write_LUT_SonySPI3D**(*LUT: colour.io.luts.lut.LUT3D | colour.io.luts.sequence.LUTSequence, path: str, decimals: int = 7*) → bool

Write given *LUT* to given *Sony .spi3d LUT* file.

Parameters

- **LUT** (colour.io.luts.lut.LUT3D | colour.io.luts.sequence.LUTSequence) – LUT3D or LUTSequence class instance to write at given path.
- **path** (str) – *LUT* path.
- **decimals** (int) – Formatting decimals.

Returns Definition success.

Return type bool

> **Warning:**
>
> - If a LUTSequence class instance is passed as LUT, the first *LUT* in the *LUT* sequence will be used.

Examples

Writing a 3D *Sony .spi3d LUT*:

```
>>> LUT = LUT3D(
...     LUT3D.linear_table(16) ** (1 / 2.2),
...     "My LUT",
...     np.array([[0, 0, 0], [1, 1, 1]]),
...     comments=["A first comment.", "A second comment."],
... )
>>> write_LUT_SonySPI3D(LUT, "My_LUT.cube")
```

CSV Tabular Data

colour

read_sds_from_csv_file(path, **kwargs)	Read the spectral data from given *CSV* file and returns its content as a *dict* of colour.SpectralDistribution class instances.
read_spectral_data_from_csv_file(path, **kwargs)	Read the spectral data from given *CSV* file in the following form.
write_sds_to_csv_file(sds, path)	Write the given spectral distributions to given *CSV* file.

colour.read_sds_from_csv_file

colour.**read_sds_from_csv_file**(*path: str*, ***kwargs: Any*) → Dict[str, *colour.colorimetry.spectrum.SpectralDistribution*]

Read the spectral data from given *CSV* file and returns its content as a *dict* of `colour.SpectralDistribution` class instances.

Parameters

- **path** (`str`) – *CSV* file path.
- **kwargs** (`Any`) – Keywords arguments passed to `numpy.recfromcsv()` definition.

Returns *dict* of `colour.SpectralDistribution` class instances.

Return type `dict`

Examples

```
>>> from colour.utilities import numpy_print_options
>>> import os
>>> csv_file = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "colorchecker_n_ohta.csv",
... )
>>> sds = read_sds_from_csv_file(csv_file)
>>> print(tuple(sds.keys()))
('1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16
↪', '17', '18', '19', '20', '21', '22', '23', '24')
>>> with numpy_print_options(suppress=True):
...     sds["1"]
...
SpectralDistribution([[ 380.   ,    0.048],
                      [ 385.   ,    0.051],
                      [ 390.   ,    0.055],
                      [ 395.   ,    0.06 ],
                      [ 400.   ,    0.065],
                      [ 405.   ,    0.068],
                      [ 410.   ,    0.068],
                      [ 415.   ,    0.067],
                      [ 420.   ,    0.064],
                      [ 425.   ,    0.062],
                      [ 430.   ,    0.059],
                      [ 435.   ,    0.057],
                      [ 440.   ,    0.055],
                      [ 445.   ,    0.054],
                      [ 450.   ,    0.053],
                      [ 455.   ,    0.053],
                      [ 460.   ,    0.052],
                      [ 465.   ,    0.052],
                      [ 470.   ,    0.052],
                      [ 475.   ,    0.053],
                      [ 480.   ,    0.054],
                      [ 485.   ,    0.055],
                      [ 490.   ,    0.057],
                      [ 495.   ,    0.059],
```

(continues on next page)

(continued from previous page)

```
                    [ 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],
```

(continues on next page)

(continued from previous page)

```
                       [ 780.    ,     0.421]],
                      SpragueInterpolator,
                      {},
                      Extrapolator,
                      {'method': 'Constant', 'left': None, 'right': None})
```

colour.read_spectral_data_from_csv_file

colour.**read_spectral_data_from_csv_file**(*path: str*, ***kwargs: Any*) → Dict[str, <sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ceba22f90>]

Read 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 *dict* as follows:

```
{
    'wavelength': ndarray,
    'field 1': ndarray,
    'field 2': ndarray,
    ...,
    'field n': ndarray
}
```

Parameters

- **path** (str) – *CSV* file path.
- **kwargs** (Any) – Keywords arguments passed to numpy.recfromcsv() definition.

Returns *CSV* file content.

Return type dict

Notes

- A *CSV* spectral data file should define at least define two fields: one for the wavelengths and one for the associated values of one spectral distribution.

Examples

```
>>> import os
>>> from pprint import pprint
>>> csv_file = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "colorchecker_n_ohta.csv",
... )
```

(continues on next page)

(continued from previous page)

```
>>> sds_data = read_spectral_data_from_csv_file(csv_file)
>>> pprint(list(sds_data.keys()))
['wavelength',
 '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_sds_to_csv_file

colour.**write_sds_to_csv_file**(*sds: Dict[str, colour.colorimetry.spectrum.SpectralDistribution], path: str*) → bool

Write the given spectral distributions to given *CSV* file.

Parameters

- **sds** (Dict[str, colour.colorimetry.spectrum.SpectralDistribution]) – Spectral distributions to write to given *CSV* file.
- **path** (str) – *CSV* file path.

Returns Definition success.

Return type bool

Raises ValueError – If the given spectral distributions have different shapes.

IES TM-27-14 Data

colour

SpectralDistribution_IESTM2714([path, ...])	Define a *IES TM-27-14* spectral distribution.

colour.SpectralDistribution_IESTM2714

class colour.**SpectralDistribution_IESTM2714**(*path: str | None = None, header: colour.io.tm2714.Header_IESTM2714 | None = None, spectral_quantity: Optional[Literal['absorptance', 'exitance', 'flux', 'intensity', 'irradiance', 'radiance', 'reflectance', 'relative', 'transmittance', 'R-Factor', 'T-Factor', 'other']] = None, reflection_geometry: Optional[Literal['di:8', 'de:8', '8:di', '8:de', 'd:d', 'd:0', '45a:0', '45c:0', '0:45a', '45x:0', '0:45x', 'other']] = None, transmission_geometry: Optional[Literal['0:0', 'di:0', 'de:0', '0:di', '0:de', 'd:d', 'other']] = None, bandwidth_FWHM: float | None = None, bandwidth_corrected: bool | None = None, **kwargs*)

Bases: colour.colorimetry.spectrum.SpectralDistribution

Define a *IES TM-27-14* spectral distribution.

This class can read and write *IES TM-27-14* spectral data *XML* files.

Parameters

- **path** (str | None) – Spectral data *XML* file path.
- **header** (Header_IESTM2714 | None) – *IES TM-27-14* spectral distribution header.
- **spectral_quantity** (Literal['absorptance', 'exitance', 'flux', 'intensity', 'irradiance', 'radiance', 'reflectance', 'relative', 'transmittance', 'R-Factor', 'T-Factor', 'other'] | None) – Quantity of measurement for each element of the spectral data.
- **reflection_geometry** (Literal['di:8', 'de:8', '8:di', '8:de', 'd:d', 'd:0', '45a:0', '45c:0', '0:45a', '45x:0', '0:45x', 'other'] | None) – Spectral reflectance factors geometric conditions.
- **transmission_geometry** (Literal['0:0', 'di:0', 'de:0', '0:di', '0:de', 'd:d', 'other'] | None) – Spectral transmittance factors geometric conditions.
- **bandwidth_FWHM** (float | None) – Spectroradiometer full-width half-maximum bandwidth in nanometers.
- **bandwidth_corrected** (bool | None) – Specifies if bandwidth correction has been applied to the measured data.
- **data** – Data to be stored in the spectral distribution.
- **domain** – Values to initialise the colour.SpectralDistribution.wavelength property with. If both data and domain arguments are defined, the latter will be used to initialise the colour.SpectralDistribution.wavelength property.
- **extrapolator** – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function.
- **interpolator** – Interpolator class type to use as interpolating function.

- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function.
- **name** – Spectral distribution name.
- **display_name** – Spectral distribution name for figures, default to `colour.SpectralDistribution.name` property value.

Return type None

Notes

Reflection Geometry

- di:8: Diffuse / eight-degree, specular component included.
- de:8: Diffuse / eight-degree, specular component excluded.
- 8:di: Eight-degree / diffuse, specular component included.
- 8:de: Eight-degree / diffuse, specular component excluded.
- d:d: Diffuse / diffuse.
- d:0: Alternative diffuse.
- 45a:0: Forty-five degree annular / normal.
- 45c:0: Forty-five degree circumferential / normal.
- 0:45a: Normal / forty-five degree annular.
- 45x:0: Forty-five degree directional / normal.
- 0:45x: Normal / forty-five degree directional.
- other: User-specified in comments.

Transmission Geometry

- 0:0: Normal / normal.
- di:0: Diffuse / normal, regular component included.
- de:0: Diffuse / normal, regular component excluded.
- 0:di: Normal / diffuse, regular component included.
- 0:de: Normal / diffuse, regular component excluded.
- d:d: Diffuse / diffuse.
- other: User-specified in comments.

Attributes

- `mapping`
- `path`
- `header`
- `spectral_quantity`
- `reflection_geometry`
- `transmission_geometry`
- `bandwidth_FWHM`
- `bandwidth_corrected`

Methods

- `__init__()`
- `__str__()`
- `__repr__()`
- `read()`
- `write()`

References

[IESCCommitteeTM2714WGroup14]

Examples

```
>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), "tests", "resources")
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, "Fluorescent.spdx")
... ).read()
>>> sd.name
'Unknown - N/A - Rare earth fluorescent lamp'
>>> sd.header.comments
'Ambient temperature 25 degrees C.'
>>> sd[501.7]
0.0950000...
```

__init__(*path: str | None = None, header: colour.io.tm2714.Header_IESTM2714 | None = None, spectral_quantity: Optional[Literal['absorptance', 'exitance', 'flux', 'intensity', 'irradiance', 'radiance', 'reflectance', 'relative', 'transmittance', 'R-Factor', 'T-Factor', 'other']] = None, reflection_geometry: Optional[Literal['di:8', 'de:8', '8:di', '8:de', 'd:d', 'd:0', '45a:0', '45c:0', '0:45a', '45x:0', '0:45x', 'other']] = None, transmission_geometry: Optional[Literal['0:0', 'di:0', 'de:0', '0:di', '0:de', 'd:d', 'other']] = None, bandwidth_FWHM: float | None = None, bandwidth_corrected: bool | None = None, **kwargs*) → None

Parameters

- **path** (`str | None`) –
- **header** (`colour.io.tm2714.Header_IESTM2714 | None`) –
- **spectral_quantity** (`Optional[Literal['absorptance', 'exitance', 'flux', 'intensity', 'irradiance', 'radiance', 'reflectance', 'relative', 'transmittance', 'R-Factor', 'T-Factor', 'other']]`) –
- **reflection_geometry** (`Optional[Literal['di:8', 'de:8', '8:di', '8:de', 'd:d', 'd:0', '45a:0', '45c:0', '0:45a', '45x:0', '0:45x', 'other']]`) –
- **transmission_geometry** (`Optional[Literal['0:0', 'di:0', 'de:0', '0:di', '0:de', 'd:d', 'other']]`) –
- **bandwidth_FWHM** (`float | None`) –
- **bandwidth_corrected** (`bool | None`) –

Return type None

property mapping: colour.utilities.data_structures.Structure

Getter property for the mapping structure.

Returns Mapping structure.

Return type `colour.utilities.Structure`

property path: str | None

Getter and setter property for the path.

Parameters **value** – Value to set the path with.

Returns Path.

Return type `None` or `str`

property header: colour.io.tm2714.Header_IESTM2714

Getter and setter property for the header.

Parameters **value** – Value to set the header with.

Returns Header.

Return type `colour.io.tm2714.Header_IESTM2714`

property spectral_quantity: Optional[Literal['absorptance', 'exitance', 'flux', 'intensity', 'irradiance', 'radiance', 'reflectance', 'relative', 'transmittance', 'R-Factor', 'T-Factor', 'other']]

Getter and setter property for the spectral quantity.

Parameters **value** – Value to set the spectral quantity with.

Returns Spectral quantity.

Return type `None` or `str`

property reflection_geometry: Optional[Literal['di:8', 'de:8', '8:di', '8:de', 'd:d', 'd:0', '45a:0', '45c:0', '0:45a', '45x:0', '0:45x', 'other']]

Getter and setter property for the reflection geometry.

Parameters **value** – Value to set the reflection geometry with.

Returns Reflection geometry.

Return type `None` or `str`

property transmission_geometry: Optional[Literal['0:0', 'di:0', 'de:0', '0:di', '0:de', 'd:d', 'other']]

Getter and setter property for the transmission geometry.

Parameters **value** – Value to set the transmission geometry with.

Returns Transmission geometry.

Return type `None` or `str`

property bandwidth_FWHM: float | None

Getter and setter property for the full-width half-maximum bandwidth.

Parameters **value** – Value to set the full-width half-maximum bandwidth with.

Returns Full-width half-maximum bandwidth.

Return type `None` or `float`

property bandwidth_corrected: bool | None

Getter and setter property for whether bandwidth correction has been applied to the measured data.

Parameters **value** – Whether bandwidth correction has been applied to the measured data.

Returns Whether bandwidth correction has been applied to the measured data.

Return type None or bool

__str__() → str

Return a formatted string representation of the *IES TM-27-14* spectral distribution.

Returns Formatted string representation.

Return type str

Examples

```
>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), "tests", "resources")
>>> print(
...     SpectralDistribution_IESTM2714(
...         join(directory, "Fluorescent.spdx")
...     ).read()
... )
...
IES TM-27-14 Spectral Distribution
==================================

Path                   : ...
Spectral Quantity      : relative
Reflection Geometry    : other
Transmission Geometry : other
Bandwidth (FWHM)       : 2.0
Bandwidth Corrected    : True

Header
------

Manufacturer            : Unknown
Catalog Number          : N/A
Description             : Rare earth fluorescent lamp
Document Creator        : byHeart Consultants
Unique Identifier       : C3567553-C75B-4354-961E-35CEB9FEB42C
Measurement Equipment   : None
Laboratory              : N/A
Report Number           : N/A
Report Date             : N/A
Document Creation Date : 2014-06-23
Comments                : Ambient temperature 25 degrees C.

Spectral Data
-------------

[[  4.00000000e+02   3.40000000e-02]
 [  4.03100000e+02   3.70000000e-02]
 [  4.05500000e+02   6.90000000e-02]
 [  4.07500000e+02   3.70000000e-02]
 [  4.20600000e+02   4.20000000e-02]
 [  4.31000000e+02   4.90000000e-02]
 [  4.33700000e+02   6.00000000e-02]
```

(continues on next page)

(continued from previous page)

```
[  4.37000000e+02   3.57000000e-01]
[  4.38900000e+02   6.00000000e-02]
[  4.60000000e+02   6.80000000e-02]
[  4.77000000e+02   7.50000000e-02]
[  4.81000000e+02   8.50000000e-02]
[  4.88200000e+02   2.04000000e-01]
[  4.92600000e+02   1.66000000e-01]
[  5.01700000e+02   9.50000000e-02]
[  5.07600000e+02   7.80000000e-02]
[  5.17600000e+02   7.10000000e-02]
[  5.29900000e+02   7.60000000e-02]
[  5.35400000e+02   9.90000000e-02]
[  5.39900000e+02   4.23000000e-01]
[  5.43200000e+02   8.02000000e-01]
[  5.44400000e+02   7.13000000e-01]
[  5.47200000e+02   9.99000000e-01]
[  5.48700000e+02   5.73000000e-01]
[  5.50200000e+02   3.40000000e-01]
[  5.53800000e+02   2.08000000e-01]
[  5.57300000e+02   1.39000000e-01]
[  5.63700000e+02   1.29000000e-01]
[  5.74800000e+02   1.31000000e-01]
[  5.78000000e+02   1.98000000e-01]
[  5.79200000e+02   1.90000000e-01]
[  5.80400000e+02   2.05000000e-01]
[  5.84800000e+02   2.44000000e-01]
[  5.85900000e+02   2.36000000e-01]
[  5.87500000e+02   2.56000000e-01]
[  5.90300000e+02   1.80000000e-01]
[  5.93500000e+02   2.18000000e-01]
[  5.95500000e+02   1.59000000e-01]
[  5.97000000e+02   1.47000000e-01]
[  5.99400000e+02   1.70000000e-01]
[  6.02200000e+02   1.34000000e-01]
[  6.04600000e+02   1.21000000e-01]
[  6.07400000e+02   1.40000000e-01]
[  6.09400000e+02   2.29000000e-01]
[  6.10200000e+02   4.65000000e-01]
[  6.12000000e+02   9.52000000e-01]
[  6.14600000e+02   4.77000000e-01]
[  6.16900000e+02   2.08000000e-01]
[  6.18500000e+02   1.35000000e-01]
[  6.22100000e+02   1.50000000e-01]
[  6.25600000e+02   1.55000000e-01]
[  6.28400000e+02   1.34000000e-01]
[  6.31200000e+02   1.68000000e-01]
[  6.33200000e+02   8.70000000e-02]
[  6.35600000e+02   6.80000000e-02]
[  6.42700000e+02   5.80000000e-02]
[  6.48700000e+02   5.80000000e-02]
[  6.50700000e+02   7.40000000e-02]
[  6.52600000e+02   6.30000000e-02]
[  6.56200000e+02   5.30000000e-02]
[  6.57000000e+02   5.60000000e-02]
[  6.60600000e+02   4.90000000e-02]
[  6.62600000e+02   5.90000000e-02]
```

(continues on next page)

(continued from previous page)

```
[  6.64200000e+02   4.80000000e-02]
[  6.86000000e+02   4.10000000e-02]
[  6.87600000e+02   4.80000000e-02]
[  6.89200000e+02   3.90000000e-02]
[  6.92400000e+02   3.80000000e-02]
[  6.93500000e+02   4.40000000e-02]
[  6.95500000e+02   3.40000000e-02]
[  7.02300000e+02   3.60000000e-02]
[  7.06700000e+02   4.20000000e-02]
[  7.07100000e+02   6.10000000e-02]
[  7.10200000e+02   6.10000000e-02]
[  7.11000000e+02   4.10000000e-02]
[  7.12200000e+02   5.20000000e-02]
[  7.14200000e+02   3.30000000e-02]
[  7.48400000e+02   3.40000000e-02]
[  7.57900000e+02   3.10000000e-02]
[  7.60700000e+02   3.90000000e-02]
[  7.63900000e+02   2.90000000e-02]
[  8.08800000e+02   2.90000000e-02]
[  8.10700000e+02   3.90000000e-02]
[  8.12700000e+02   3.00000000e-02]
[  8.50100000e+02   3.00000000e-02]]
```

__repr__() → str

Return an evaluable string representation of the *IES TM-27-14* spectral distribution.

Returns Evaluable string representation.

Return type str

Examples

```
>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), "tests", "resources")
>>> SpectralDistribution_IESTM2714(
...     join(directory, "Fluorescent.spdx")
... ).read()
...
SpectralDistribution_IESTM2714('...',
                               Header_IESTM2714('Unknown',
                                                'N/A',
                                                'Rare earth ...',
                                                'byHeart Consultants',
                                                'C3567553-C75B-...',
                                                None,
                                                'N/A',
                                                'N/A',
                                                'N/A',
                                                '2014-06-23',
                                                'Ambient ...'),
                               'relative',
                               'other',
                               'other',
                               2.0,
                               True,
                               [[  4.00000000e+02,   3.40000000e-02],
```

(continues on next page)

(continued from previous page)

```
                                   [  4.03100000e+02,   3.70000000e-02],
                                   [  4.05500000e+02,   6.90000000e-02],
                                   [  4.07500000e+02,   3.70000000e-02],
                                   [  4.20600000e+02,   4.20000000e-02],
                                   [  4.31000000e+02,   4.90000000e-02],
                                   [  4.33700000e+02,   6.00000000e-02],
                                   [  4.37000000e+02,   3.57000000e-01],
                                   [  4.38900000e+02,   6.00000000e-02],
                                   [  4.60000000e+02,   6.80000000e-02],
                                   [  4.77000000e+02,   7.50000000e-02],
                                   [  4.81000000e+02,   8.50000000e-02],
                                   [  4.88200000e+02,   2.04000000e-01],
                                   [  4.92600000e+02,   1.66000000e-01],
                                   [  5.01700000e+02,   9.50000000e-02],
                                   [  5.07600000e+02,   7.80000000e-02],
                                   [  5.17600000e+02,   7.10000000e-02],
                                   [  5.29900000e+02,   7.60000000e-02],
                                   [  5.35400000e+02,   9.90000000e-02],
                                   [  5.39900000e+02,   4.23000000e-01],
                                   [  5.43200000e+02,   8.02000000e-01],
                                   [  5.44400000e+02,   7.13000000e-01],
                                   [  5.47200000e+02,   9.99000000e-01],
                                   [  5.48700000e+02,   5.73000000e-01],
                                   [  5.50200000e+02,   3.40000000e-01],
                                   [  5.53800000e+02,   2.08000000e-01],
                                   [  5.57300000e+02,   1.39000000e-01],
                                   [  5.63700000e+02,   1.29000000e-01],
                                   [  5.74800000e+02,   1.31000000e-01],
                                   [  5.78000000e+02,   1.98000000e-01],
                                   [  5.79200000e+02,   1.90000000e-01],
                                   [  5.80400000e+02,   2.05000000e-01],
                                   [  5.84800000e+02,   2.44000000e-01],
                                   [  5.85900000e+02,   2.36000000e-01],
                                   [  5.87500000e+02,   2.56000000e-01],
                                   [  5.90300000e+02,   1.80000000e-01],
                                   [  5.93500000e+02,   2.18000000e-01],
                                   [  5.95500000e+02,   1.59000000e-01],
                                   [  5.97000000e+02,   1.47000000e-01],
                                   [  5.99400000e+02,   1.70000000e-01],
                                   [  6.02200000e+02,   1.34000000e-01],
                                   [  6.04600000e+02,   1.21000000e-01],
                                   [  6.07400000e+02,   1.40000000e-01],
                                   [  6.09400000e+02,   2.29000000e-01],
                                   [  6.10200000e+02,   4.65000000e-01],
                                   [  6.12000000e+02,   9.52000000e-01],
                                   [  6.14600000e+02,   4.77000000e-01],
                                   [  6.16900000e+02,   2.08000000e-01],
                                   [  6.18500000e+02,   1.35000000e-01],
                                   [  6.22100000e+02,   1.50000000e-01],
                                   [  6.25600000e+02,   1.55000000e-01],
                                   [  6.28400000e+02,   1.34000000e-01],
                                   [  6.31200000e+02,   1.68000000e-01],
                                   [  6.33200000e+02,   8.70000000e-02],
                                   [  6.35600000e+02,   6.80000000e-02],
                                   [  6.42700000e+02,   5.80000000e-02],
                                   [  6.48700000e+02,   5.80000000e-02],
```

(continues on next page)

(continued from previous page)

```
                                  [  6.50700000e+02,   7.40000000e-02],
                                  [  6.52600000e+02,   6.30000000e-02],
                                  [  6.56200000e+02,   5.30000000e-02],
                                  [  6.57000000e+02,   5.60000000e-02],
                                  [  6.60600000e+02,   4.90000000e-02],
                                  [  6.62600000e+02,   5.90000000e-02],
                                  [  6.64200000e+02,   4.80000000e-02],
                                  [  6.86000000e+02,   4.10000000e-02],
                                  [  6.87600000e+02,   4.80000000e-02],
                                  [  6.89200000e+02,   3.90000000e-02],
                                  [  6.92400000e+02,   3.80000000e-02],
                                  [  6.93500000e+02,   4.40000000e-02],
                                  [  6.95500000e+02,   3.40000000e-02],
                                  [  7.02300000e+02,   3.60000000e-02],
                                  [  7.06700000e+02,   4.20000000e-02],
                                  [  7.07100000e+02,   6.10000000e-02],
                                  [  7.10200000e+02,   6.10000000e-02],
                                  [  7.11000000e+02,   4.10000000e-02],
                                  [  7.12200000e+02,   5.20000000e-02],
                                  [  7.14200000e+02,   3.30000000e-02],
                                  [  7.48400000e+02,   3.40000000e-02],
                                  [  7.57900000e+02,   3.10000000e-02],
                                  [  7.60700000e+02,   3.90000000e-02],
                                  [  7.63900000e+02,   2.90000000e-02],
                                  [  8.08800000e+02,   2.90000000e-02],
                                  [  8.10700000e+02,   3.90000000e-02],
                                  [  8.12700000e+02,   3.00000000e-02],
                                  [  8.50100000e+02,   3.00000000e-02]],
                                 CubicSplineInterpolator,
                                 {},
                                 Extrapolator,
                                 {...})
```

read() → *colour.io.tm2714.SpectralDistribution_IESTM2714*

Read and parses the spectral data *XML* file path.

Returns *IES TM-27-14* spectral distribution.

Return type `colour.SpectralDistribution_IESTM2714`

Raises `ValueError` – If the *IES TM-27-14* spectral distribution path is undefined.

Examples

```
>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), "tests", "resources")
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, "Fluorescent.spdx")
... ).read()
>>> sd.name
'Unknown - N/A - Rare earth fluorescent lamp'
>>> sd.header.comments
'Ambient temperature 25 degrees C.'
>>> sd[400]
0.0340000...
```

write() → bool

Write the spectral distribution spectral data to *XML* file path.

Returns Definition success.

Return type bool

Examples

```
>>> from os.path import dirname, join
>>> from shutil import rmtree
>>> from tempfile import mkdtemp
>>> directory = join(dirname(__file__), "tests", "resources")
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, "Fluorescent.spdx")
... ).read()
>>> temporary_directory = mkdtemp()
>>> sd.path = join(temporary_directory, "Fluorescent.spdx")
>>> sd.write()
True
>>> rmtree(temporary_directory)
```

UPRTek and Sekonic Spectral Data

colour

SpectralDistribution_UPRTek(path, **kwargs)	Implement support to read and write *IES TM-27-14* spectral data XML file from a *UPRTek Pseudo-XLS* file.
SpectralDistribution_Sekonic(path, **kwargs)	Implement support to read and write *IES TM-27-14* spectral data XML file from a *Sekonic CSV* file.

colour.SpectralDistribution_UPRTek

class colour.**SpectralDistribution_UPRTek**(*path: str*, ***kwargs: Any*)

Bases: colour.io.tm2714.SpectralDistribution_IESTM2714

Implement support to read and write *IES TM-27-14* spectral data XML file from a *UPRTek Pseudo-XLS* file.

Parameters

- **path** (str) – Path for *UPRTek Pseudo-XLS* file.
- **kwargs** (Any) –

Return type None

Attributes

- metadata

Methods

- __init__()
- __str__()
- read()

Examples

```
>>> from os.path import dirname, join
>>> from colour import SpectralShape
>>> directory = join(dirname(__file__), "tests", "resources")
>>> sd = SpectralDistribution_UPRTek(
...     join(directory, "ESPD2021_0104_231446.xls")
... )
>>> print(sd.read().align(SpectralShape(380, 780, 10)))
...
UPRTek
======

Path                  : ...
Spectral Quantity     : irradiance
Reflection Geometry   : None
Transmission Geometry : None
Bandwidth (FWHM)      : None
Bandwidth Corrected   : None

Header
------

Manufacturer           : UPRTek
Catalog Number         : None
Description            : None
Document Creator       : None
Unique Identifier      : None
Measurement Equipment  : CV600
Laboratory             : None
Report Number          : None
Report Date            : 2021/01/04_23:14:46
Document Creation Date : None
Comments               : {...}

Spectral Data
-------------

[[  3.80000000e+02   3.02670000e-02]
 [  3.90000000e+02   3.52230000e-02]
 [  4.00000000e+02   1.93250000e-02]
 [  4.10000000e+02   2.94260000e-02]
 [  4.20000000e+02   8.76780000e-02]
 [  4.30000000e+02   6.32578000e-01]
 [  4.40000000e+02   3.62565900e+00]
```

(continues on next page)

(continued from previous page)

```
[   4.50000000e+02   1.42069180e+01]
[   4.60000000e+02   1.70112970e+01]
[   4.70000000e+02   1.19673130e+01]
[   4.80000000e+02   8.42736200e+00]
[   4.90000000e+02   7.97729800e+00]
[   5.00000000e+02   8.71903600e+00]
[   5.10000000e+02   9.55321500e+00]
[   5.20000000e+02   9.90610500e+00]
[   5.30000000e+02   9.91394400e+00]
[   5.40000000e+02   9.74738000e+00]
[   5.50000000e+02   9.53404900e+00]
[   5.60000000e+02   9.27392200e+00]
[   5.70000000e+02   9.02323400e+00]
[   5.80000000e+02   8.91788800e+00]
[   5.90000000e+02   9.11454600e+00]
[   6.00000000e+02   9.55787100e+00]
[   6.10000000e+02   1.00600760e+01]
[   6.20000000e+02   1.04846200e+01]
[   6.30000000e+02   1.05679540e+01]
[   6.40000000e+02   1.04359870e+01]
[   6.50000000e+02   9.82122300e+00]
[   6.60000000e+02   8.77578300e+00]
[   6.70000000e+02   7.56471800e+00]
[   6.80000000e+02   6.29808600e+00]
[   6.90000000e+02   5.15623400e+00]
[   7.00000000e+02   4.05390600e+00]
[   7.10000000e+02   3.06638600e+00]
[   7.20000000e+02   2.19250000e+00]
[   7.30000000e+02   1.53922800e+00]
[   7.40000000e+02   1.14938200e+00]
[   7.50000000e+02   9.05095000e-01]
[   7.60000000e+02   6.90947000e-01]
[   7.70000000e+02   5.08426000e-01]
[   7.80000000e+02   4.11766000e-01]]
```

```
>>> sd.header.comments
'{"Model Name": "CV600", "Serial Number": "19J00789", "Time": "2021/01/04_23:14:46",
↪"Memo": [], "LUX": 695.154907, "fc": 64.605476, "CCT": 5198.0, "Duv": -0.00062, "I-
↪Time": 12000.0, "X": 682.470886, "Y": 695.154907, "Z": 631.635071, "x": 0.339663,
↪"y": 0.345975, "u\'": 0.209915, "v\'": 0.481087, "LambdaP": 456.0, "LambdaPValue":␣
↪18.404581, "CRI": 92.956993, "R1": 91.651062, "R2": 93.014732, "R3": 97.032013, "R4
↪": 93.513229, "R5": 92.48259, "R6": 91.48687, "R7": 93.016129, "R8": 91.459312, "R9
↪": 77.613075, "R10": 86.981613, "R11": 94.841324, "R12": 74.139542, "R13": 91.
↪073837, "R14": 97.064323, "R15": 88.615669, "TLCI": 97.495056, "TLMF-A": 1.270032,
↪"SSI-A": 44.881924, "Rf": 87.234917, "Rg": 98.510712, "IRR": 2.607891}'
>>> sd.metadata.keys()
dict_keys(['Model Name', 'Serial Number', 'Time', 'Memo', 'LUX', 'fc', 'CCT', 'Duv',
↪'I-Time', 'X', 'Y', 'Z', 'x', 'y', "u'", "v'", 'LambdaP', 'LambdaPValue', 'CRI',
↪'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11', 'R12', 'R13',
↪'R14', 'R15', 'TLCI', 'TLMF-A', 'SSI-A', 'Rf', 'Rg', 'IRR'])
>>> sd.write(join(directory, "ESPD2021_0104_231446.spdx"))
...
```

__init__(*path: str*, ***kwargs: Any*) → None

Parameters

- **path** (str) –
- **kwargs** (Any) –

Return type None

property metadata: dict

Getter property for the metadata.

Returns Metadata.

Return type dict

__str__() → str

Return a formatted string representation of the *UPRTek* spectral distribution.

Returns Formatted string representation.

Return type str

Examples

```
>>> from os.path import dirname, join
>>> from colour import SpectralShape
>>> directory = join(dirname(__file__), "tests", "resources")
>>> sd = SpectralDistribution_UPRTek(
...     join(directory, "ESPD2021_0104_231446.xls")
... )
>>> print(sd.read().align(SpectralShape(380, 780, 10)))
...
UPRTek
======

Path                  : ...
Spectral Quantity     : irradiance
Reflection Geometry   : None
Transmission Geometry : None
Bandwidth (FWHM)      : None
Bandwidth Corrected   : None

Header
------

Manufacturer           : UPRTek
Catalog Number         : None
Description            : None
Document Creator       : None
Unique Identifier      : None
Measurement Equipment  : CV600
Laboratory             : None
Report Number          : None
Report Date            : 2021/01/04_23:14:46
Document Creation Date : None
Comments               : {...}

Spectral Data
-------------

[[  3.80000000e+02   3.02670000e-02]
 [  3.90000000e+02   3.52230000e-02]
```

(continues on next page)

(continued from previous page)

```
[   4.00000000e+02   1.93250000e-02]
[   4.10000000e+02   2.94260000e-02]
[   4.20000000e+02   8.76780000e-02]
[   4.30000000e+02   6.32578000e-01]
[   4.40000000e+02   3.62565900e+00]
[   4.50000000e+02   1.42069180e+01]
[   4.60000000e+02   1.70112970e+01]
[   4.70000000e+02   1.19673130e+01]
[   4.80000000e+02   8.42736200e+00]
[   4.90000000e+02   7.97729800e+00]
[   5.00000000e+02   8.71903600e+00]
[   5.10000000e+02   9.55321500e+00]
[   5.20000000e+02   9.90610500e+00]
[   5.30000000e+02   9.91394400e+00]
[   5.40000000e+02   9.74738000e+00]
[   5.50000000e+02   9.53404900e+00]
[   5.60000000e+02   9.27392200e+00]
[   5.70000000e+02   9.02323400e+00]
[   5.80000000e+02   8.91788800e+00]
[   5.90000000e+02   9.11454600e+00]
[   6.00000000e+02   9.55787100e+00]
[   6.10000000e+02   1.00600760e+01]
[   6.20000000e+02   1.04846200e+01]
[   6.30000000e+02   1.05679540e+01]
[   6.40000000e+02   1.04359870e+01]
[   6.50000000e+02   9.82122300e+00]
[   6.60000000e+02   8.77578300e+00]
[   6.70000000e+02   7.56471800e+00]
[   6.80000000e+02   6.29808600e+00]
[   6.90000000e+02   5.15623400e+00]
[   7.00000000e+02   4.05390600e+00]
[   7.10000000e+02   3.06638600e+00]
[   7.20000000e+02   2.19250000e+00]
[   7.30000000e+02   1.53922800e+00]
[   7.40000000e+02   1.14938200e+00]
[   7.50000000e+02   9.05095000e-01]
[   7.60000000e+02   6.90947000e-01]
[   7.70000000e+02   5.08426000e-01]
[   7.80000000e+02   4.11766000e-01]]
```

read() → *colour.io.uprtek_sekonic.SpectralDistribution_UPRTek*

Read and parses the spectral data from a given *UPRTek CSV* file.

Returns *UPRTek* spectral distribution.

Return type `colour.SpectralDistribution_UPRTek`

Examples

```
>>> from os.path import dirname, join
>>> from colour import SpectralShape
>>> directory = join(dirname(__file__), "tests", "resources")
>>> sd = SpectralDistribution_UPRTek(
...     join(directory, "ESPD2021_0104_231446.xls")
... )
>>> print(sd.read().align(SpectralShape(380, 780, 10)))
...
UPRTek
======

Path                  : ...
Spectral Quantity     : irradiance
Reflection Geometry   : None
Transmission Geometry : None
Bandwidth (FWHM)      : None
Bandwidth Corrected   : None

Header
------

Manufacturer           : UPRTek
Catalog Number         : None
Description            : None
Document Creator       : None
Unique Identifier      : None
Measurement Equipment  : CV600
Laboratory             : None
Report Number          : None
Report Date            : 2021/01/04_23:14:46
Document Creation Date : None
Comments               : {...}

Spectral Data
-------------

[[  3.80000000e+02   3.02670000e-02]
 [  3.90000000e+02   3.52230000e-02]
 [  4.00000000e+02   1.93250000e-02]
 [  4.10000000e+02   2.94260000e-02]
 [  4.20000000e+02   8.76780000e-02]
 [  4.30000000e+02   6.32578000e-01]
 [  4.40000000e+02   3.62565900e+00]
 [  4.50000000e+02   1.42069180e+01]
 [  4.60000000e+02   1.70112970e+01]
 [  4.70000000e+02   1.19673130e+01]
 [  4.80000000e+02   8.42736200e+00]
 [  4.90000000e+02   7.97729800e+00]
 [  5.00000000e+02   8.71903600e+00]
 [  5.10000000e+02   9.55321500e+00]
 [  5.20000000e+02   9.90610500e+00]
 [  5.30000000e+02   9.91394400e+00]
 [  5.40000000e+02   9.74738000e+00]
 [  5.50000000e+02   9.53404900e+00]
 [  5.60000000e+02   9.27392200e+00]
```

(continues on next page)

(continued from previous page)

```
[  5.70000000e+02   9.02323400e+00]
[  5.80000000e+02   8.91788800e+00]
[  5.90000000e+02   9.11454600e+00]
[  6.00000000e+02   9.55787100e+00]
[  6.10000000e+02   1.00600760e+01]
[  6.20000000e+02   1.04846200e+01]
[  6.30000000e+02   1.05679540e+01]
[  6.40000000e+02   1.04359870e+01]
[  6.50000000e+02   9.82122300e+00]
[  6.60000000e+02   8.77578300e+00]
[  6.70000000e+02   7.56471800e+00]
[  6.80000000e+02   6.29808600e+00]
[  6.90000000e+02   5.15623400e+00]
[  7.00000000e+02   4.05390600e+00]
[  7.10000000e+02   3.06638600e+00]
[  7.20000000e+02   2.19250000e+00]
[  7.30000000e+02   1.53922800e+00]
[  7.40000000e+02   1.14938200e+00]
[  7.50000000e+02   9.05095000e-01]
[  7.60000000e+02   6.90947000e-01]
[  7.70000000e+02   5.08426000e-01]
[  7.80000000e+02   4.11766000e-01]]
```

colour.SpectralDistribution_Sekonic

class colour.**SpectralDistribution_Sekonic**(*path: str*, ***kwargs: Any*)

Bases: `colour.io.uprtek_sekonic.SpectralDistribution_UPRTek`

Implement support to read and write *IES TM-27-14* spectral data XML file from a *Sekonic CSV* file.

Parameters

- **path** (`str`) – Path for *Sekonic CSV* file.
- **kwargs** (`Any`) –

Return type None

Attributes

- `metadata`

Methods

- `__init__()`
- `__str__()`
- `read()`

Examples

```
>>> from os.path import dirname, join
>>> from colour import SpectralShape
>>> directory = join(dirname(__file__), "tests", "resources")
>>> sd = SpectralDistribution_Sekonic(
...     join(directory, "RANDOM_001_02._3262K.csv")
... )
>>> print(sd.read().align(SpectralShape(380, 780, 10)))
...
Sekonic
=======

Path                  : ...
Spectral Quantity     : irradiance
Reflection Geometry   : None
Transmission Geometry : None
Bandwidth (FWHM)      : None
Bandwidth Corrected   : None

Header
------

Manufacturer           : Sekonic
Catalog Number         : None
Description            : None
Document Creator       : None
Unique Identifier      : None
Measurement Equipment  : None
Laboratory             : None
Report Number          : None
Report Date            : 15/03/2021 3:44:14 p.m.
Document Creation Date : None
Comments               : {...}

Spectral Data
-------------

[[  3.80000000e+02   1.69406589e-21]
 [  3.90000000e+02   2.11758237e-22]
 [  4.00000000e+02   1.19813650e-05]
 [  4.10000000e+02   1.97110530e-05]
 [  4.20000000e+02   2.99661440e-05]
 [  4.30000000e+02   6.38192720e-05]
 [  4.40000000e+02   1.68909683e-04]
 [  4.50000000e+02   3.31902935e-04]
 [  4.60000000e+02   3.33143020e-04]
 [  4.70000000e+02   2.30227481e-04]
 [  4.80000000e+02   1.66981976e-04]
 [  4.90000000e+02   1.64439844e-04]
 [  5.00000000e+02   2.01534538e-04]
 [  5.10000000e+02   2.57840526e-04]
 [  5.20000000e+02   3.04612651e-04]
 [  5.30000000e+02   3.41368344e-04]
 [  5.40000000e+02   3.63639323e-04]
 [  5.50000000e+02   3.87050648e-04]
 [  5.60000000e+02   4.21619130e-04]
```

(continues on next page)

(continued from previous page)

```
 [  5.70000000e+02   4.58150520e-04]
 [  5.80000000e+02   5.01176575e-04]
 [  5.90000000e+02   5.40883630e-04]
 [  6.00000000e+02   5.71256795e-04]
 [  6.10000000e+02   5.83703280e-04]
 [  6.20000000e+02   5.57688472e-04]
 [  6.30000000e+02   5.17328095e-04]
 [  6.40000000e+02   4.39994939e-04]
 [  6.50000000e+02   3.62766819e-04]
 [  6.60000000e+02   2.96465587e-04]
 [  6.70000000e+02   2.43966802e-04]
 [  6.80000000e+02   2.04134776e-04]
 [  6.90000000e+02   1.75304012e-04]
 [  7.00000000e+02   1.52887544e-04]
 [  7.10000000e+02   1.29795619e-04]
 [  7.20000000e+02   1.03122693e-04]
 [  7.30000000e+02   8.77607820e-05]
 [  7.40000000e+02   7.61524130e-05]
 [  7.50000000e+02   7.06516880e-05]
 [  7.60000000e+02   3.72199210e-05]
 [  7.70000000e+02   3.63058860e-05]
 [  7.80000000e+02   3.55755470e-05]]
>>> sd.header.comments
>>> sd.metadata.keys()
>>> sd.write(join(directory, "RANDOM_001_02._3262K.spdx"))
...
```

__init__(*path: str*, ***kwargs: Any*) → None

Parameters

- **path** (str) –
- **kwargs** (Any) –

Return type None

__str__() → str

Return a formatted string representation of the *Sekonic* spectral distribution.

Returns Formatted string representation.

Return type str

Examples

```
>>> from os.path import dirname, join
>>> from colour import SpectralShape
>>> directory = join(dirname(__file__), "tests", "resources")
>>> sd = SpectralDistribution_UPRTek(
...     join(directory, "ESPD2021_0104_231446.xls")
... )
>>> print(sd.read().align(SpectralShape(380, 780, 10)))
...
UPRTek
======

Path                      : ...
```

(continues on next page)

(continued from previous page)

```
Spectral Quantity     : irradiance
Reflection Geometry   : None
Transmission Geometry : None
Bandwidth (FWHM)      : None
Bandwidth Corrected   : None

Header
------

Manufacturer           : UPRTek
Catalog Number         : None
Description            : None
Document Creator       : None
Unique Identifier      : None
Measurement Equipment  : CV600
Laboratory             : None
Report Number          : None
Report Date            : 2021/01/04_23:14:46
Document Creation Date : None
Comments               : {...}

Spectral Data
-------------

[[  3.80000000e+02   3.02670000e-02]
 [  3.90000000e+02   3.52230000e-02]
 [  4.00000000e+02   1.93250000e-02]
 [  4.10000000e+02   2.94260000e-02]
 [  4.20000000e+02   8.76780000e-02]
 [  4.30000000e+02   6.32578000e-01]
 [  4.40000000e+02   3.62565900e+00]
 [  4.50000000e+02   1.42069180e+01]
 [  4.60000000e+02   1.70112970e+01]
 [  4.70000000e+02   1.19673130e+01]
 [  4.80000000e+02   8.42736200e+00]
 [  4.90000000e+02   7.97729800e+00]
 [  5.00000000e+02   8.71903600e+00]
 [  5.10000000e+02   9.55321500e+00]
 [  5.20000000e+02   9.90610500e+00]
 [  5.30000000e+02   9.91394400e+00]
 [  5.40000000e+02   9.74738000e+00]
 [  5.50000000e+02   9.53404900e+00]
 [  5.60000000e+02   9.27392200e+00]
 [  5.70000000e+02   9.02323400e+00]
 [  5.80000000e+02   8.91788800e+00]
 [  5.90000000e+02   9.11454600e+00]
 [  6.00000000e+02   9.55787100e+00]
 [  6.10000000e+02   1.00600760e+01]
 [  6.20000000e+02   1.04846200e+01]
 [  6.30000000e+02   1.05679540e+01]
 [  6.40000000e+02   1.04359870e+01]
 [  6.50000000e+02   9.82122300e+00]
 [  6.60000000e+02   8.77578300e+00]
 [  6.70000000e+02   7.56471800e+00]
 [  6.80000000e+02   6.29808600e+00]
 [  6.90000000e+02   5.15623400e+00]
```

(continues on next page)

(continued from previous page)

```
[  7.00000000e+02   4.05390600e+00]
[  7.10000000e+02   3.06638600e+00]
[  7.20000000e+02   2.19250000e+00]
[  7.30000000e+02   1.53922800e+00]
[  7.40000000e+02   1.14938200e+00]
[  7.50000000e+02   9.05095000e-01]
[  7.60000000e+02   6.90947000e-01]
[  7.70000000e+02   5.08426000e-01]
[  7.80000000e+02   4.11766000e-01]]
```

read() → *colour.io.uprtek_sekonic.SpectralDistribution_Sekonic*

Read and parses the spectral data from a given *Sekonic Pseudo-XLS* file.

Returns *Sekonic* spectral distribution.

Return type `colour.SpectralDistribution_Sekonic`

Examples

```
>>> from os.path import dirname, join
>>> from colour import SpectralShape
>>> directory = join(dirname(__file__), "tests", "resources")
>>> sd = SpectralDistribution_Sekonic(
...     join(directory, "RANDOM_001_02._3262K.csv")
... )
>>> print(sd.read().align(SpectralShape(380, 780, 10)))
...
Sekonic
=======

Path                   : ...
Spectral Quantity      : irradiance
Reflection Geometry    : None
Transmission Geometry  : None
Bandwidth (FWHM)       : None
Bandwidth Corrected    : None

Header
------

Manufacturer            : Sekonic
Catalog Number          : None
Description             : None
Document Creator        : None
Unique Identifier       : None
Measurement Equipment   : None
Laboratory              : None
Report Number           : None
Report Date             : 15/03/2021 3:44:14 p.m.
Document Creation Date  : None
Comments                : {...}

Spectral Data
-------------

[[  3.80000000e+02   1.69406589e-21]
```

(continues on next page)

(continued from previous page)

```
[  3.90000000e+02   2.11758237e-22]
[  4.00000000e+02   1.19813650e-05]
[  4.10000000e+02   1.97110530e-05]
[  4.20000000e+02   2.99661440e-05]
[  4.30000000e+02   6.38192720e-05]
[  4.40000000e+02   1.68909683e-04]
[  4.50000000e+02   3.31902935e-04]
[  4.60000000e+02   3.33143020e-04]
[  4.70000000e+02   2.30227481e-04]
[  4.80000000e+02   1.66981976e-04]
[  4.90000000e+02   1.64439844e-04]
[  5.00000000e+02   2.01534538e-04]
[  5.10000000e+02   2.57840526e-04]
[  5.20000000e+02   3.04612651e-04]
[  5.30000000e+02   3.41368344e-04]
[  5.40000000e+02   3.63639323e-04]
[  5.50000000e+02   3.87050648e-04]
[  5.60000000e+02   4.21619130e-04]
[  5.70000000e+02   4.58150520e-04]
[  5.80000000e+02   5.01176575e-04]
[  5.90000000e+02   5.40883630e-04]
[  6.00000000e+02   5.71256795e-04]
[  6.10000000e+02   5.83703280e-04]
[  6.20000000e+02   5.57688472e-04]
[  6.30000000e+02   5.17328095e-04]
[  6.40000000e+02   4.39994939e-04]
[  6.50000000e+02   3.62766819e-04]
[  6.60000000e+02   2.96465587e-04]
[  6.70000000e+02   2.43966802e-04]
[  6.80000000e+02   2.04134776e-04]
[  6.90000000e+02   1.75304012e-04]
[  7.00000000e+02   1.52887544e-04]
[  7.10000000e+02   1.29795619e-04]
[  7.20000000e+02   1.03122693e-04]
[  7.30000000e+02   8.77607820e-05]
[  7.40000000e+02   7.61524130e-05]
[  7.50000000e+02   7.06516880e-05]
[  7.60000000e+02   3.72199210e-05]
[  7.70000000e+02   3.63058860e-05]
[  7.80000000e+02   3.55755470e-05]]
```

X-Rite Data

`colour`

`read_sds_from_xrite_file`(path)	Read the spectral data from given *X-Rite* file and returns it as a *dict* of `colour.SpectralDistribution` class instances.

colour.read_sds_from_xrite_file

colour.**read_sds_from_xrite_file**(*path: str*) → Dict[str, *colour.colorimetry.spectrum.SpectralDistribution*]

Read the spectral data from given *X-Rite* file and returns it as a *dict* of colour.SpectralDistribution class instances.

Parameters **path** (str) – Absolute *X-Rite* file path.

Returns *dict* of colour.SpectralDistribution class instances.

Return type dict

Notes

- This parser is minimalistic and absolutely not bullet-proof.

Examples

```
>>> import os
>>> from pprint import pprint
>>> xrite_file = os.path.join(
...     os.path.dirname(__file__),
...     "tests",
...     "resources",
...     "X-Rite_Digital_Colour_Checker.txt",
... )
>>> sds_data = read_sds_from_xrite_file(xrite_file)
>>> pprint(list(sds_data.keys()))
['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10']
```

Colour Models

Tristimulus Values, CIE xyY Colourspace and Chromaticity Coordinates

colour

XYZ_to_xyY(XYZ)	Convert from *CIE XYZ* tristimulus values to *CIE xyY* colourspace.
xyY_to_XYZ(xyY)	Convert from *CIE xyY* colourspace to *CIE XYZ* tristimulus values.
XYZ_to_xy(XYZ)	Return the *CIE xy* chromaticity coordinates from given *CIE XYZ* tristimulus values.
xy_to_XYZ(xy)	Return the *CIE XYZ* tristimulus values from given *CIE xy* chromaticity coordinates.
xyY_to_xy(xyY)	Convert from *CIE xyY* colourspace to *CIE xy* chromaticity coordinates.
xy_to_xyY(xy[, Y])	Convert from *CIE xy* chromaticity coordinates to *CIE xyY* colourspace by extending the array last dimension with given Y *luminance*.

colour.XYZ_to_xyY

`colour.XYZ_to_xyY(XYZ: ArrayLike) → NDArrayFloat`

Convert from *CIE XYZ* tristimulus values to *CIE xyY* colourspace.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.

Returns *CIE xyY* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[CIET14804f], [Wikipedia05d]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_xyY(XYZ)
array([ 0.5436955...,  0.3210794...,  0.1219722...])
```

colour.xyY_to_XYZ

`colour.xyY_to_XYZ(xyY: ArrayLike) → NDArrayFloat`

Convert from *CIE xyY* colourspace to *CIE XYZ* tristimulus values.

Parameters **xyY** (ArrayLike) – *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[CIET14804f], [Wikipedia05d]

Examples

```
>>> xyY = np.array([0.54369557, 0.32107944, 0.12197225])
>>> xyY_to_XYZ(xyY)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.XYZ_to_xy

colour.**XYZ_to_xy**(*XYZ: ArrayLike*) → NDArrayFloat

Return the *CIE xy* chromaticity coordinates from given *CIE XYZ* tristimulus values.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.

Returns *CIE xy* chromaticity coordinates.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[CIET14804f], [Wikipedia05d]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_xy(XYZ)
array([ 0.5436955...,  0.3210794...])
```

colour.xy_to_XYZ

colour.**xy_to_XYZ**(*xy: ArrayLike*) → NDArrayFloat

Return the *CIE XYZ* tristimulus values from given *CIE xy* chromaticity coordinates.

Parameters **xy** (ArrayLike) – *CIE xy* chromaticity coordinates.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
xy	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[CIET14804f], [Wikipedia05d]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_XYZ(xy)
array([ 1.6933366...,  1.        ,  0.4211574...])
```

colour.xyY_to_xy

colour.**xyY_to_xy**(*xyY: ArrayLike*) → NDArrayFloat

Convert from *CIE xyY* colourspace to *CIE xy* chromaticity coordinates.

`xyY` argument with last dimension being equal to 2 will be assumed to be a *CIE xy* chromaticity coordinates argument and will be returned directly by the definition.

Parameters **xyY** (`ArrayLike`) – *CIE xyY* colourspace array or *CIE xy* chromaticity coordinates.

Returns *CIE xy* chromaticity coordinates.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[CIET14804f], [Wikipedia05d]

Examples

```
>>> xyY = np.array([0.54369557, 0.32107944, 0.12197225])
>>> xyY_to_xy(xyY)
array([ 0.54369557...,  0.32107944...])
>>> xy = np.array([0.54369557, 0.32107944])
>>> xyY_to_xy(xy)
array([ 0.54369557...,  0.32107944...])
```

colour.xy_to_xyY

colour.**xy_to_xyY**(*xy: ArrayLike*, *Y: float = 1*) → NDArrayFloat

Convert from *CIE xy* chromaticity coordinates to *CIE xyY* colourspace by extending the array last dimension with given *Y luminance*.

xy argument with last dimension being equal to 3 will be assumed to be a *CIE xyY* colourspace array argument and will be returned directly by the definition.

Parameters

- **xy** (ArrayLike) – *CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y** (float) – Optional *Y luminance* value used to construct the *CIE xyY* colourspace array, the default *Y luminance* value is 1.

Returns *CIE xyY* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
xy	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

- This definition is a convenient object provided to implement support of illuminant argument *luminance* value in various colour.models package objects such as colour.Lab_to_XYZ() or colour.Luv_to_XYZ().

References

[CIET14804f], [Wikipedia05d]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_xyY(xy)
array([ 0.5436955...,  0.3210794...,  1.        ])
>>> xy = np.array([0.54369557, 0.32107944, 1.00000000])
>>> xy_to_xyY(xy)
array([ 0.5436955...,  0.3210794...,  1.        ])
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_xyY(xy, 100)
array([   0.5436955...,    0.3210794...,  100.        ])
```

Common Models

`colour`

`COLOURSPACE_MODELS`	Colourspace models supporting a direct conversion to *CIE XYZ* tristimulus values.

colour.COLOURSPACE_MODELS

`colour.`**`COLOURSPACE_MODELS = ('CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IPT Ragoo 2021', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT', 'Yrg')`**

Colourspace models supporting a direct conversion to *CIE XYZ* tristimulus values.

`colour.models`

`Jab_to_JCh`(Jab)	Convert from *Jab* colour representation to *JCh* colour representation.
`JCh_to_Jab`(JCh)	Convert from *JCh* colour representation to *Jab* colour representation.
`XYZ_to_Iab`(XYZ, LMS_to_LMS_p_callable, ...)	Convert from *CIE XYZ* tristimulus values to *IPT*-like Iab colour representation.
`Iab_to_XYZ`(Iab, LMS_p_to_LMS_callable, ...)	Convert from *IPT*-like Iab colour representation to *CIE XYZ* tristimulus values.

colour.models.Jab_to_JCh

`colour.models.`**`Jab_to_JCh`**(*Jab: ArrayLike*) → NDArrayFloat

Convert from *Jab* colour representation to *JCh* colour representation.

This definition is used to perform conversion from *CIE L*a*b** colourspace to *CIE L*C*Hab* colourspace and for other similar conversions. It implements a generic transformation from *Lightness* J, a and b opponent colour dimensions to the correlates of *Lightness* J, chroma C and hue angle h.

Parameters **`Jab`** (`ArrayLike`) – *Jab* colour representation array.

Returns *JCh* colour representation array.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
Jab	J : [0, 100] a : [-100, 100] b : [-100, 100]	J : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
JCh	J : [0, 100] C : [0, 100] h : [0, 360]	J : [0, 1] C : [0, 1] h : [0, 1]

References

[CIET14804h]

Examples

```
>>> Jab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Jab_to_JCh(Jab)
array([ 41.5278752...,  59.1242590...,  27.0884878...])
```

colour.models.JCh_to_Jab

colour.models.**JCh_to_Jab**(*JCh: ArrayLike*) → NDArrayFloat

Convert from *JCh* colour representation to *Jab* colour representation.

This definition is used to perform conversion from *CIE L*C*Hab* colourspace to *CIE L*a*b** colourspace and for other similar conversions. It implements a generic transformation from the correlates of *Lightness* J, chroma C and hue angle h to *Lightness* J, a and b opponent colour dimensions.

Parameters **JCh** (ArrayLike) – *JCh* colour representation array.

Returns *Jab* colour representation array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
JCh	J : [0, 100] C : [0, 100] h : [0, 360]	J : [0, 1] C : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jab	J : [0, 100] a : [-100, 100] b : [-100, 100]	J : [0, 1] a : [-1, 1] b : [-1, 1]

References

[CIET14804h]

Examples

```
>>> JCh = np.array([41.52787529, 59.12425901, 27.08848784])
>>> JCh_to_Jab(JCh)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

colour.models.XYZ_to_Iab

colour.models.**XYZ_to_Iab**(*XYZ: ArrayLike*, *LMS_to_LMS_p_callable: Callable*, *matrix_XYZ_to_LMS: ArrayLike*, *matrix_LMS_p_to_Iab: ArrayLike*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *IPT*-like Iab colour representation.

This definition is used to perform conversion from *CIE XYZ* tristimulus values to *IPT* colourspace and for other similar conversions. It implements a generic transformation from *CIE XYZ* tristimulus values to *Lightness* I, a and b representing red-green dimension, i.e. the dimension lost by protanopes and the yellow-blue dimension, i.e. the dimension lost by tritanopes, respectively.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **LMS_to_LMS_p_callable** (Callable) – Callable applying the forward non-linearity to the LMS colourspace array.
- **matrix_XYZ_to_LMS** (ArrayLike) – Matrix converting from *CIE XYZ* tristimulus values to LMS colourspace.
- **matrix_LMS_p_to_Iab** (ArrayLike) – Matrix converting from non-linear LMS_p colourspace to *IPT*-like Iab colour representation.

Returns *IPT*-like Iab colour representation.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Iab	I : [0, 1] a : [-1, 1] b : [-1, 1]	I : [0, 1] a : [-1, 1] b : [-1, 1]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> LMS_to_LMS_p = lambda x: x**0.43
>>> M_XYZ_to_LMS = np.array(
...     [
...         [0.4002, 0.7075, -0.0807],
...         [-0.2280, 1.1500, 0.0612],
...         [0.0000, 0.0000, 0.9184],
...     ]
... )
>>> M_LMS_p_to_Iab = np.array(
...     [
...         [0.4000, 0.4000, 0.2000],
...         [4.4550, -4.8510, 0.3960],
...         [0.8056, 0.3572, -1.1628],
...     ]
... )
>>> XYZ_to_Iab(XYZ, LMS_to_LMS_p, M_XYZ_to_LMS, M_LMS_p_to_Iab)
...
array([ 0.3842619...,  0.3848730...,  0.1888683...])
```

colour.models.Iab_to_XYZ

colour.models.**Iab_to_XYZ**(*Iab: ArrayLike*, *LMS_p_to_LMS_callable: Callable*, *matrix_Iab_to_LMS_p: ArrayLike*, *matrix_LMS_to_XYZ: ArrayLike*) → NDArrayFloat

Convert from *IPT*-like Iab colour representation to *CIE XYZ* tristimulus values.

This definition is used to perform conversion from *IPT* colourspace to *CIE XYZ* tristimulus values and for other similar conversions. It implements a generic transformation from *Lightness* I, a and b representing red-green dimension, i.e. the dimension lost by protanopes and the yellow-blue dimension, i.e. the dimension lost by tritanopes, respectively to *CIE XYZ* tristimulus values.

Parameters

- **Iab** (ArrayLike) – *IPT*-like Iab colour representation.
- **LMS_p_to_LMS_callable** (Callable) – Callable applying the reverse non-linearity to the LMS_p colourspace array.
- **matrix_Iab_to_LMS_p** (ArrayLike) – Matrix converting from *IPT*-like Iab colour representation to non-linear LMS_p colourspace.
- **matrix_LMS_to_XYZ** (ArrayLike) – Matrix converting from LMS colourspace to *CIE XYZ* tristimulus values.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Iab	I : [0, 1] a : [-1, 1] b : [-1, 1]	I : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> Iab = np.array([0.38426191, 0.38487306, 0.18886838])
>>> LMS_p_to_LMS = lambda x: x ** (1 / 0.43)
>>> M_Iab_to_LMS_p = np.linalg.inv(
...     np.array(
...         [
...             [0.4000, 0.4000, 0.2000],
...             [4.4550, -4.8510, 0.3960],
...             [0.8056, 0.3572, -1.1628],
...         ]
...     )
... )
>>> M_LMS_to_XYZ = np.linalg.inv(
...     np.array(
...         [
...             [0.4002, 0.7075, -0.0807],
...             [-0.2280, 1.1500, 0.0612],
...             [0.0000, 0.0000, 0.9184],
...         ]
...     )
... )
>>> Iab_to_XYZ(Iab, LMS_p_to_LMS, M_Iab_to_LMS_p, M_LMS_to_XYZ)
...
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

CIE L*a*b* Colourspace

colour

XYZ_to_Lab(XYZ[, illuminant])	Convert from *CIE XYZ* tristimulus values to *CIE L*a*b** colourspace.
Lab_to_XYZ(Lab[, illuminant])	Convert from *CIE L*a*b** colourspace to *CIE XYZ* tristimulus values.
Lab_to_LCHab(Lab)	Convert from *CIE L*a*b** colourspace to *CIE L*C*Hab* colourspace.
LCHab_to_Lab(LCHab)	Convert from *CIE L*C*Hab* colourspace to *CIE L*a*b** colourspace.

colour.XYZ_to_Lab

colour.**XYZ_to_Lab**(*XYZ: ArrayLike, illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *CIE L*a*b** colourspace.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE L*a*b** colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[CIET14804h]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Lab(XYZ)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

colour.Lab_to_XYZ

colour.**Lab_to_XYZ**(*Lab: ArrayLike, illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Convert from *CIE L*a*b** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Lab** (ArrayLike) – *CIE L*a*b** colourspace array.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[CIET14804h]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_XYZ(Lab)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.Lab_to_LCHab

colour.**Lab_to_LCHab**(*Lab: ArrayLike*) → NDArrayFloat

Convert from *CIE L*a*b** colourspace to *CIE L*C*Hab* colourspace.

Parameters **Lab** (ArrayLike) – *CIE L*a*b** colourspace array.

Returns *CIE L*C*Hab* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
LCHab	L : [0, 100] C : [0, 100] Hab : [0, 360]	L : [0, 1] C : [0, 1] Hab : [0, 1]

References

[CIET14804h]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_LCHab(Lab)
array([ 41.5278752...,  59.1242590...,  27.0884878...])
```

colour.LCHab_to_Lab

`colour.LCHab_to_Lab(LCHab: ArrayLike) → NDArrayFloat`

Convert from *CIE L*C*Hab* colourspace to *CIE L*a*b** colourspace.

Parameters **LCHab** (ArrayLike) – *CIE L*C*Hab* colourspace array.

Returns *CIE L*a*b** colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
LCHab	L : [0, 100] C : [0, 100] Hab : [0, 360]	L : [0, 1] C : [0, 1] Hab : [0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[CIET14804h]

Examples

```
>>> import numpy as np
>>> LCHab = np.array([41.52787529, 59.12425901, 27.08848784])
>>> LCHab_to_Lab(LCHab)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

CIE L*u*v* Colourspace

colour

XYZ_to_Luv(XYZ[, illuminant])	Convert from *CIE XYZ* tristimulus values to *CIE L*u*v** colourspace.
Luv_to_XYZ(Luv[, illuminant])	Convert from *CIE L*u*v** colourspace to *CIE XYZ* tristimulus values.
Luv_to_LCHuv(Luv)	Convert from *CIE L*u*v** colourspace to *CIE L*C*Huv* colourspace.
LCHuv_to_Luv(LCHuv)	Convert from *CIE L*C*Huv* colourspace to *CIE L*u*v** colourspace.
Luv_to_uv(Luv[, illuminant])	Return the uv^p chromaticity coordinates from given *CIE L*u*v** colourspace array.
uv_to_Luv(uv[, illuminant, Y])	Return the *CIE L*u*v** colourspace array from given uv^p chromaticity coordinates by extending the array last dimension with given L *Lightness*.
Luv_uv_to_xy(uv)	Return the *CIE xy* chromaticity coordinates from given *CIE L*u*v** colourspace uv^p chromaticity coordinates.
xy_to_Luv_uv(xy)	Return the *CIE L*u*v** colourspace uv^p chromaticity coordinates from given *CIE xy* chromaticity coordinates.

colour.XYZ_to_Luv

colour.**XYZ_to_Luv**(*XYZ: ArrayLike*, *illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *CIE L*u*v** colourspace.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE L*u*v** colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]

References

[CIET14804h], [Wikipedia07e]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Luv(XYZ)
array([ 41.5278752...,  96.8362605...,  17.7521014...])
```

colour.Luv_to_XYZ

colour.**Luv_to_XYZ**(*Luv: ArrayLike, illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Convert from *CIE L*u*v** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Luv** (ArrayLike) – *CIE L*u*v** colourspace array.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[CIET14804h], [Wikipedia07e]

Examples

```
>>> import numpy as np
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_XYZ(Luv)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.Luv_to_LCHuv

colour.**Luv_to_LCHuv**(*Luv: ArrayLike*) → NDArrayFloat

Convert from *CIE L*u*v** colourspace to *CIE L*C*Huv* colourspace.

Parameters **Luv** (ArrayLike) – *CIE L*u*v** colourspace array.

Returns *CIE L*C*Huv* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]

Range	Scale - Reference	Scale - 1
LCHuv	L : [0, 100] C : [0, 100] Huv : [0, 360]	L : [0, 1] C : [0, 1] Huv : [0, 1]

References

[CIET14804h]

Examples

```
>>> import numpy as np
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_LCHuv(Luv)
array([ 41.5278752...,  98.4499795...,  10.3881634...])
```

colour.LCHuv_to_Luv

colour.**LCHuv_to_Luv**(*LCHuv: ArrayLike*) → NDArrayFloat

Convert from *CIE L*C*Huv* colourspace to *CIE L*u*v** colourspace.

Parameters **LCHuv** (ArrayLike) – *CIE L*C*Huv* colourspace array.

Returns *CIE L*u*v** colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
LCHuv	L : [0, 100] C : [0, 100] Huv : [0, 360]	L : [0, 1] C : [0, 1] Huv : [0, 1]

Range	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]

References

[CIET14804h]

Examples

```
>>> import numpy as np
>>> LCHuv = np.array([41.52787529, 98.44997950, 10.38816348])
>>> LCHuv_to_Luv(LCHuv)
array([ 41.5278752...,  96.8362605...,  17.7521014...])
```

colour.Luv_to_uv

colour.**Luv_to_uv**(*Luv: ArrayLike*, *illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Return the uv^p chromaticity coordinates from given *CIE L*u*v** colourspace array.

Parameters

- **Luv** (ArrayLike) – *CIE L*u*v** colourspace array.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns uv^p chromaticity coordinates.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]
illuminant	[0, 1]	[0, 1]

References

[CIET14804g]

Examples

```
>>> import numpy as np
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_uv(Luv)
array([ 0.3772021...,  0.5012026...])
```

colour.uv_to_Luv

colour.**uv_to_Luv**(*uv: ArrayLike*, *illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*, *Y: float = 1*) → NDArrayFloat

Return the *CIE L*u*v** colourspace array from given uv^p chromaticity coordinates by extending the array last dimension with given L *Lightness*.

Parameters

- **uv** (ArrayLike) – uv^p chromaticity coordinates.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y** (float) – Optional Y *luminance* value used to construct the intermediate *CIE XYZ* colourspace array, the default Y *luminance* value is 1.

Returns *CIE L*u*v** colourspace array.

Return type numpy.ndarray

Notes

Range	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]
illuminant	[0, 1]	[0, 1]

References

[CIET14804g]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.37720213, 0.50120264])
>>> uv_to_Luv(uv)
array([ 100.        ,  233.1837603...,   42.7474385...])
```

colour.Luv_uv_to_xy

colour.**Luv_uv_to_xy**(*uv: ArrayLike*) → NDArrayFloat

Return the *CIE xy* chromaticity coordinates from given *CIE L*u*v** colourspace uv^p chromaticity coordinates.

Parameters **uv** (ArrayLike) – *CIE L*u*v* u"v"* chromaticity coordinates.

Returns *CIE xy* chromaticity coordinates.

Return type numpy.ndarray

References

[Wikipedia07b]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.37720213, 0.50120264])
>>> Luv_uv_to_xy(uv)
array([ 0.5436955...,  0.3210794...])
```

colour.xy_to_Luv_uv

colour.**xy_to_Luv_uv**(*xy: ArrayLike*) → NDArrayFloat

Return the *CIE L*u*v** colourspace uv^p chromaticity coordinates from given *CIE xy* chromaticity coordinates.

Parameters **xy** (ArrayLike) – *CIE xy* chromaticity coordinates.

Returns *CIE L*u*v* u"v"* chromaticity coordinates.

Return type numpy.ndarray

References

[Wikipedia07e]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.54369558, 0.32107944])
>>> xy_to_Luv_uv(xy)
array([ 0.3772021...,  0.5012026...])
```

CIE 1960 UCS Colourspace

colour

XYZ_to_UCS(XYZ)	Convert from *CIE XYZ* tristimulus values to *CIE 1960 UCS* colourspace.
UCS_to_XYZ(UVW)	Convert from *CIE 1960 UCS* colourspace to *CIE XYZ* tristimulus values.
UCS_to_uv(UVW)	Return the *uv* chromaticity coordinates from given *CIE 1960 UCS* colourspace array.
uv_to_UCS(uv[, V])	Return the *CIE 1960 UCS* colourspace array from given *uv* chromaticity coordinates.
UCS_uv_to_xy(uv)	Return the *CIE xy* chromaticity coordinates from given *CIE 1960 UCS* colourspace *uv* chromaticity coordinates.
xy_to_UCS_uv(xy)	Return the *CIE 1960 UCS* colourspace *uv* chromaticity coordinates from given *CIE xy* chromaticity coordinates.

colour.XYZ_to_UCS

colour.**XYZ_to_UCS**(*XYZ: ArrayLike*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *CIE 1960 UCS* colourspace.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.

Returns *CIE 1960 UCS* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
UVW	[0, 1]	[0, 1]

References

[Wikipedia08b], [Wikipedia08c]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_UCS(XYZ)
array([ 0.1376933...,  0.1219722...,  0.1053731...])
```

colour.UCS_to_XYZ

colour.**UCS_to_XYZ**(*UVW: ArrayLike*) → NDArrayFloat

Convert from *CIE 1960 UCS* colourspace to *CIE XYZ* tristimulus values.

Parameters **UVW** (ArrayLike) – *CIE 1960 UCS* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
UVW	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Wikipedia08b], [Wikipedia08c]

Examples

```
>>> import numpy as np
>>> UVW = np.array([0.13769339, 0.12197225, 0.10537310])
>>> UCS_to_XYZ(UVW)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.UCS_to_uv

colour.**UCS_to_uv**(*UVW: ArrayLike*) → NDArrayFloat

Return the *uv* chromaticity coordinates from given *CIE 1960 UCS* colourspace array.

Parameters **UVW** (ArrayLike) – *CIE 1960 UCS* colourspace array.

Returns *uv* chromaticity coordinates.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
UVW	[0, 1]	[0, 1]

References

[Wikipedia08b]

Examples

```
>>> import numpy as np
>>> UVW = np.array([0.13769339, 0.12197225, 0.10537310])
>>> UCS_to_uv(UVW)
array([ 0.3772021...,  0.3341350...])
```

colour.uv_to_UCS

colour.**uv_to_UCS**(*uv: ArrayLike*, *V: float = 1*) → NDArrayFloat

Return the *CIE 1960 UCS* colourspace array from given *uv* chromaticity coordinates.

Parameters

- **uv** (ArrayLike) – *uv* chromaticity coordinates.
- **V** (float) – Optional *V luminance* value used to construct the *CIE 1960 UCS* colourspace array, the default *V luminance* is set to 1.

Returns *CIE 1960 UCS* colourspace array.

Return type numpy.ndarray

References

[Wikipedia08b]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.37720213, 0.33413508])
>>> uv_to_UCS(uv)
array([ 1.1288911...,  1.        ,  0.8639104...])
```

colour.UCS_uv_to_xy

colour.**UCS_uv_to_xy**(*uv: ArrayLike*) → NDArrayFloat

Return the *CIE xy* chromaticity coordinates from given *CIE 1960 UCS* colourspace *uv* chromaticity coordinates.

Parameters **uv** (ArrayLike) – *CIE UCS uv* chromaticity coordinates.

Returns *CIE xy* chromaticity coordinates.

Return type numpy.ndarray

References

[Wikipedia08b]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.37720213, 0.33413508])
>>> UCS_uv_to_xy(uv)
array([ 0.5436955...,  0.3210794...])
```

colour.xy_to_UCS_uv

colour.**xy_to_UCS_uv**(*xy: ArrayLike*) → NDArrayFloat

Return the *CIE 1960 UCS* colourspace *uv* chromaticity coordinates from given *CIE xy* chromaticity coordinates.

Parameters **xy** (ArrayLike) – *CIE xy* chromaticity coordinates.

Returns *CIE UCS uv* chromaticity coordinates.

Return type numpy.ndarray

References

[Wikipedia08b]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.54369555, 0.32107941])
>>> xy_to_UCS_uv(xy)
array([ 0.3772021...,  0.3341350...])
```

CIE 1964 U*V*W* Colourspace

colour

XYZ_to_UVW(XYZ[, illuminant])	Convert from *CIE XYZ* tristimulus values to *CIE 1964 U*V*W** colourspace.
UVW_to_XYZ(UVW[, illuminant])	Convert *CIE 1964 U*V*W** colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_UVW

colour.**XYZ_to_UVW**(*XYZ: ArrayLike*, *illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *CIE 1964 U*V*W** colourspace.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE 1964 U*V*W** colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
UVW	U : [-100, 100] V : [-100, 100] W : [0, 100]	U : [-1, 1] V : [-1, 1] W : [0, 1]

References

[Wikipedia08d]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> XYZ_to_UVW(XYZ)
array([ 94.5503572...,  11.5553652...,  40.5475740...])
```

colour.UVW_to_XYZ

colour.**UVW_to_XYZ**(*UVW: ArrayLike*, *illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Convert *CIE 1964 U*V*W** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **UVW** (ArrayLike) – *CIE 1964 U*V*W** colourspace array.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
UVW	U : [-100, 100] V : [-100, 100] W : [0, 100]	U : [-1, 1] V : [-1, 1] W : [0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100	[0, 1]

References

[Wikipedia08d]

Examples

```
>>> import numpy as np
>>> UVW = np.array([94.55035725, 11.55536523, 40.54757405])
>>> UVW_to_XYZ(UVW)
array([ 20.654008,  12.197225,   5.136952])
```

Hunter L,a,b Colour Scale

colour

XYZ_to_Hunter_Lab(XYZ[, XYZ_n, K_ab])	Convert from *CIE XYZ* tristimulus values to *Hunter L,a,b* colour scale.
Hunter_Lab_to_XYZ(Lab[, XYZ_n, K_ab])	Convert from *Hunter L,a,b* colour scale to *CIE XYZ* tristimulus values.
XYZ_to_K_ab_HunterLab1966(XYZ)	Convert 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: ArrayLike, XYZ_n: ArrayLike = TVS_ILLUMINANTS_HUNTERLAB['CIE 1931 2 Degree Standard Observer']['D65'].XYZ_n, K_ab: ArrayLike = TVS_ILLUMINANTS_HUNTERLAB['CIE 1931 2 Degree Standard Observer']['D65'].K_ab*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Hunter L,a,b* colour scale.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **XYZ_n** (ArrayLike) – Reference *illuminant* tristimulus values.
- **K_ab** (ArrayLike) – 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 numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[HunterLab08a]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> D65 = TVS_ILLUMINANTS_HUNTERLAB["CIE 1931 2 Degree Standard Observer"][
...     "D65"
... ]
>>> XYZ_to_Hunter_Lab(XYZ, D65.XYZ_n, D65.K_ab)
array([ 34.9245257...,  47.0618985...,  14.3861510...])
```

colour.Hunter_Lab_to_XYZ

colour.**Hunter_Lab_to_XYZ**(*Lab: ArrayLike, XYZ_n: ArrayLike = TVS_ILLUMINANTS_HUNTERLAB['CIE 1931 2 Degree Standard Observer']['D65'].XYZ_n, K_ab: ArrayLike = TVS_ILLUMINANTS_HUNTERLAB['CIE 1931 2 Degree Standard Observer']['D65'].K_ab*) → NDArrayFloat

Convert from *Hunter L,a,b* colour scale to *CIE XYZ* tristimulus values.

Parameters

- **Lab** (ArrayLike) – *Hunter L,a,b* colour scale array.
- **XYZ_n** (ArrayLike) – Reference *illuminant* tristimulus values.
- **K_ab** (ArrayLike) – 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 numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[HunterLab08a]

Examples

```
>>> Lab = np.array([34.92452577, 47.06189858, 14.38615107])
>>> D65 = TVS_ILLUMINANTS_HUNTERLAB["CIE 1931 2 Degree Standard Observer"][
...     "D65"
... ]
>>> Hunter_Lab_to_XYZ(Lab, D65.XYZ_n, D65.K_ab)
array([ 20.654008,  12.197225,   5.136952])
```

colour.XYZ_to_K_ab_HunterLab1966

colour.**XYZ_to_K_ab_HunterLab1966**(*XYZ: ArrayLike*) → NDArrayFloat

Convert from *whitepoint CIE XYZ* tristimulus values to *Hunter L,a,b* K_a and K_b chromaticity coefficients.

Parameters **XYZ** (ArrayLike) – *Whitepoint CIE XYZ* tristimulus values.

Returns *Hunter L,a,b* K_a and K_b chromaticity coefficients.

Return type numpy.ndarray

References

[HunterLab08b]

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])	Convert from *CIE XYZ* tristimulus values to *Hunter Rd,a,b* colour scale.
Hunter_Rdab_to_XYZ(R_d_ab[, XYZ_n, K_ab])	Convert from *Hunter Rd,a,b* colour scale to *CIE XYZ* tristimulus values.

colour.XYZ_to_Hunter_Rdab

colour.**XYZ_to_Hunter_Rdab**(*XYZ: ArrayLike, XYZ_n: ArrayLike = TVS_ILLUMINANTS_HUNTERLAB['CIE 1931 2 Degree Standard Observer']['D65'].XYZ_n, K_ab: ArrayLike = TVS_ILLUMINANTS_HUNTERLAB['CIE 1931 2 Degree Standard Observer']['D65'].K_ab*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Hunter Rd,a,b* colour scale.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **XYZ_n** (ArrayLike) – Reference *illuminant* tristimulus values.
- **K_ab** (ArrayLike) – 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 numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
R_d_ab	R_d : [0, 100] a_Rd : [-100, 100] b_Rd : [-100, 100]	R_d : [0, 1] a_Rd : [-1, 1] b_Rd : [-1, 1]

References

[HunterLab12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> D65 = TVS_ILLUMINANTS_HUNTERLAB["CIE 1931 2 Degree Standard Observer"][
...     "D65"
... ]
>>> XYZ_to_Hunter_Rdab(XYZ, D65.XYZ_n, D65.K_ab)
...
array([ 12.197225 ...,  57.1253787...,  17.4624134...])
```

colour.Hunter_Rdab_to_XYZ

colour.**Hunter_Rdab_to_XYZ**(*R_d_ab: ArrayLike, XYZ_n: ArrayLike = TVS_ILLUMINANTS_HUNTERLAB['CIE 1931 2 Degree Standard Observer']['D65'].XYZ_n, K_ab: ArrayLike = TVS_ILLUMINANTS_HUNTERLAB['CIE 1931 2 Degree Standard Observer']['D65'].K_ab*) → NDArrayFloat

Convert from *Hunter Rd,a,b* colour scale to *CIE XYZ* tristimulus values.

Parameters

- **R_d_ab** (ArrayLike) – *Hunter Rd,a,b* colour scale array.
- **XYZ_n** (ArrayLike) – Reference *illuminant* tristimulus values.
- **K_ab** (ArrayLike) – 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 numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
R_d_ab	R_d : [0, 100] a_Rd : [-100, 100] b_Rd : [-100, 100]	R_d : [0, 1] a_Rd : [-1, 1] b_Rd : [-1, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[HunterLab12]

Examples

```
>>> import numpy as np
>>> R_d_ab = np.array([12.19722500, 57.12537874, 17.46241341])
>>> D65 = TVS_ILLUMINANTS_HUNTERLAB["CIE 1931 2 Degree Standard Observer"][
...     "D65"
... ]
>>> Hunter_Rdab_to_XYZ(R_d_ab, D65.XYZ_n, D65.K_ab)
array([ 20.654008,  12.197225,   5.136952])
```

DIN99 Colourspace and DIN99b, DIN99c, DIN99d Refined Formulas

`colour`

`Lab_to_DIN99`(Lab[, k_E, k_CH, method])	Convert from *CIE L*a*b** colourspace to *DIN99* colourspace or one of the *DIN99b*, *DIN99c*, *DIN99d* refined formulas according to *Cui et al. (2002)*.
`DIN99_to_Lab`(Lab_99[, k_E, k_CH, method])	Convert from *DIN99* colourspace or one of the *DIN99b*, *DIN99c*, *DIN99d* refined formulas according to *Cui et al. (2002)* to *CIE L*a*b** colourspace.
`XYZ_to_DIN99`(XYZ[, illuminant, k_E, k_CH, ...])	Convert from *CIE XYZ* tristimulus values to *DIN99* colourspace or one of the *DIN99b*, *DIN99c*, *DIN99d* refined formulas according to *Cui et al. (2002)*.
`DIN99_to_XYZ`(Lab_99[, illuminant, k_E, ...])	Convert from *DIN99* colourspace or one of the *DIN99b*, *DIN99c*, *DIN99d* refined formulas according to *Cui et al. (2002)* to *CIE XYZ* tristimulus values.

colour.Lab_to_DIN99

`colour.`**`Lab_to_DIN99`**(*Lab: ArrayLike, k_E: float = 1, k_CH: float = 1, method: Union[Literal['ASTMD2244-07', 'DIN99', 'DIN99b', 'DIN99c', 'DIN99d'], str] = 'DIN99'*) → NDArrayFloat

Convert from *CIE L*a*b** colourspace to *DIN99* colourspace or one of the *DIN99b*, *DIN99c*, *DIN99d* refined formulas according to *Cui et al. (2002)*.

Parameters

- **Lab** (`ArrayLike`) – *CIE L*a*b** colourspace array.
- **k_E** (`float`) – Parametric factor K_E used to compensate for texture and other specimen presentation effects.
- **k_CH** (`float`) – Parametric factor K_{CH} used to compensate for texture and other specimen presentation effects.
- **method** (`Union[Literal['ASTMD2244-07', 'DIN99', 'DIN99b', 'DIN99c', 'DIN99d'], str]`) – Computation method to choose between the [ASTMInternational07] formula and the refined formulas according to *Cui et al. (2002)*.

Returns *DIN99* colourspace array.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
Lab_99	L_99 : [0, 100] a_99 : [-100, 100] b_99 : [-100, 100]	L_99 : [0, 1] a_99 : [-1, 1] b_99 : [-1, 1]

References

[ASTMInternational07], [CLR+02]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_DIN99(Lab)
array([ 53.2282198...,  28.4163465...,   3.8983955...])
```

colour.DIN99_to_Lab

colour.**DIN99_to_Lab**(*Lab_99: ArrayLike, k_E: float = 1, k_CH: float = 1, method: Union[Literal['ASTMD2244-07', 'DIN99', 'DIN99b', 'DIN99c', 'DIN99d'], str] = 'DIN99'*) → NDArrayFloat

Convert from *DIN99* colourspace or one of the *DIN99b*, *DIN99c*, *DIN99d* refined formulas according to *Cui et al. (2002)* to *CIE L*a*b** colourspace.

Parameters

- **Lab_99** (ArrayLike) – *DIN99* colourspace array.
- **k_E** (float) – Parametric factor K_E used to compensate for texture and other specimen presentation effects.
- **k_CH** (float) – Parametric factor K_{CH} used to compensate for texture and other specimen presentation effects.
- **method** (Union[Literal['ASTMD2244-07', 'DIN99', 'DIN99b', 'DIN99c', 'DIN99d'], str]) – Computation method to choose between the [ASTMInternational07] formula and the refined formulas according to *Cui et al. (2002)*.

Returns *CIE L*a*b** colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_99	L_99 : [0, 100] a_99 : [-100, 100] b_99 : [-100, 100]	L_99 : [0, 1] a_99 : [-1, 1] b_99 : [-1, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[ASTMInternational07], [CLR+02]

Examples

```
>>> import numpy as np
>>> Lab_99 = np.array([53.22821988, 28.41634656, 3.89839552])
>>> DIN99_to_Lab(Lab_99)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

colour.XYZ_to_DIN99

colour.**XYZ_to_DIN99**(*XYZ: ArrayLike, illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65'], k_E: float = 1, k_CH: float = 1, method: Union[Literal['ASTMD2244-07', 'DIN99', 'DIN99b', 'DIN99c', 'DIN99d'], str] = 'DIN99'*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *DIN99* colourspace or one of the *DIN99b*, *DIN99c*, *DIN99d* refined formulas according to *Cui et al. (2002)*.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **k_E** (float) – Parametric factor K_E used to compensate for texture and other specimen presentation effects.
- **k_CH** (float) – Parametric factor K_{CH} used to compensate for texture and other specimen presentation effects.
- **method** (Union[Literal['ASTMD2244-07', 'DIN99', 'DIN99b', 'DIN99c', 'DIN99d'], str]) – Computation method to choose between the [ASTMInternational07] formula and the refined formulas according to *Cui et al. (2002)*.

Returns *DIN99* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab_99	L_99 : [0, 100] a_99 : [-100, 100] b_99 : [-100, 100]	L_99 : [0, 1] a_99 : [-1, 1] b_99 : [-1, 1]

References

[ASTMInternational07]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_DIN99(XYZ)
array([ 53.2282198...,  28.4163465...,   3.8983955...])
```

colour.DIN99_to_XYZ

colour.**DIN99_to_XYZ**(*Lab_99: ArrayLike, illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65'], k_E: float = 1, k_CH: float = 1, method: Union[Literal['ASTMD2244-07', 'DIN99', 'DIN99b', 'DIN99c', 'DIN99d'], str] = 'DIN99'*) → NDArrayFloat

Convert from *DIN99* colourspace or one of the *DIN99b*, *DIN99c*, *DIN99d* refined formulas according to *Cui et al. (2002)* to *CIE XYZ* tristimulus values.

Parameters

- **Lab_99** (ArrayLike) – *DIN99* colourspace array.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **k_E** (float) – Parametric factor K_E used to compensate for texture and other specimen presentation effects.
- **k_CH** (float) – Parametric factor K_{CH} used to compensate for texture and other specimen presentation effects.
- **method** (Union[Literal['ASTMD2244-07', 'DIN99', 'DIN99b', 'DIN99c', 'DIN99d'], str]) – Computation method to choose between the [ASTMInternational07] formula and the refined formulas according to *Cui et al. (2002)*.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_99	L_99 : [0, 100] a_99 : [-100, 100] b_99 : [-100, 100]	L_99 : [0, 1] a_99 : [-1, 1] b_99 : [-1, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[ASTMInternational07]

Examples

```
>>> import numpy as np
>>> Lab_99 = np.array([53.22821989, 28.41634656, 3.89839552])
>>> DIN99_to_XYZ(Lab_99)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006)

colour

JMh_CIECAM02_to_CAM02LCD(JMh)	Convert from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.
CAM02LCD_to_JMh_CIECAM02(Jpapbp)	Convert from *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.
JMh_CIECAM02_to_CAM02SCD(JMh)	Convert from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.
CAM02SCD_to_JMh_CIECAM02(Jpapbp)	Convert from *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.
JMh_CIECAM02_to_CAM02UCS(JMh)	Convert from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.
CAM02UCS_to_JMh_CIECAM02(Jpapbp)	Convert from *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.
XYZ_to_CAM02LCD(XYZ, **kwargs)	Convert from *CIE XYZ* tristimulus values to *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.
CAM02LCD_to_XYZ(Jpapbp, **kwargs)	Convert from *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.
XYZ_to_CAM02SCD(XYZ, **kwargs)	Convert from *CIE XYZ* tristimulus values to *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.
CAM02SCD_to_XYZ(Jpapbp, **kwargs)	Convert from *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.
XYZ_to_CAM02UCS(XYZ, **kwargs)	Convert from *CIE XYZ* tristimulus values to *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.
CAM02UCS_to_XYZ(Jpapbp, **kwargs)	Convert from *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.

colour.JMh_CIECAM02_to_CAM02LCD

colour.**JMh_CIECAM02_to_CAM02LCD**(*JMh: ArrayLike*) → NDArrayFloat

Convert from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Parameters **JMh** (ArrayLike) – *CIECAM02* correlates array JMh.

Returns *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Return type numpy.ndarray

Notes

- *LCD* in *CAM02-LCD* stands for *Large Colour Differences*.

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LCL06]

Examples

```
>>> from colour.appearance import (
...     VIEWING_CONDITIONS_CIECAM02,
...     XYZ_to_CIECAM02,
... )
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = VIEWING_CONDITIONS_CIECAM02["Average"]
>>> specification = XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02LCD(JMh)
array([ 54.9043313...,  -0.0845039...,  -0.0685483...])
```

colour.CAM02LCD_to_JMh_CIECAM02

colour.**CAM02LCD_to_JMh_CIECAM02**(*Jpapbp: ArrayLike*) → NDArrayFloat

Convert from *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.

Parameters **Jpapbp** (ArrayLike) – *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array JMh.

Return type numpy.ndarray

Notes

- *LCD* in *CAM02-LCD* stands for *Large Colour Differences*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08450395, -0.06854831])
>>> CAM02LCD_to_JMh_CIECAM02(Jpapbp)
array([  4.1731091...e+01,   1.0884217...e-01,   2.1904843...e+02])
```

colour.JMh_CIECAM02_to_CAM02SCD

colour.**JMh_CIECAM02_to_CAM02SCD**(*JMh: ArrayLike*) → NDArrayFloat

Convert from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Parameters **JMh** (ArrayLike) – *CIECAM02* correlates array JMh.

Returns *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Return type numpy.ndarray

Notes

- *SCD* in *CAM02-SCD* stands for *Small Colour Differences*.

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LCL06]

Examples

```
>>> from colour.appearance import (
...     VIEWING_CONDITIONS_CIECAM02,
...     XYZ_to_CIECAM02,
... )
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = VIEWING_CONDITIONS_CIECAM02["Average"]
>>> specification = XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02SCD(JMh)
array([ 54.9043313...,  -0.0843617...,  -0.0684329...])
```

colour.CAM02SCD_to_JMh_CIECAM02

colour.**CAM02SCD_to_JMh_CIECAM02**(*Jpapbp: ArrayLike*) → NDArrayFloat

Convert from *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.

Parameters **Jpapbp** (ArrayLike) – *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array JMh.

Return type numpy.ndarray

Notes

- *SCD* in *CAM02-SCD* stands for *Small Colour Differences*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08436178, -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: ArrayLike*) → NDArrayFloat

Convert from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Parameters **JMh** (ArrayLike) – *CIECAM02* correlates array JMh.

Returns *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Return type numpy.ndarray

Notes

- *UCS* in *CAM02-UCS* stands for *Uniform Colour Colourspace*.

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LCL06]

Examples

```
>>> from colour.appearance import (
...     VIEWING_CONDITIONS_CIECAM02,
...     XYZ_to_CIECAM02,
... )
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = VIEWING_CONDITIONS_CIECAM02["Average"]
```

(continues on next page)

(continued from previous page)

```
>>> 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: ArrayLike*) → NDArrayFloat

Convert from *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.

Parameters **Jpapbp** (ArrayLike) – *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array JMh.

Return type numpy.ndarray

Notes

- *UCS* in *CAM02-UCS* stands for *Uniform Colour Colourspace*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08442362, -0.06848314])
>>> CAM02UCS_to_JMh_CIECAM02(Jpapbp)
array([  4.1731091...e+01,   1.0884217...e-01,   2.1904843...e+02])
```

colour.XYZ_to_CAM02LCD

colour.**XYZ_to_CAM02LCD**(*XYZ: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **kwargs** (Any) – {colour.XYZ_to_CIECAM02()}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.

Returns *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Return type numpy.ndarray

Warning: The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *LCD* in *CAM02-LCD* stands for *Large Colour Differences*.

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LCL06]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_CAM02LCD(XYZ)
array([ 46.6138615...,  39.3576023...,  15.9673043...])
```

colour.CAM02LCD_to_XYZ

colour.**CAM02LCD_to_XYZ**(*Jpapbp: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Convert from *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.

Parameters

- **Jpapbp** (ArrayLike) – *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.
- **kwargs** (Any) – {colour.CIECAM02_to_XYZ()}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

> **Warning:** The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *LCD* in *CAM02-LCD* stands for *Large Colour Differences*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([46.61386154, 39.35760236, 15.96730435])
>>> CAM02LCD_to_XYZ(Jpapbp)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.XYZ_to_CAM02SCD

colour.**XYZ_to_CAM02SCD**(*XYZ: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **kwargs** (Any) – {colour.XYZ_to_CIECAM02()}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.

Returns *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Return type numpy.ndarray

> **Warning:** The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *SCD* in *CAM02-SCD* stands for *Small Colour Differences*.

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LCL06]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_CAM02SCD(XYZ)
array([ 46.6138615...,  25.6287988...,  10.3975548...])
```

colour.CAM02SCD_to_XYZ

colour.**CAM02SCD_to_XYZ**(*Jpapbp: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Convert from *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.

Parameters

- **Jpapbp** (ArrayLike) – *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.
- **kwargs** (Any) – {colour.CIECAM02_to_XYZ()}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Warning: The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *SCD* in *CAM02-SCD* stands for *Small Colour Differences*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([46.61386154, 25.62879882, 10.39755489])
>>> CAM02SCD_to_XYZ(Jpapbp)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.XYZ_to_CAM02UCS

colour.**XYZ_to_CAM02UCS**(*XYZ: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **kwargs** (Any) – {colour.XYZ_to_CIECAM02()}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.

Returns *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Return type numpy.ndarray

> **Warning:** The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *UCS* in *CAM02-UCS* stands for *Uniform Colour Space*.

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LCL06]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_CAM02UCS(XYZ)
array([ 46.6138615...,  29.8831001...,  12.1235168...])
```

colour.CAM02UCS_to_XYZ

colour.**CAM02UCS_to_XYZ**(*Jpapbp: ArrayLike*, ***kwargs: Any*) → NDArrayFloat

Convert from *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.

Parameters

- **Jpapbp** (ArrayLike) – *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.
- **kwargs** (Any) – {colour.CIECAM02_to_XYZ()}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

> **Warning:** The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *UCS* in *CAM02-UCS* stands for *Uniform Colour Space*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([46.61386154, 29.88310013, 12.12351683])
>>> CAM02UCS_to_XYZ(Jpapbp)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017)

colour

JMh_CAM16_to_CAM16LCD(JMh, *[, coefficients])	Convert from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.
CAM16LCD_to_JMh_CAM16(Jpapbp, *[, coefficients])	Convert from *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.
JMh_CAM16_to_CAM16SCD(JMh, *[, coefficients])	Convert from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.
CAM16SCD_to_JMh_CAM16(Jpapbp, *[, coefficients])	Convert from *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.
JMh_CAM16_to_CAM16UCS(JMh, *[, coefficients])	Convert from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.
CAM16UCS_to_JMh_CAM16(Jpapbp, *[, coefficients])	Convert from *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.
XYZ_to_CAM16LCD(XYZ, *[, coefficients])	Convert from *CIE XYZ* tristimulus values to *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.
CAM16LCD_to_XYZ(Jpapbp, *[, coefficients])	Convert from *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.
XYZ_to_CAM16SCD(XYZ, *[, coefficients])	Convert from *CIE XYZ* tristimulus values to *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.
CAM16SCD_to_XYZ(Jpapbp, *[, coefficients])	Convert from *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.
XYZ_to_CAM16UCS(XYZ, *[, coefficients])	Convert from *CIE XYZ* tristimulus values to *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.
CAM16UCS_to_XYZ(Jpapbp, *[, coefficients])	Convert from *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.

colour.JMh_CAM16_to_CAM16LCD

colour.**JMh_CAM16_to_CAM16LCD**(*JMh, *, coefficients=Coefficients_UCS_Luo2006(K_L=0.77, c_1=0.007, c_2=0.0053)*)

Convert from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Parameters **JMh** – *CAM16* correlates array JMh.

Returns *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Return type numpy.ndarray

Notes

- *LCD* in *CAM16-LCD* stands for *Large Colour Differences*.

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LLW+17]

colour.CAM16LCD_to_JMh_CAM16

colour.**CAM16LCD_to_JMh_CAM16**(*Jpapbp*, ***, *coefficients=Coefficients_UCS_Luo2006(K_L=0.77, c_1=0.007, c_2=0.0053)*)

Convert from *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

Parameters **Jpapbp** – *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Returns *CAM16* correlates array JMh.

Return type numpy.ndarray

Notes

- *LCD* in *CAM16-LCD* stands for *Large Colour Differences*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LLW+17]

colour.JMh_CAM16_to_CAM16SCD

colour.**JMh_CAM16_to_CAM16SCD**(*JMh, *, coefficients=Coefficients_UCS_Luo2006(K_L=1.24, c_1=0.007, c_2=0.0363)*)

Convert from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Parameters **JMh** – *CAM16* correlates array JMh.

Returns *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Return type numpy.ndarray

Notes

- *SCD* in *CAM16-SCD* stands for *Small Colour Differences*.

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LLW+17]

colour.CAM16SCD_to_JMh_CAM16

colour.**CAM16SCD_to_JMh_CAM16**(*Jpapbp, *, coefficients=Coefficients_UCS_Luo2006(K_L=1.24, c_1=0.007, c_2=0.0363)*)

Convert from *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

Parameters **Jpapbp** – *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Returns *CAM16* correlates array JMh.

Return type numpy.ndarray

Notes

- *SCD* in *CAM16-SCD* stands for *Small Colour Differences*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LLW+17]

colour.JMh_CAM16_to_CAM16UCS

colour.**JMh_CAM16_to_CAM16UCS**(*JMh*, ***, *coefficients=Coefficients_UCS_Luo2006(K_L=1.0, c_1=0.007, c_2=0.0228)*)

Convert from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Parameters **JMh** – *CAM16* correlates array JMh.

Returns *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Return type numpy.ndarray

Notes

- *UCS* in *CAM16-UCS* stands for *Uniform Colour Colourspace*.

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LLW+17]

colour.CAM16UCS_to_JMh_CAM16

colour.**CAM16UCS_to_JMh_CAM16**(*Jpapbp*, ***, *coefficients=Coefficients_UCS_Luo2006(K_L=1.0, c_1=0.007, c_2=0.0228)*)

Convert from *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

Parameters **Jpapbp** – *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Returns *CAM16* correlates array JMh.

Return type numpy.ndarray

Notes

- *UCS* in *CAM16-UCS* stands for *Uniform Colour Colourspace*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LLW+17]

colour.XYZ_to_CAM16LCD

colour.**XYZ_to_CAM16LCD**(*XYZ: ArrayLike*, ***, *coefficients: ArrayLike = Coefficients_UCS_Luo2006(K_L=0.77, c_1=0.007, c_2=0.0053)*, ***kwargs: Any*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **kwargs** (Any) – {colour.XYZ_to_CAM16()}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.
- **coefficients** (ArrayLike) –

Returns *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Return type numpy.ndarray

> **Warning:** The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *LCD* in *CAM16-LCD* stands for *Large Colour Differences*.

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LLW+17]

colour.CAM16LCD_to_XYZ

colour.**CAM16LCD_to_XYZ**(*Jpapbp: ArrayLike, *, coefficients: ArrayLike = Coefficients_UCS_Luo2006(K_L=0.77, c_1=0.007, c_2=0.0053), **kwargs: Any*) → NDArrayFloat

Convert from *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.

Parameters

- **Jpapbp** (ArrayLike) – *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.
- **kwargs** (Any) – {colour.CAM16_to_XYZ()}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.
- **coefficients** (ArrayLike) –

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

> **Warning:** The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *LCD* in *CAM16-LCD* stands for *Large Colour Differences*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[LLW+17]

colour.XYZ_to_CAM16SCD

`colour.XYZ_to_CAM16SCD`(*XYZ: ArrayLike*, ***, *coefficients: ArrayLike = Coefficients_UCS_Luo2006(K_L=1.24, c_1=0.007, c_2=0.0363)*, ***kwargs: Any*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **kwargs** (Any) – {colour.XYZ_to_CAM16()}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.
- **coefficients** (ArrayLike) –

Returns *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Return type numpy.ndarray

> **Warning:** The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *SCD* in *CAM16-SCD* stands for *Small Colour Differences*.

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LLW+17]

colour.CAM16SCD_to_XYZ

`colour.CAM16SCD_to_XYZ`(*Jpapbp: ArrayLike*, ***, *coefficients: ArrayLike = Coefficients_UCS_Luo2006(K_L=1.24, c_1=0.007, c_2=0.0363)*, ***kwargs: Any*) → NDArrayFloat

Convert from *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.

Parameters

- **Jpapbp** (`ArrayLike`) – *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.
- **kwargs** (`Any`) – {`colour.CAM16_to_XYZ()`}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.
- **coefficients** (`ArrayLike`) –

Returns *CIE XYZ* tristimulus values.

Return type `numpy.ndarray`

> **Warning:** The `XYZ_w` parameter for `colour.XYZ_to_CAM16()` definition must be given in the same domain-range scale than the `XYZ` parameter.

Notes

- *SCD* in *CAM16-SCD* stands for *Small Colour Differences*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[LLW+17]

colour.XYZ_to_CAM16UCS

`colour.XYZ_to_CAM16UCS`(*XYZ: ArrayLike*, ***, *coefficients: ArrayLike = Coefficients_UCS_Luo2006(K_L=1.0, c_1=0.007, c_2=0.0228)*, ***kwargs: Any*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **kwargs** (Any) – {`colour.XYZ_to_CAM16()`}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.
- **coefficients** (ArrayLike) –

Returns *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Return type numpy.ndarray

Warning: The XYZ_w parameter for `colour.XYZ_to_CAM16()` definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *UCS* in *CAM16-UCS* stands for *Uniform Colour Space*.

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

References

[LLW+17]

colour.CAM16UCS_to_XYZ

`colour.CAM16UCS_to_XYZ`(*Jpapbp: ArrayLike*, ***, *coefficients: ArrayLike = Coefficients_UCS_Luo2006(K_L=1.0, c_1=0.007, c_2=0.0228)*, ***kwargs: Any*) → NDArrayFloat

Convert from *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array to *CIE XYZ* tristimulus values.

Parameters

- **Jpapbp** (ArrayLike) – *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.
- **kwargs** (Any) – {`colour.CAM16_to_XYZ()`}, See the documentation of the previously listed definition. The default viewing conditions are that of *IEC 61966-2-1:1999*, i.e. *sRGB* 64 Lux ambient illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.

- **coefficients** (ArrayLike) –

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

> **Warning:** The XYZ_w parameter for colour.XYZ_to_CAM16() definition must be given in the same domain-range scale than the XYZ parameter.

Notes

- *UCS* in *CAM16-UCS* stands for *Uniform Colour Space*.

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp : [0, 100] ap : [-100, 100] bp : [-100, 100]	Jp : [0, 1] ap : [-1, 1] bp : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[LLW+17]

IC_AC_B Colourspace

colour

XYZ_to_ICaCb(XYZ)	Convert from *CIE XYZ* tristimulus values to IC_AC_B colourspace.
ICaCb_to_XYZ(ICaCb)	Convert from IC_AC_B tristimulus values to *CIE XYZ* colourspace.

colour.XYZ_to_ICaCb

colour.**XYZ_to_ICaCb**(*XYZ: ArrayLike*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to IC_AC_B colourspace.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.

Returns IC_AC_B colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
ICaCb	I : [0, 1] Ca : [-1, 1] Cb : [-1, 1]	I : [0, 1] Ca: [-1, 1] Cb: [-1, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Standard Illuminant D Series D65*.

References

[Frohlich17]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_ICaCb(XYZ)
array([ 0.06875297,  0.05753352,  0.02081548])
```

colour.ICaCb_to_XYZ

colour.**ICaCb_to_XYZ**(*ICaCb: ArrayLike*) → NDArrayFloat

Convert from IC_AC_B tristimulus values to *CIE XYZ* colourspace.

Parameters **ICaCb** (ArrayLike) – IC_AC_B tristimulus values.

Returns *CIE XYZ* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
ICaCb	I : [0, 1] Ca : [-1, 1] Cb : [-1, 1]	I : [0, 1] Ca: [-1, 1] Cb: [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Frohlich17]

Examples

```
>>> XYZ = np.array([0.06875297, 0.05753352, 0.02081548])
>>> ICaCb_to_XYZ(XYZ)
array([ 0.20654008,  0.12197225,  0.05136951])
```

$I_G P_G T_G$ Colourspace

colour

XYZ_to_IgPgTg(XYZ)	Convert from *CIE XYZ* tristimulus values to $I_G P_G T_G$ colourspace.
IgPgTg_to_XYZ(IgPgTg)	Convert from $I_G P_G T_G$ colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_IgPgTg

colour.**XYZ_to_IgPgTg**(*XYZ: ArrayLike*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to $I_G P_G T_G$ colourspace.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.

Returns $I_G P_G T_G$ colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
IgPgTg	IG : [0, 1] PG : [-1, 1] TG : [-1, 1]	IG : [0, 1] PG : [-1, 1] TG : [-1, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Standard Illuminant D Series D65*.

References

[HF20]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_IgPgTg(XYZ)
array([ 0.4242125...,  0.1863249...,  0.1068922...])
```

colour.IgPgTg_to_XYZ

colour.**IgPgTg_to_XYZ**(*IgPgTg: ArrayLike*) → NDArrayFloat

Convert from $I_G P_G T_G$ colourspace to *CIE XYZ* tristimulus values.

Parameters **IgPgTg** (ArrayLike) – $I_G P_G T_G$ colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
IgPgTg	IG : [0, 1] PG : [-1, 1] TG : [-1, 1]	IG : [0, 1] PG : [-1, 1] TG : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[HF20]

Examples

```
>>> IgPgTg = np.array([0.42421258, 0.18632491, 0.10689223])
>>> IgPgTg_to_XYZ(IgPgTg)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

IPT Colourspace

colour

XYZ_to_IPT(XYZ)	Convert from *CIE XYZ* tristimulus values to *IPT* colourspace.
IPT_to_XYZ(IPT)	Convert from *IPT* colourspace to *CIE XYZ* tristimulus values.
IPT_hue_angle(IPT)	Compute the hue angle in degrees from *IPT* colourspace.

colour.XYZ_to_IPT

`colour.XYZ_to_IPT(XYZ: ArrayLike) → NDArrayFloat`

Convert from *CIE XYZ* tristimulus values to *IPT* colourspace.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.

Returns *IPT* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
IPT	I : [0, 1] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Standard Illuminant D Series D65*.

References

[Fai13g]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_IPT(XYZ)
array([ 0.3842619...,  0.3848730...,  0.1888683...])
```

colour.IPT_to_XYZ

`colour.IPT_to_XYZ(IPT: ArrayLike) → NDArrayFloat`

Convert from *IPT* colourspace to *CIE XYZ* tristimulus values.

Parameters **IPT** (ArrayLike) – *IPT* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
IPT	I : [0, 1] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Fai13g]

Examples

```
>>> IPT = np.array([0.38426191, 0.38487306, 0.18886838])
>>> IPT_to_XYZ(IPT)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.IPT_hue_angle

colour.**IPT_hue_angle**(*IPT: ArrayLike*) → NDArrayFloat

Compute the hue angle in degrees from *IPT* colourspace.

Parameters **IPT** (ArrayLike) – *IPT* colourspace array.

Returns Hue angle in degrees.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
IPT	I : [0, 1] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

Range	Scale - Reference	Scale - 1
hue	[0, 360]	[0, 1]

References

[Fai13g]

Examples

```
>>> IPT = np.array([0.96907232, 1, 1.12179215])
>>> IPT_hue_angle(IPT)
48.2852074...
```

Ragoo and Farup (2021) Optimised IPT Colourspace

colour

XYZ_to_IPT_Ragoo2021(XYZ)	Convert from *CIE XYZ* tristimulus values to *Ragoo and Farup (2021) Optimised IPT* colourspace.
IPT_Ragoo2021_to_XYZ(IPT)	Convert from *Ragoo and Farup (2021) Optimised IPT* colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_IPT_Ragoo2021

colour.**XYZ_to_IPT_Ragoo2021**(*XYZ: ArrayLike*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Ragoo and Farup (2021) Optimised IPT* colourspace.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.

Returns *Ragoo and Farup (2021) Optimised IPT* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
IPT	I : [0, 1] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Standard Illuminant D Series D65*.

References

[RF21]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_IPT_Ragoo2021(XYZ)
array([ 0.4224824...,  0.2910514...,  0.2041066...])
```

colour.IPT_Ragoo2021_to_XYZ

colour.**IPT_Ragoo2021_to_XYZ**(*IPT: ArrayLike*) → NDArrayFloat

Convert from *Ragoo and Farup (2021) Optimised IPT* colourspace to *CIE XYZ* tristimulus values.

Parameters **IPT** (ArrayLike) – *Ragoo and Farup (2021) Optimised IPT* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
IPT	I : [0, 1] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[RF21]

Examples

```
>>> IPT = np.array([0.42248243, 0.2910514, 0.20410663])
>>> IPT_Ragoo2021_to_XYZ(IPT)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

hdr-CIELAB Colourspace

colour

XYZ_to_hdr_CIELab(XYZ[, illuminant, Y_s, ...])	Convert from *CIE XYZ* tristimulus values to *hdr-CIELAB* colourspace.
hdr_CIELab_to_XYZ(Lab_hdr[, illuminant, ...])	Convert 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: ArrayLike, illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65'], Y_s: ArrayLike = 0.2, Y_abs: ArrayLike = 100, method: Union[Literal['Fairchild 2011', 'Fairchild 2010'], str] = 'Fairchild 2011'*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *hdr-CIELAB* colourspace.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y_s** (ArrayLike) – Relative luminance Y_s of the surround.
- **Y_abs** (ArrayLike) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2.
- **method** (Union[Literal['Fairchild 2011', 'Fairchild 2010'], str]) – Computation method.

Returns *hdr-CIELAB* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab_hdr	L_hdr : [0, 100] a_hdr : [-100, 100] b_hdr : [-100, 100]	L_hdr : [0, 1] a_hdr : [-1, 1] b_hdr : [-1, 1]

- Conversion to polar coordinates to compute the *chroma* C_{hdr} and *hue* h_{hdr} correlates can be safely performed with `colour.Lab_to_LCHab()` definition.
- Conversion to cartesian coordinates from the *Lightness* L_{hdr}, *chroma* C_{hdr} and *hue* h_{hdr} correlates can be safely performed with `colour.LCHab_to_Lab()` definition.

References

[FW10], [FC11]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_hdr_CIELab(XYZ)
array([ 51.8700206...,  60.4763385...,  32.1455191...])
>>> XYZ_to_hdr_CIELab(XYZ, method="Fairchild 2010")
array([  31.9962111...,  128.0076303...,   48.7695230...])
```

colour.hdr_CIELab_to_XYZ

`colour.hdr_CIELab_to_XYZ`(*Lab_hdr: ArrayLike, illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65'], Y_s: ArrayLike = 0.2, Y_abs: ArrayLike = 100, method: Union[Literal['Fairchild 2011', 'Fairchild 2010'], str] = 'Fairchild 2011'*) → NDArrayFloat

Convert from *hdr-CIELAB* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Lab_hdr** (ArrayLike) – *hdr-CIELAB* colourspace array.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y_s** (ArrayLike) – Relative luminance Y_s of the surround.
- **Y_abs** (ArrayLike) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2.
- **method** (Union[Literal['Fairchild 2011', 'Fairchild 2010'], str]) – Computation method.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_hdr	L_hdr : [0, 100] a_hdr : [-100, 100] b_hdr : [-100, 100]	L_hdr : [0, 1] a_hdr : [-1, 1] b_hdr : [-1, 1]
illuminant	[0, 1]	[0, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[FW10], [FC11]

Examples

```
>>> Lab_hdr = np.array([51.87002062, 60.4763385, 32.14551912])
>>> hdr_CIELab_to_XYZ(Lab_hdr)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
>>> Lab_hdr = np.array([31.99621114, 128.00763036, 48.76952309])
>>> hdr_CIELab_to_XYZ(Lab_hdr, method="Fairchild 2010")
...
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.HDR_CIELAB_METHODS

colour.**HDR_CIELAB_METHODS = ('Fairchild 2010', 'Fairchild 2011')**
Supported *hdr-CIELAB* colourspace computation methods.

References

[FW10], [FC11]

hdr-IPT Colourspace

colour

XYZ_to_hdr_IPT(XYZ[, Y_s, Y_abs, method])	Convert from *CIE XYZ* tristimulus values to *hdr-IPT* colourspace.
hdr_IPT_to_XYZ(IPT_hdr[, Y_s, Y_abs, method])	Convert 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: ArrayLike, Y_s: ArrayLike = 0.2, Y_abs: ArrayLike = 100, method: Union[Literal['Fairchild 2011', 'Fairchild 2010'], str] = 'Fairchild 2011'*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *hdr-IPT* colourspace.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **Y_s** (ArrayLike) – Relative luminance Y_s of the surround.
- **Y_abs** (ArrayLike) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2.
- **method** (Union[Literal['Fairchild 2011', 'Fairchild 2010'], str]) – Computation method.

Returns *hdr-IPT* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
IPT_hdr	I_hdr : [0, 100] P_hdr : [-100, 100] T_hdr : [-100, 100]	I_hdr : [0, 1] P_hdr : [-1, 1] T_hdr : [-1, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Standard Illuminant D Series D65*.

References

[FW10], [FC11]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_hdr_IPT(XYZ)
array([ 48.3937634...,  42.4499020...,  22.0195403...])
>>> XYZ_to_hdr_IPT(XYZ, method="Fairchild 2010")
array([ 30.0287314...,  83.9384506...,  34.9028738...])
```

colour.hdr_IPT_to_XYZ

colour.**hdr_IPT_to_XYZ**(*IPT_hdr: ArrayLike, Y_s: ArrayLike = 0.2, Y_abs: ArrayLike = 100, method: Union[Literal['Fairchild 2011', 'Fairchild 2010'], str] = 'Fairchild 2011'*) → NDArrayFloat

Convert from *hdr-IPT* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **IPT_hdr** (ArrayLike) – *hdr-IPT* colourspace array.
- **Y_s** (ArrayLike) – Relative luminance Y_s of the surround.
- **Y_abs** (ArrayLike) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2.
- **method** (Union[Literal['Fairchild 2011', 'Fairchild 2010'], str]) – Computation method.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
IPT_hdr	I_hdr : [0, 100] P_hdr : [-100, 100] T_hdr : [-100, 100]	I_hdr : [0, 1] P_hdr : [-1, 1] T_hdr : [-1, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[FW10], [FC11]

Examples

```
>>> IPT_hdr = np.array([48.39376346, 42.44990202, 22.01954033])
>>> hdr_IPT_to_XYZ(IPT_hdr)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
>>> IPT_hdr = np.array([30.02873147, 83.93845061, 34.90287382])
>>> hdr_IPT_to_XYZ(IPT_hdr, method="Fairchild 2010")
...
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.HDR_IPT_METHODS

colour.**HDR_IPT_METHODS = ('Fairchild 2010', 'Fairchild 2011')**

Supported *hdr-IPT* colourspace computation methods.

References

[FW10], [FC11]

Oklab Colourspace

colour

XYZ_to_Oklab(XYZ)	Convert from *CIE XYZ* tristimulus values to *Oklab* colourspace.
Oklab_to_XYZ(Lab)	Convert from *Oklab* colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_Oklab

colour.**XYZ_to_Oklab**(*XYZ: ArrayLike*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Oklab* colourspace.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.

Returns *Oklab* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 1] a : [-1, 1] b : [-1, 1]	L : [0, 1] a : [-1, 1] b : [-1, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Standard Illuminant D Series D65*.

References

[Ott20]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Oklab(XYZ)
array([ 0.5163401...,  0.154695 ...,  0.0628957...])
```

colour.Oklab_to_XYZ

colour.**Oklab_to_XYZ**(*Lab: ArrayLike*) → NDArrayFloat

Convert from *Oklab* colourspace to *CIE XYZ* tristimulus values.

Parameters **Lab** (ArrayLike) – *Oklab* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 1] a : [-1, 1] b : [-1, 1]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Ott20]

Examples

```
>>> Lab = np.array([0.51634019, 0.15469500, 0.06289579])
>>> Oklab_to_XYZ(Lab)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

OSA UCS Colourspace

colour

XYZ_to_OSA_UCS(XYZ)	Convert from *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer* to *OSA UCS* colourspace.
OSA_UCS_to_XYZ(Ljg[, optimisation_kwargs])	Convert from *OSA UCS* colourspace to *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

colour.XYZ_to_OSA_UCS

colour.**XYZ_to_OSA_UCS**(*XYZ: ArrayLike*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer* to *OSA UCS* colourspace.

The lightness axis, *L* is usually in range [-9, 5] and centered around middle gray (Munsell N/6). The yellow-blue axis, *j* is usually in range [-15, 15]. The red-green axis, *g* is usually in range [-20, 15].

Parameters XYZ (ArrayLike) – *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Returns *OSA UCS* Ljg lightness, jaune (yellowness), and greenness.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Ljg	L : [-100, 100] j : [-100, 100] g : [-100, 100]	L : [-1, 1] j : [-1, 1] g : [-1, 1]

- *OSA UCS* uses the *CIE 1964 10 Degree Standard Observer*.

References

[CTS13], [Mor03]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> XYZ_to_OSA_UCS(XYZ)
array([-3.0049979...,  2.9971369..., -9.6678423...])
```

colour.OSA_UCS_to_XYZ

colour.**OSA_UCS_to_XYZ**(*Ljg: ArrayLike*, *optimisation_kwargs: dict | None = None*) → NDArrayFloat

Convert from *OSA UCS* colourspace to *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Parameters

- **Ljg** (ArrayLike) – *OSA UCS* Ljg lightness, jaune (yellowness), and greenness.
- **optimisation_kwargs** (dict | None) – Parameters for scipy.optimize.fmin() definition.

Returns *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Return type numpy.ndarray

> **Warning:** There is no analytical inverse transformation from *OSA UCS* to Ljg lightness, jaune (yellowness), and greenness to *CIE XYZ* tristimulus values, the current implementation relies on optimisation using scipy.optimize.fmin() definition and thus has reduced precision and poor performance.

Notes

Domain	Scale - Reference	Scale - 1
Ljg	L : [-100, 100] j : [-100, 100] g : [-100, 100]	L : [-1, 1] j : [-1, 1] g : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- *OSA UCS* uses the *CIE 1964 10 Degree Standard Observer*.

References

[CTS13], [Mor03]

Examples

```
>>> import numpy as np
>>> Ljg = np.array([-3.00499790, 2.99713697, -9.66784231])
>>> OSA_UCS_to_XYZ(Ljg)
array([ 20.6540240...,  12.1972369...,   5.1369372...])
```

ProLab Colourspace

colour

XYZ_to_ProLab(XYZ[, illuminant])	Convert from *CIE XYZ* tristimulus values to *ProLab* colourspace.
ProLab_to_XYZ(ProLab[, illuminant])	Convert from *ProLab* colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_ProLab

colour.**XYZ_to_ProLab**(*XYZ: ArrayLike*, *illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *ProLab* colourspace.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **illuminant** (ArrayLike) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *ProLab* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
ProLab	L : [0, 1] a : [-1, 1] b : [-1, 1]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[KSNN21]

Examples

```
>>> Lab = np.array([0.51634019, 0.15469500, 0.06289579])
>>> XYZ_to_ProLab(Lab)
array([  59.846628... ,  115.039635... ,   20.1251035...])
```

colour.ProLab_to_XYZ

`colour.`**`ProLab_to_XYZ`**(*ProLab: ArrayLike*, *illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']*) → NDArrayFloat

Convert from *ProLab* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **`ProLab`** (`ArrayLike`) – *ProLab* colourspace array.
- **`illuminant`** (`ArrayLike`) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 1] a : [-1, 1] b : [-1, 1]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[KSNN21]

Examples

```
>>> ProLab = np.array([59.8466286, 115.0396354, 20.12510352])
>>> ProLab_to_XYZ(ProLab)
array([ 0.5163401...,  0.154695 ...,  0.0628957...])
```

Yrg Colourspace - Kirk (2019)

colour

XYZ_to_Yrg(XYZ)	Convert from *CIE XYZ* tristimulus values to *Kirk (2019) Yrg* colourspace.
Yrg_to_XYZ(Yrg)	Convert from *Kirk (2019) Yrg* colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_Yrg

colour.**XYZ_to_Yrg**(*XYZ: ArrayLike*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *Kirk (2019) Yrg* colourspace.

Parameters **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values values.

Returns *Kirk (2019) Yrg* Yrg luminance, redness, and greenness.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Yrg	Y : [0, 1] r : [0, 1] g : [0, 1]	Y : [0, 1] r : [0, 1] g : [0, 1]

References

[Kir19]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Yrg(XYZ)
array([ 0.1313780...,  0.4903764...,  0.3777738...])
```

colour.Yrg_to_XYZ

colour.**Yrg_to_XYZ**(*Yrg: ArrayLike*) → NDArrayFloat

Convert from *Kirk (2019) Yrg* colourspace to *CIE XYZ* tristimulus values.

Parameters **Yrg** (ArrayLike) – *Kirk (2019) Yrg* Yrg luminance, redness, and greenness.

Returns *CIE XYZ* tristimulus values values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Yrg	Y : [0, 1] r : [0, 1] g : [0, 1]	Y : [0, 1] r : [0, 1] g : [0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[Kir19]

Examples

```
>>> import numpy as np
>>> Yrg = np.array([0.13137801, 0.49037645, 0.37777388])
>>> Yrg_to_XYZ(Yrg)
array([ 0.2065468...,  0.1219717...,  0.0513819...])
```

colour.models

LMS_to_Yrg(LMS)	Convert from *LMS* colourspace to *Kirk (2019) Yrg* colourspace.
Yrg_to_LMS(Yrg)	Convert from *Kirk (2019) Yrg* colourspace to *LMS* colourspace.

colour.models.LMS_to_Yrg

colour.models.**LMS_to_Yrg**(*LMS: ArrayLike*) → NDArrayFloat

Convert from *LMS* colourspace to *Kirk (2019) Yrg* colourspace.

Parameters **LMS** (ArrayLike) – *LMS* colourspace values.

Returns *Kirk (2019) Yrg* Yrg luminance, redness, and greenness.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
LMS	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Yrg	Y : [0, 1] r : [0, 1] g : [0, 1]	Y : [0, 1] r : [0, 1] g : [0, 1]

References

[Kir19]

Examples

```
>>> import numpy as np
>>> LMS = np.array([0.15639195, 0.06741689, 0.03281398])
>>> LMS_to_Yrg(LMS)
array([ 0.1313780...,  0.4903764...,  0.3777739...])
```

colour.models.Yrg_to_LMS

colour.models.**Yrg_to_LMS**(*Yrg: ArrayLike*) → NDArrayFloat

Convert from *Kirk (2019) Yrg* colourspace to *LMS* colourspace.

Parameters **Yrg** (ArrayLike) – *Kirk (2019) Yrg* Yrg luminance, redness, and greenness.

Returns *LMS* colourspace values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
Yrg	Y : [0, 1] r : [0, 1] g : [0, 1]	Y : [0, 1] r : [0, 1] g : [0, 1]

Range	Scale - Reference	Scale - 1
LMS	[0, 100]	[0, 1]

References

[Kir19]

Examples

```
>>> import numpy as np
>>> Yrg = np.array([0.13137801, 0.49037644, 0.37777391])
>>> Yrg_to_LMS(Yrg)
array([ 0.1563929...,  0.0674150...,  0.0328213...])
```

$Jzazbz$ Colourspace

colour

XYZ_to_Jzazbz(XYZ_D65[, constants])	Convert from *CIE XYZ* tristimulus values to $J_z a_z b_z$ colourspace.
Jzazbz_to_XYZ(Jzazbz[, constants])	Convert from $J_z a_z b_z$ colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_Jzazbz

colour.**XYZ_to_Jzazbz**(*XYZ_D65: ArrayLike*, *constants:* colour.utilities.data_structures.Structure = *CONSTANTS_JZAZBZ_SAFDAR2017*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to $J_z a_z b_z$ colourspace.

Parameters

- **XYZ_D65** (ArrayLike) – *CIE XYZ* tristimulus values under *CIE Standard Illuminant D Series D65*.
- **constants** (colour.utilities.data_structures.Structure) – $J_z a_z b_z$ colourspace constants.

Returns $J_z a_z b_z$ colourspace array where J_z is Lightness, a_z is redness-greenness and b_z is yellowness-blueness.

Return type numpy.ndarray

Warning: The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations. The effective domain of *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) is [0.0001, 10000].

Domain	Scale - Reference	Scale - 1
XYZ	UN	UN

Range	Scale - Reference	Scale - 1
Jzazbz	Jz : [0, 1] az : [-1, 1] bz : [-1, 1]	Jz : [0, 1] az : [-1, 1] bz : [-1, 1]

References

[SCKL17]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Jzazbz(XYZ)
array([ 0.0053504...,  0.0092430...,  0.0052600...])
```

colour.Jzazbz_to_XYZ

`colour.`**`Jzazbz_to_XYZ`**(*Jzazbz: ArrayLike*, *constants:* colour.utilities.data_structures.Structure = *CONSTANTS_JZAZBZ_SAFDAR2017*) → NDArrayFloat

Convert from $J_z a_z b_z$ colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Jzazbz** (ArrayLike) – $J_z a_z b_z$ colourspace array where J_z is Lightness, a_z is redness-greenness and b_z is yellowness-blueness.
- **constants** (colour.utilities.data_structures.Structure) – $J_z a_z b_z$ colourspace constants.

Returns *CIE XYZ* tristimulus values under *CIE Standard Illuminant D Series D65*.

Return type numpy.ndarray

> **Warning:** The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
Jzazbz	Jz : [0, 1] az : [-1, 1] bz : [-1, 1]	Jz : [0, 1] az : [-1, 1] bz : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	UN	UN

References

[SCKL17]

Examples

```
>>> Jzazbz = np.array([0.00535048, 0.00924302, 0.00526007])
>>> Jzazbz_to_XYZ(Jzazbz)
array([ 0.2065402...,  0.1219723...,  0.0513696...])
```

Ancillary Objects

colour.models

IZAZBZ_METHODS	Supported $I_z a_z b_z$ computation methods.
XYZ_to_Izazbz(XYZ_D65[, constants, method])	Convert from *CIE XYZ* tristimulus values to $I_z a_z b_z$ colourspace.
Izazbz_to_XYZ(Izazbz[, constants, method])	Convert from $I_z a_z b_z$ colourspace to *CIE XYZ* tristimulus values.

colour.models.IZAZBZ_METHODS

colour.models.**IZAZBZ_METHODS = ('Safdar 2017', 'Safdar 2021', 'ZCAM')**

Supported $I_z a_z b_z$ computation methods.

References

[SCKL17], [SHRL21]

colour.models.XYZ_to_Izazbz

colour.models.**XYZ_to_Izazbz**(*XYZ_D65: ArrayLike, constants: colour.utilities.data_structures.Structure | None = None, method: Union[Literal['Safdar 2017', 'Safdar 2021', 'ZCAM'], str] = 'Safdar 2017'*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to $I_z a_z b_z$ colourspace.

Parameters

- **XYZ_D65** (ArrayLike) – *CIE XYZ* tristimulus values under *CIE Standard Illuminant D Series D65*.
- **constants** (colour.utilities.data_structures.Structure | None) – $J_z a_z b_z$ colourspace constants.
- **method** (Union[Literal['Safdar 2017', 'Safdar 2021', 'ZCAM'], str]) – Computation method, *Safdar 2021* and *ZCAM* methods are equivalent.

Returns $I_z a_z b_z$ colourspace array where I_z is the achromatic response, a_z is redness-greenness and b_z is yellowness-blueness.

Return type numpy.ndarray

> **Warning:** The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations. The effective domain of *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) is [0.0001, 10000].

Domain	Scale - Reference	Scale - 1
XYZ	UN	UN

Range	Scale - Reference	Scale - 1
Izazbz	Iz : [0, 1] az : [-1, 1] bz : [-1, 1]	Iz : [0, 1] az : [-1, 1] bz : [-1, 1]

References

[SCKL17], [SHRL21]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Izazbz(XYZ)
array([ 0.0120779...,  0.0092430...,  0.0052600...])
```

colour.models.Izazbz_to_XYZ

colour.models.**Izazbz_to_XYZ**(*Izazbz: ArrayLike, constants:* colour.utilities.data_structures.Structure | *None = None, method: Union[Literal['Safdar 2017', 'Safdar 2021', 'ZCAM'], str] = 'Safdar 2017'*) → NDArrayFloat

Convert from $I_z a_z b_z$ colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Izazbz** (ArrayLike) – $I_z a_z b_z$ colourspace array where I_z is the achromatic response, a_z is redness-greenness and b_z is yellowness-blueness.
- **constants** (colour.utilities.data_structures.Structure | None) – $J_z a_z b_z$ colourspace constants.
- **method** (Union[Literal['Safdar 2017', 'Safdar 2021', 'ZCAM'], str]) – Computation method, *Safdar 2021* and *ZCAM* methods are equivalent.

Returns *CIE XYZ* tristimulus values under *CIE Standard Illuminant D Series D65*.

Return type numpy.ndarray

> **Warning:** The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
Izazbz	Iz : [0, 1] az : [-1, 1] bz : [-1, 1]	Iz : [0, 1] az : [-1, 1] bz : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	UN	UN

References

[SCKL17], [SHRL21]

Examples

```
>>> Izazbz = np.array([0.01207793, 0.00924302, 0.00526007])
>>> Izazbz_to_XYZ(Izazbz)
array([ 0.2065401...,  0.1219723...,  0.0513696...])
```

RGB Colourspace and Transformations

colour

XYZ_to_RGB(XYZ, colourspace[, illuminant, ...])	Convert from *CIE XYZ* tristimulus values to *RGB* colourspace array.
RGB_to_XYZ(RGB, colourspace[, illuminant, ...])	Convert given *RGB* colourspace array to *CIE XYZ* tristimulus values.
RGB_to_RGB(RGB, input_colourspace, ...[, ...])	Convert given *RGB* colourspace array from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.
matrix_RGB_to_RGB(input_colourspace, ...[, ...])	Compute 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: ArrayLike, colourspace:* RGB_Colourspace | *str, illuminant: ArrayLike* | *None = None, chromatic_adaptation_transform: Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling']* | *str* | *None = 'CAT02', apply_cctf_encoding: bool = False, *args: Any, **kwargs: Any*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *RGB* colourspace array.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.

- **colourspace** (RGB_Colourspace | str) – Output *RGB* colourspace.
- **illuminant** (ArrayLike | None) – *CIE xy* chromaticity coordinates or *CIE xyY* colourspace array of the *illuminant* for the input *CIE XYZ* tristimulus values.
- **chromatic_adaptation_transform** (Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'] | str | None) – *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **apply_cctf_encoding** (bool) – Apply the *RGB* colourspace encoding colour component transfer function / opto-electronic transfer function.
- **args** (Any) – Arguments for deprecation management.
- **kwargs** (Any) – Keywords arguments for deprecation management.

Returns *RGB* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant_XYZ	[0, 1]	[0, 1]
illuminant_RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB
>>> XYZ = np.array([0.21638819, 0.12570000, 0.03847493])
>>> illuminant = np.array([0.34570, 0.35850])
>>> XYZ_to_RGB(XYZ, RGB_COLOURSPACE_sRGB, illuminant, "Bradford")
...
array([ 0.4559528...,  0.0304078...,  0.0408731...])
>>> XYZ_to_RGB(XYZ, "sRGB", illuminant, "Bradford")
...
array([ 0.4559528...,  0.0304078...,  0.0408731...])
```

colour.RGB_to_XYZ

colour.**RGB_to_XYZ**(*RGB: ArrayLike, colourspace:* RGB_Colourspace | *str, illuminant: ArrayLike* | *None = None, chromatic_adaptation_transform: Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling']* | *str* | *None = 'CAT02', apply_cctf_decoding: bool = False, *args, **kwargs*) → NDArrayFloat

Convert given *RGB* colourspace array to *CIE XYZ* tristimulus values.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array.

- **colourspace** (RGB_Colourspace | str) – Input *RGB* colourspace.
- **illuminant** (ArrayLike | None) – *CIE xy* chromaticity coordinates or *CIE xyY* colourspace array of the *illuminant* for the output *CIE XYZ* tristimulus values.
- **chromatic_adaptation_transform** (Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'] | str | None) – *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **apply_cctf_decoding** (bool) – Apply the *RGB* colourspace decoding colour component transfer function / opto-electronic transfer function.
- **args** – Arguments for deprecation management.
- **kwargs** – Keywords arguments for deprecation management.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]
illuminant_XYZ	[0, 1]	[0, 1]
illuminant_RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB
>>> RGB = np.array([0.45595571, 0.03039702, 0.04087245])
>>> illuminant = np.array([0.34570, 0.35850])
>>> RGB_to_XYZ(RGB, RGB_COLOURSPACE_sRGB, illuminant, "Bradford")
...
array([ 0.2163881...,  0.1257    ,  0.0384749...])
>>> RGB_to_XYZ(RGB, "sRGB", illuminant, "Bradford")
...
array([ 0.2163881...,  0.1257    ,  0.0384749...])
```

colour.RGB_to_RGB

colour.**RGB_to_RGB**(*RGB: ArrayLike, input_colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace | str, output_colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace | str, chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02', apply_cctf_decoding: bool = False, apply_cctf_encoding: bool = False, **kwargs: Any*) → NDArrayFloat

Convert given *RGB* colourspace array from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array.
- **input_colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace | str) – *RGB* input colourspace.
- **output_colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace | str) – *RGB* output colourspace.
- **chromatic_adaptation_transform** (Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]) – *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **apply_cctf_decoding** (bool) – Apply the input colourspace decoding colour component transfer function / electro-optical transfer function.
- **apply_cctf_encoding** (bool) – Apply the output colourspace encoding colour component transfer function / opto-electronic transfer function.
- **kwargs** (Any) – Keywords arguments for the colour component transfer functions.

Returns *RGB* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> from colour.models import (
...     RGB_COLOURSPACE_sRGB,
...     RGB_COLOURSPACE_PROPHOTO_RGB,
... )
>>> RGB = np.array([0.45595571, 0.03039702, 0.04087245])
>>> RGB_to_RGB(RGB, RGB_COLOURSPACE_sRGB, RGB_COLOURSPACE_PROPHOTO_RGB)
...
array([ 0.2568891...,  0.0721446...,  0.0465553...])
>>> RGB_to_RGB(RGB, "sRGB", "ProPhoto RGB")
...
array([ 0.2568891...,  0.0721446...,  0.0465553...])
```

colour.matrix_RGB_to_RGB

colour.**matrix_RGB_to_RGB**(*input_colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace | str*, *output_colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace | str*, *chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02'*) → NDArrayFloat

Compute the matrix M converting from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.

Parameters

- **input_colourspace** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace | str`) – *RGB* input colourspace.
- **output_colourspace** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace | str`) – *RGB* output colourspace.
- **chromatic_adaptation_transform** (`Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]`) – *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.

Returns Conversion matrix M.

Return type `numpy.ndarray`

Examples

```
>>> from colour.models import (
...     RGB_COLOURSPACE_sRGB,
...     RGB_COLOURSPACE_PROPHOTO_RGB,
... )
>>> matrix_RGB_to_RGB(RGB_COLOURSPACE_sRGB, RGB_COLOURSPACE_PROPHOTO_RGB)
...
array([[ 0.5288241...,  0.3340609...,  0.1373616...],
       [ 0.0975294...,  0.8790074...,  0.0233981...],
       [ 0.0163599...,  0.1066124...,  0.8772485...]])
>>> matrix_RGB_to_RGB("sRGB", "ProPhoto RGB")
...
array([[ 0.5288241...,  0.3340609...,  0.1373616...],
       [ 0.0975294...,  0.8790074...,  0.0233981...],
       [ 0.0163599...,  0.1066124...,  0.8772485...]])
```

Ancillary Objects

colour

`XYZ_to_sRGB`(XYZ[, illuminant, ...])	Convert from *CIE XYZ* tristimulus values to *sRGB* colourspace.
`sRGB_to_XYZ`(RGB[, illuminant, ...])	Convert from *sRGB* colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_sRGB

colour.**XYZ_to_sRGB**(*XYZ: ArrayLike, illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65'], chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02', apply_cctf_encoding: bool = True*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to *sRGB* colourspace.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **illuminant** (ArrayLike) – Source illuminant chromaticity coordinates.
- **chromatic_adaptation_transform** (Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]) – *Chromatic adaptation* transform.
- **apply_cctf_encoding** (bool) – Whether to apply the *sRGB* encoding colour component transfer function / inverse electro-optical transfer function.

Returns *sRGB* colour array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_sRGB(XYZ)
array([ 0.7057393...,  0.1924826...,  0.2235416...])
```

colour.sRGB_to_XYZ

colour.**sRGB_to_XYZ**(*RGB: ArrayLike, illuminant: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65'], chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02', apply_cctf_decoding: bool = True*) → NDArrayFloat

Convert from *sRGB* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **RGB** (ArrayLike) – *sRGB* colourspace array.
- **illuminant** (ArrayLike) – Source illuminant chromaticity coordinates.

- **chromatic_adaptation_transform** (`Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]`) – *Chromatic adaptation* transform.
- **apply_cctf_decoding** (`bool`) – Whether to apply the *sRGB* decoding colour component transfer function / electro-optical transfer function.

Returns *CIE XYZ* tristimulus values.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> RGB = np.array([0.70573936, 0.19248266, 0.22354169])
>>> sRGB_to_XYZ(RGB)
array([ 0.2065429...,  0.1219794...,  0.0513714...])
```

RGB Colourspace Derivation

`colour`

`normalised_primary_matrix`(primaries, whitepoint)	Compute the *Normalised Primary Matrix* (NPM) converting a *RGB* colourspace array to *CIE XYZ* tristimulus values using given *primaries* and *whitepoint* xy chromaticity coordinates.
`chromatically_adapted_primaries`(primaries, ...)	Chromatically adapt given *primaries* xy chromaticity coordinates from test `whitepoint_t` to reference `whitepoint_r`.
`primaries_whitepoint`(npm)	Compute the *primaries* and *whitepoint* xy chromaticity coordinates using given *Normalised Primary Matrix* (NPM).
`RGB_luminance`(RGB, primaries, whitepoint)	Return the *luminance* Y of given *RGB* components from given *primaries* and *whitepoint*.
`RGB_luminance_equation`(primaries, whitepoint)	Return the *luminance equation* from given *primaries* and *whitepoint*.

colour.normalised_primary_matrix

colour.**normalised_primary_matrix**(*primaries: ArrayLike*, *whitepoint: ArrayLike*) → NDArrayFloat

Compute the *Normalised Primary Matrix* (NPM) converting a *RGB* colourspace array to *CIE XYZ* tristimulus values using given *primaries* and *whitepoint* xy chromaticity coordinates.

Parameters

- **primaries** (ArrayLike) – Primaries xy chromaticity coordinates.
- **whitepoint** (ArrayLike) – Illuminant / whitepoint xy chromaticity coordinates.

Returns *Normalised Primary Matrix* (NPM).

Return type numpy.ndarray

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: ArrayLike*, *whitepoint_t: ArrayLike*, *whitepoint_r: ArrayLike*, *chromatic_adaptation_transform: Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str] = 'CAT02'*) → NDArrayFloat

Chromatically adapt given *primaries* xy chromaticity coordinates from test whitepoint_t to reference whitepoint_r.

Parameters

- **primaries** (ArrayLike) – Primaries xy chromaticity coordinates.
- **whitepoint_t** (ArrayLike) – Test illuminant / whitepoint xy chromaticity coordinates.
- **whitepoint_r** (ArrayLike) – Reference illuminant / whitepoint xy chromaticity coordinates.
- **chromatic_adaptation_transform** (Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]) – *Chromatic adaptation* transform.

Returns Chromatically adapted primaries xy chromaticity coordinates.

Return type numpy.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: ArrayLike*) → Tuple[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ced8ffe50>, <sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ced8ff690>]

Compute the *primaries* and *whitepoint* xy chromaticity coordinates using given *Normalised Primary Matrix* (NPM).

Parameters **npm** (ArrayLike) – *Normalised Primary Matrix*.

Returns *Primaries* and *whitepoint* xy chromaticity coordinates.

Return type tuple

References

[Tri15]

Examples

```
>>> npm = np.array(
...     [
...         [9.52552396e-01, 0.00000000e00, 9.36786317e-05],
...         [3.43966450e-01, 7.28166097e-01, -7.21325464e-02],
...         [0.00000000e00, 0.00000000e00, 1.00882518e00],
...     ]
... )
>>> 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: ArrayLike*, *primaries: ArrayLike*, *whitepoint: ArrayLike*) → NDArrayFloat

Return the *luminance* Y of given *RGB* components from given *primaries* and *whitepoint*.

Parameters

- **RGB** (ArrayLike) – *RGB* chromaticity coordinate matrix.
- **primaries** (ArrayLike) – Primaries chromaticity coordinate matrix.
- **whitepoint** (ArrayLike) – Illuminant / whitepoint chromaticity coordinates.

Returns *Luminance* Y.

Return type numpy.ndarray

Examples

```
>>> RGB = np.array([0.21959402, 0.06986677, 0.04703877])
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> RGB_luminance(RGB, p, whitepoint)
0.1230145...
```

colour.RGB_luminance_equation

colour.**RGB_luminance_equation**(*primaries: ArrayLike*, *whitepoint: ArrayLike*) → str

Return the *luminance equation* from given *primaries* and *whitepoint*.

Parameters

- **primaries** (ArrayLike) – Primaries chromaticity coordinates.
- **whitepoint** (ArrayLike) – Illuminant / whitepoint chromaticity coordinates.

Returns *Luminance* equation.

Return type str

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> RGB_luminance_equation(p, whitepoint)
'Y = 0.3439664...(R) + 0.7281660...(G) + -0.0721325...(B)'
```

RGB Colourspaces

colour

RGB_Colourspace(name, primaries, whitepoint)	Implement support for the *RGB* colourspaces datasets from colour.models.datasets.aces_rgb, etc....

colour.RGB_Colourspace

class colour.**RGB_Colourspace**(*name: str, primaries: ArrayLike, whitepoint: ArrayLike, whitepoint_name: str | None = None, matrix_RGB_to_XYZ: ArrayLike | None = None, matrix_XYZ_to_RGB: ArrayLike | None = None, cctf_encoding: Callable | None = None, cctf_decoding: Callable | None = None, use_derived_matrix_RGB_to_XYZ: bool = False, use_derived_matrix_XYZ_to_RGB: bool = False*)

Bases: `object`

Implement support for the *RGB* colourspaces datasets from `colour.models.datasets.aces_rgb`, etc. . . .

Colour science literature related to *RGB* colourspaces and encodings defines their dataset using different degree of precision or rounding. While instances where a whitepoint is being defined with a value different from its canonical agreed one are rare, it is however very common to have normalised primary matrices rounded at different decimals. This can yield large discrepancies in computations.

Such an occurrence is the *V-Gamut* colourspace white paper, that defines the *V-Gamut* to *ITU-R BT.709* conversion matrix as follows:

```
[[ 1.806576 -0.695697 -0.110879]
 [-0.170090  1.305955 -0.135865]
 [-0.025206 -0.154468  1.179674]]
```

Computing this matrix using *ITU-R BT.709* colourspace derived normalised primary matrix yields:

```
[[ 1.8065736 -0.6956981 -0.1108786]
 [-0.1700890  1.3059548 -0.1358648]
 [-0.0252057 -0.1544678  1.1796737]]
```

The latter matrix is almost equals with the former, however performing the same computation using *IEC 61966-2-1:1999 sRGB* colourspace normalised primary matrix introduces severe disparities:

```
[[ 1.8063853 -0.6956147 -0.1109453]
 [-0.1699311  1.3058387 -0.1358616]
 [-0.0251630 -0.1544899  1.1797117]]
```

In order to provide support for both literature defined dataset and accurate computations enabling transformations without loss of precision, the `colour.RGB_Colourspace` class provides two sets of transformation matrices:

- Instantiation transformation matrices
- Derived transformation matrices

Upon instantiation, the `colour.RGB_Colourspace` class stores the given `matrix_RGB_to_XYZ` and `matrix_XYZ_to_RGB` arguments and also computes their derived counterpart using the `primaries` and `whitepoint` arguments.

Whether the initialisation or derived matrices are used in subsequent computations is dependent on the `colour.RGB_Colourspace.use_derived_matrix_RGB_to_XYZ` and `colour.RGB_Colourspace.use_derived_matrix_XYZ_to_RGB` attribute values.

Parameters

- **name** (`str`) – *RGB* colourspace name.
- **primaries** (`ArrayLike`) – *RGB* colourspace primaries.
- **whitepoint** (`ArrayLike`) – *RGB* colourspace whitepoint.
- **whitepoint_name** (`str | None`) – *RGB* colourspace whitepoint name.

- **matrix_RGB_to_XYZ** (ArrayLike | None) – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.
- **matrix_XYZ_to_RGB** (ArrayLike | None) – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.
- **cctf_encoding** (Callable | None) – Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF) that maps estimated tristimulus values in a scene to $R'G'B'$ video component signal value.
- **cctf_decoding** (Callable | None) – Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF) that maps an $R'G'B'$ video component signal value to tristimulus values at the display.
- **use_derived_matrix_RGB_to_XYZ** (bool) – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.
- **use_derived_matrix_XYZ_to_RGB** (bool) – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

Return type None

Attributes

- name
- primaries
- whitepoint
- whitepoint_name
- matrix_RGB_to_XYZ
- matrix_XYZ_to_RGB
- cctf_encoding
- cctf_decoding
- use_derived_matrix_RGB_to_XYZ
- use_derived_matrix_XYZ_to_RGB

Methods

- __init__
- __str__
- __repr__
- use_derived_transformation_matrices
- chromatically_adapt
- copy

Notes

- The normalised primary matrix defined by `colour.RGB_Colourspace.matrix_RGB_to_XYZ` property 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], [Panasonic14]

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> matrix_RGB_to_XYZ = np.identity(3)
>>> matrix_XYZ_to_RGB = np.identity(3)
>>> colourspace = RGB_Colourspace(
...     "RGB Colourspace",
...     p,
...     whitepoint,
...     "ACES",
...     matrix_RGB_to_XYZ,
...     matrix_XYZ_to_RGB,
... )
>>> colourspace.matrix_RGB_to_XYZ
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.matrix_XYZ_to_RGB
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_transformation_matrices(True)
>>> colourspace.matrix_RGB_to_XYZ
array([[  9.5255239...e-01,   0.0000000...e+00,   9.3678631...e-05],
       [  3.4396645...e-01,   7.2816609...e-01,  -7.2132546...e-02],
       [  0.0000000...e+00,   0.0000000...e+00,   1.0088251...e+00]])
>>> colourspace.matrix_XYZ_to_RGB
array([[  1.0498110...e+00,   0.0000000...e+00,  -9.7484540...e-05],
       [ -4.9590302...e-01,   1.3733130...e+00,   9.8240036...e-02],
       [  0.0000000...e+00,   0.0000000...e+00,   9.9125201...e-01]])
>>> colourspace.use_derived_matrix_RGB_to_XYZ = False
>>> colourspace.matrix_RGB_to_XYZ
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_matrix_XYZ_to_RGB = False
>>> colourspace.matrix_XYZ_to_RGB
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

__init__(*name: str, primaries: ArrayLike, whitepoint: ArrayLike, whitepoint_name: str | None = None, matrix_RGB_to_XYZ: ArrayLike | None = None, matrix_XYZ_to_RGB: ArrayLike | None = None, cctf_encoding: Callable | None = None, cctf_decoding: Callable | None = None, use_derived_matrix_RGB_to_XYZ: bool = False, use_derived_matrix_XYZ_to_RGB: bool = False*) → None

Parameters

- **name** (`str`) –
- **primaries** (`ArrayLike`) –
- **whitepoint** (`ArrayLike`) –
- **whitepoint_name** (`str | None`) –
- **matrix_RGB_to_XYZ** (`ArrayLike | None`) –
- **matrix_XYZ_to_RGB** (`ArrayLike | None`) –
- **cctf_encoding** (`Callable | None`) –
- **cctf_decoding** (`Callable | None`) –
- **use_derived_matrix_RGB_to_XYZ** (`bool`) –
- **use_derived_matrix_XYZ_to_RGB** (`bool`) –

Return type None

property name: str

Getter and setter property for the name.

Parameters **value** – Value to set the name with.

Returns *RGB* colourspace name.

Return type `str`

property primaries: NDArrayFloat

Getter and setter property for the primaries.

Parameters **value** – Value to set the primaries with.

Returns *RGB* colourspace primaries.

Return type `numpy.ndarray`

property whitepoint: NDArrayFloat

Getter and setter property for the whitepoint.

Parameters **value** – Value to set the whitepoint with.

Returns *RGB* colourspace whitepoint.

Return type `numpy.ndarray`

property whitepoint_name: str | None

Getter and setter property for the whitepoint_name.

Parameters **value** – Value to set the whitepoint_name with.

Returns *RGB* colourspace whitepoint name.

Return type `None` or `str`

property matrix_RGB_to_XYZ: NDArrayFloat

Getter and setter property for the transformation matrix from colourspace to *CIE XYZ* tristimulus values.

Parameters **value** – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.

Returns Transformation matrix from colourspace to *CIE XYZ* tristimulus values.

Return type numpy.ndarray

__weakref__

list of weak references to the object (if defined)

property matrix_XYZ_to_RGB: NDArrayFloat

Getter and setter property for the transformation matrix from *CIE XYZ* tristimulus values to colourspace.

Parameters value – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.

Returns Transformation matrix from *CIE XYZ* tristimulus values to colourspace.

Return type numpy.ndarray

property cctf_encoding: Optional[Callable]

Getter and setter property for the encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF).

Parameters value – Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF).

Returns Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF).

Return type None or Callable

property cctf_decoding: Optional[Callable]

Getter and setter property for the decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF).

Parameters value – Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF).

Returns Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF).

Return type None or Callable

property use_derived_matrix_RGB_to_XYZ: bool

Getter and setter property for whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

Parameters value – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

Returns Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

Return type bool

property use_derived_matrix_XYZ_to_RGB: bool

Getter and setter property for Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

Parameters value – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

Returns Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

Return type bool

__str__() → str

Return a formatted string representation of the *RGB* colourspace.

Returns Formatted string representation.

Return type str

Examples

```
>>> p = np.array(
...     [0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700]
... )
>>> whitepoint = np.array([0.32168, 0.33767])
>>> matrix_RGB_to_XYZ = np.identity(3)
>>> matrix_XYZ_to_RGB = np.identity(3)
>>> cctf_encoding = lambda x: x
>>> cctf_decoding = lambda x: x
>>> print(
...     RGB_Colourspace(
...         "RGB Colourspace",
...         p,
...         whitepoint,
...         "ACES",
...         matrix_RGB_to_XYZ,
...         matrix_XYZ_to_RGB,
...         cctf_encoding,
...         cctf_decoding,
...     )
... )
...
RGB Colourspace
---------------

Primaries          : [[  7.34700000e-01   2.65300000e-01]
                      [  0.00000000e+00   1.00000000e+00]
                      [  1.00000000e-04  -7.70000000e-02]]
Whitepoint         : [ 0.32168  0.33767]
Whitepoint Name    : ACES
Encoding CCTF      : <function <lambda> at 0x...>
Decoding CCTF      : <function <lambda> at 0x...>
NPM                : [[ 1.  0.  0.]
                      [ 0.  1.  0.]
                      [ 0.  0.  1.]]
NPM -1             : [[ 1.  0.  0.]
                      [ 0.  1.  0.]
                      [ 0.  0.  1.]]
Derived NPM        : [[  9.5255239...e-01   0.0000000...e+00   9.3678631...e-05]
                      [  3.4396645...e-01   7.2816609...e-01  -7.2132546...e-02]
                      [  0.0000000...e+00   0.0000000...e+00   1.0088251...e+00]]
Derived NPM -1     : [[  1.0498110...e+00   0.0000000...e+00  -9.7484540...e-05]
                      [ -4.9590302...e-01   1.3733130...e+00   9.8240036...e-02]
                      [  0.0000000...e+00   0.0000000...e+00   9.9125201...e-01]]
Use Derived NPM    : False
Use Derived NPM -1 : False
```

__repr__() → str

Return an (almost) evaluable string representation of the *RGB* colourspace.

Returns (Almost) evaluable string representation.

Return type class`str`

Examples

```
>>> from colour.models import linear_function
>>> p = np.array(
...     [0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700]
... )
>>> whitepoint = np.array([0.32168, 0.33767])
>>> matrix_RGB_to_XYZ = np.identity(3)
>>> matrix_XYZ_to_RGB = np.identity(3)
>>> RGB_Colourspace(
...     "RGB Colourspace",
...     p,
...     whitepoint,
...     "ACES",
...     matrix_RGB_to_XYZ,
...     matrix_XYZ_to_RGB,
...     linear_function,
...     linear_function,
... )
...
RGB_Colourspace('RGB Colourspace',
                [[  7.34700000e-01,   2.65300000e-01],
                 [  0.00000000e+00,   1.00000000e+00],
                 [  1.00000000e-04,  -7.70000000e-02]],
                [ 0.32168,  0.33767],
                'ACES',
                [[ 1.,  0.,  0.],
                 [ 0.,  1.,  0.],
                 [ 0.,  0.,  1.]],
                [[ 1.,  0.,  0.],
                 [ 0.,  1.,  0.],
                 [ 0.,  0.,  1.]],
                linear_function,
                linear_function,
                False,
                False)
```

use_derived_transformation_matrices(*usage: bool = True*)

Enable or disables usage of both derived transformations matrices, the normalised primary matrix and its inverse in subsequent computations.

Parameters **usage** (bool) – Whether to use the derived transformations matrices.

chromatically_adapt(*whitepoint: ArrayLike, whitepoint_name: str | None = None, chromatic_adaptation_transform: Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str] = 'CAT02'*) → *colour.models.rgb.rgb_colourspace.RGB_Colourspace*

Chromatically adapt the *RGB* colourspace *primaries* xy chromaticity coordinates from *RGB* colourspace whitepoint to reference `whitepoint`.

Parameters

- **whitepoint** (ArrayLike) – Reference illuminant / whitepoint xy chromaticity coordinates.

- **whitepoint_name** (str | None) – Reference illuminant / whitepoint name.
- **chromatic_adaptation_transform** (Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]) – *Chromatic adaptation* transform.

Returns Chromatically adapted *RGB* colourspace.

Return type colour.RGB_Colourspace

Examples

```
>>> p = np.array(
...     [0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700]
... )
>>> w_t = np.array([0.32168, 0.33767])
>>> w_r = np.array([0.31270, 0.32900])
>>> colourspace = RGB_Colourspace("RGB Colourspace", p, w_t, "D65")
>>> print(colourspace.chromatically_adapt(w_r, "D50", "Bradford"))
...
RGB Colourspace - Chromatically Adapted to 'D50'
------------------------------------------------

Primaries          : [[ 0.73485524  0.26422533]
                      [-0.00617091  1.01131496]
                      [ 0.01596756 -0.0642355 ]]
Whitepoint         : [ 0.3127  0.329 ]
Whitepoint Name    : D50
Encoding CCTF      : None
Decoding CCTF      : None
NPM                : None
NPM -1             : None
Derived NPM        : [[ 0.93827985 -0.00445145  0.01662752]
                      [ 0.33736889  0.72952157 -0.06689046]
                      [ 0.00117395 -0.00371071  1.09159451]]
Derived NPM -1     : [[ 1.06349549  0.00640891 -0.01580679]
                      [-0.49207413  1.36822341  0.09133709]
                      [-0.00281646  0.00464417  0.91641857]]
Use Derived NPM    : True
Use Derived NPM -1 : True
```

copy() → *colour.models.rgb.rgb_colourspace.RGB_Colourspace*

Return a copy of the *RGB* colourspace.

Returns *RGB* colourspace copy.

Return type colour.RGB_Colourspace

RGB_COLOURSPACES	Aggregated *RGB* colourspaces.

colour.RGB_COLOURSPACES

```
colour.RGB_COLOURSPACES = LazyCanonicalMapping({'ACES2065-1': ..., 'ACEScc': ...,
'ACEScct': ..., 'ACEScg': ..., 'ACESproxy': ..., 'ARRI Wide Gamut 3': ..., 'ARRI Wide Gamut
4': ..., 'Adobe RGB (1998)': ..., 'Adobe Wide Gamut RGB': ..., 'Apple RGB': ..., 'Best
RGB': ..., 'Beta RGB': ..., 'Blackmagic Wide Gamut': ..., 'CIE RGB': ..., 'Cinema Gamut':
..., 'ColorMatch RGB': ..., 'DCDM XYZ': ..., 'DCI-P3': ..., 'DCI-P3-P': ..., 'DJI D-Gamut':
..., 'DRAGONcolor': ..., 'DRAGONcolor2': ..., 'DaVinci Wide Gamut': ..., 'Display P3':
..., 'Don RGB 4': ..., 'EBU Tech. 3213-E': ..., 'ECI RGB v2': ..., 'ERIMM RGB': ..., 'Ekta
Space PS 5': ..., 'F-Gamut': ..., 'FilmLight E-Gamut': ..., 'ITU-R BT.2020': ..., 'ITU-R
BT.470 - 525': ..., 'ITU-R BT.470 - 625': ..., 'ITU-R BT.709': ..., 'ITU-T H.273 - 22
Unspecified': ..., 'ITU-T H.273 - Generic Film': ..., 'Max RGB': ..., 'N-Gamut': ..., 'NTSC
(1953)': ..., 'NTSC (1987)': ..., 'P3-D65': ..., 'PLASA ANSI E1.54': ..., 'Pal/Secam': ...,
'ProPhoto RGB': ..., 'Protune Native': ..., 'REDWideGamutRGB': ..., 'REDcolor': ...,
'REDcolor2': ..., 'REDcolor3': ..., 'REDcolor4': ..., 'RIMM RGB': ..., 'ROMM RGB': ...,
'Russell RGB': ..., 'S-Gamut': ..., 'S-Gamut3': ..., 'S-Gamut3.Cine': ..., 'SMPTE 240M':
..., 'SMPTE C': ..., 'Sharp RGB': ..., 'V-Gamut': ..., 'Venice S-Gamut3': ..., 'Venice
S-Gamut3.Cine': ..., 'Xtreme RGB': ..., 'sRGB': ..., 'aces': ..., 'adobe1998': ...,
'prophoto': ..., 'ALEXA Wide Gamut': ...})
```

Aggregated *RGB* colourspaces.

Aliases:

- 'aces': RGB_COLOURSPACE_ACES2065_1.name
- 'adobe1998': RGB_COLOURSPACE_ADOBE_RGB1998.name
- 'prophoto': RGB_COLOURSPACE_PROPHOTO_RGB.name

colour.models

RGB_COLOURSPACE_ACES2065_1	*ACES2065-1* colourspace, base encoding, used for exchange of full fidelity images and archiving.
RGB_COLOURSPACE_ACESCC	*ACEScc* colourspace, a working space for color correctors, target for ASC-CDL values created on-set.
RGB_COLOURSPACE_ACESCCT	*ACEScct* colourspace, an alternative working space for colour correctors, intended to be transient and internal to software or hardware systems, and is specifically not intended for interchange or archiving.
RGB_COLOURSPACE_ACESPROXY	*ACESproxy* colourspace, a lightweight encoding for transmission over HD-SDI (or other production transmission schemes), onset look management.
RGB_COLOURSPACE_ACESCG	*ACEScg* colourspace, a working space for paint/compositor applications that don't support ACES2065-1 or ACEScc.
RGB_COLOURSPACE_ADOBE_RGB1998	*Adobe RGB (1998)* colourspace.
RGB_COLOURSPACE_ADOBE_WIDE_GAMUT_RGB	*Adobe Wide Gamut RGB* colourspace.
RGB_COLOURSPACE_ARRI_WIDE_GAMUT_3	*ARRI Wide Gamut 3* colourspace.
RGB_COLOURSPACE_ARRI_WIDE_GAMUT_4	*ARRI Wide Gamut 4* colourspace.
RGB_COLOURSPACE_APPLE_RGB	*Apple RGB* colourspace.
RGB_COLOURSPACE_BEST_RGB	*Best RGB* colourspace.
RGB_COLOURSPACE_BETA_RGB	*Beta RGB* colourspace.
RGB_COLOURSPACE_BLACKMAGIC_WIDE_GAMUT	*Blackmagic Wide Gamut* colourspace.
RGB_COLOURSPACE_BT470_525	*Recommendation ITU-R BT.470 - 525* colourspace.
RGB_COLOURSPACE_BT470_625	*Recommendation ITU-R BT.470 - 625* colourspace.

continues on next page

Table 2 – continued from previous page

RGB_COLOURSPACE_BT709	*Recommendation ITU-R BT.709* colourspace.
RGB_COLOURSPACE_BT2020	*RecommendationITU-R BT.2020* colourspace.
RGB_COLOURSPACE_CIE_RGB	*CIE RGB* colourspace.
RGB_COLOURSPACE_CINEMA_GAMUT	*Canon Cinema Gamut* colourspace.
RGB_COLOURSPACE_COLOR_MATCH_RGB	*ColorMatch RGB* colourspace.
RGB_COLOURSPACE_DAVINCI_WIDE_GAMUT	*DaVinci Wide Gamut* colourspace.
RGB_COLOURSPACE_DCDM_XYZ	*DCDM XYZ* colourspace.
RGB_COLOURSPACE_DCI_P3	*DCI-P3* colourspace.
RGB_COLOURSPACE_DCI_P3_P	*DCI-P3+* colourspace.
RGB_COLOURSPACE_DISPLAY_P3	*Display P3* colourspace.
RGB_COLOURSPACE_DON_RGB_4	*Don RGB 4* colourspace.
RGB_COLOURSPACE_EBU_3213_E	*EBU Tech.
RGB_COLOURSPACE_ECI_RGB_V2	*ECI RGB v2* colourspace.
RGB_COLOURSPACE_EKTA_SPACE_PS_5	*Ekta Space PS 5* colourspace.
RGB_COLOURSPACE_F_GAMUT	*Fujifilm F-Gamut* colourspace.
RGB_COLOURSPACE_H273_GENERIC_FILM	*Recommendation ITU-T H.273 Generic Film* (colour filters using Illuminant C) colourspace.
RGB_COLOURSPACE_H273_22_UNSPECIFIED	*Recommendation ITU-T H.273* row *22* colourspace as given in *Table 2 - Interpretation of colour primaries (ColourPrimaries) value*.
RGB_COLOURSPACE_PROTUNE_NATIVE	*Protune Native* colourspace.
RGB_COLOURSPACE_MAX_RGB	*Max RGB* colourspace.
RGB_COLOURSPACE_NTSC1953	*NTSC (1953)* colourspace.
RGB_COLOURSPACE_NTSC1987	*NTSC (1987)* colourspace.
RGB_COLOURSPACE_P3_D65	*P3-D65* colourspace.
RGB_COLOURSPACE_PAL_SECAM	*Pal/Secam* colourspace.
RGB_COLOURSPACE_RED_COLOR	*REDcolor* colourspace.
RGB_COLOURSPACE_RED_COLOR_2	*REDcolor2* colourspace.
RGB_COLOURSPACE_RED_COLOR_3	*REDcolor3* colourspace.
RGB_COLOURSPACE_RED_COLOR_4	*REDcolor4* colourspace.
RGB_COLOURSPACE_RED_WIDE_GAMUT_RGB	*REDWideGamutRGB* colourspace.
RGB_COLOURSPACE_DRAGON_COLOR	*DRAGONcolor* colourspace.
RGB_COLOURSPACE_DRAGON_COLOR_2	*DRAGONcolor2* colourspace.
RGB_COLOURSPACE_ROMM_RGB	*ROMM RGB* colourspace.
RGB_COLOURSPACE_RIMM_RGB	*RIMM RGB* colourspace.
RGB_COLOURSPACE_ERIMM_RGB	*ERIMM RGB* colourspace.
RGB_COLOURSPACE_PROPHOTO_RGB	*ProPhoto RGB* colourspace, an alias colourspace for *ROMM RGB*.
RGB_COLOURSPACE_PLASA_ANSI_E154	*PLASA ANSI E1.54* colourspace.
RGB_COLOURSPACE_RUSSELL_RGB	*Russell RGB* colourspace.
RGB_COLOURSPACE_SMPTE_240M	*SMPTE 240M* colourspace.
RGB_COLOURSPACE_SMPTE_C	*SMPTE C* colourspace.
RGB_COLOURSPACE_S_GAMUT	*S-Gamut* colourspace.
RGB_COLOURSPACE_S_GAMUT3	*S-Gamut3* colourspace.
RGB_COLOURSPACE_S_GAMUT3_CINE	*S-Gamut3.Cine* colourspace.
RGB_COLOURSPACE_VENICE_S_GAMUT3	*Venice S-Gamut3* colourspace.
RGB_COLOURSPACE_VENICE_S_GAMUT3_CINE	*Venice S-Gamut3.Cine* colourspace.
RGB_COLOURSPACE_sRGB	*Smits (1999)* colourspace.
RGB_COLOURSPACE_V_GAMUT	*Panasonic V-Gamut* colourspace.
RGB_COLOURSPACE_XTREME_RGB	*Xtreme RGB* colourspace.

colour.models.RGB_COLOURSPACE_ACES2065_1

colour.models.**RGB_COLOURSPACE_ACES2065_1 = RGB_Colourspace('ACES2065-1', [[7.34700000e-01, 2.65300000e-01], [0.00000000e+00, 1.00000000e+00], [1.00000000e-04, -7.70000000e-02]], [0.32168, 0.33767], 'ACES', [[9.52552396e-01, 0.00000000e+00, 9.36786000e-05], [3.43966450e-01, 7.28166097e-01, -7.21325464e-02], [0.00000000e+00, 0.00000000e+00, 1.00882518e+00]], [[1.04981102e+00, 0.00000000e+00, -9.74845000e-05], [-4.95903023e-01, 1.37331305e+00, 9.82400361e-02], [0.00000000e+00, 0.00000000e+00, 9.91252018e-01]], linear_function, linear_function, False, False)**

ACES2065-1 colourspace, base encoding, used for exchange of full fidelity images and archiving.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

colour.models.RGB_COLOURSPACE_ACESCC

colour.models.**RGB_COLOURSPACE_ACESCC = RGB_Colourspace('ACEScc', [[0.713, 0.293], [0.165, 0.83], [0.128, 0.044]], [0.32168, 0.33767], 'ACES', [[0.66245418, 0.13400421, 0.15618769], [0.27222872, 0.67408177, 0.05368952], [-0.00557465, 0.00406073, 1.0103391]], [[1.64102338, -0.32480329, -0.2364247], [-0.66366286, 1.61533159, 0.01675635], [0.01172189, -0.00828444, 0.98839486]], log_encoding_ACEScc, log_decoding_ACEScc, False, False)**

ACEScc colourspace, a working space for color correctors, target for ASC-CDL values created on-set.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

colour.models.RGB_COLOURSPACE_ACESCCT

colour.models.**RGB_COLOURSPACE_ACESCCT = RGB_Colourspace('ACEScct', [[0.713, 0.293], [0.165, 0.83], [0.128, 0.044]], [0.32168, 0.33767], 'ACES', [[0.66245418, 0.13400421, 0.15618769], [0.27222872, 0.67408177, 0.05368952], [-0.00557465, 0.00406073, 1.0103391]], [[1.64102338, -0.32480329, -0.2364247], [-0.66366286, 1.61533159, 0.01675635], [0.01172189, -0.00828444, 0.98839486]], log_encoding_ACEScct, log_decoding_ACEScct, False, False)**

ACEScct colourspace, an alternative working space for colour correctors, intended to be transient and internal to software or hardware systems, and is specifically not intended for interchange or archiving.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

colour.models.RGB_COLOURSPACE_ACESPROXY

```
colour.models.RGB_COLOURSPACE_ACESPROXY = RGB_Colourspace('ACESproxy', [[ 0.713, 0.293], [ 0.165, 0.83 ], [ 0.128, 0.044]], [ 0.32168, 0.33767], 'ACES', [[ 0.66245418, 0.13400421, 0.15618769], [ 0.27222872, 0.67408177, 0.05368952], [-0.00557465, 0.00406073, 1.0103391 ]], [[ 1.64102338, -0.32480329, -0.2364247 ], [-0.66366286, 1.61533159, 0.01675635], [ 0.01172189, -0.00828444, 0.98839486]], log_encoding_ACESproxy, log_decoding_ACESproxy, False, False)
```

ACESproxy colourspace, a lightweight encoding for transmission over HD-SDI (or other production transmission schemes), onset look management. Not intended to be stored or used in production imagery or for final colour grading / mastering.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee13], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

colour.models.RGB_COLOURSPACE_ACESCG

```
colour.models.RGB_COLOURSPACE_ACESCG = RGB_Colourspace('ACEScg', [[ 0.713, 0.293], [ 0.165, 0.83 ], [ 0.128, 0.044]], [ 0.32168, 0.33767], 'ACES', [[ 0.66245418, 0.13400421, 0.15618769], [ 0.27222872, 0.67408177, 0.05368952], [-0.00557465, 0.00406073, 1.0103391 ]], [[ 1.64102338, -0.32480329, -0.2364247 ], [-0.66366286, 1.61533159, 0.01675635], [ 0.01172189, -0.00828444, 0.98839486]], linear_function, linear_function, False, False)
```

ACEScg colourspace, a working space for paint/compositor applications that don't support ACES2065-1 or ACEScc.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceandTCouncilAcademyCESACESPSubcommittee15], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

colour.models.RGB_COLOURSPACE_ADOBE_RGB1998

```
colour.models.RGB_COLOURSPACE_ADOBE_RGB1998 = RGB_Colourspace('Adobe RGB (1998)', [[ 0.64, 0.33], [ 0.21, 0.71], [ 0.15, 0.06]], [ 0.3127, 0.329 ], 'D65', [[ 0.57667, 0.18556, 0.18823], [ 0.29734, 0.62736, 0.07529], [ 0.02703, 0.07069, 0.99134]], [[ 2.04159, -0.56501, -0.34473], [-0.96924, 1.87597, 0.04156], [ 0.01344, -0.11836, 1.01517]], functools.partial(<function gamma_function>, exponent=0.4547069271758437), functools.partial(<function gamma_function>, exponent=2.19921875), False, False)
```

Adobe RGB (1998) colourspace.

References

[AdobeSystems05]

colour.models.RGB_COLOURSPACE_ADOBE_WIDE_GAMUT_RGB

```
colour.models.RGB_COLOURSPACE_ADOBE_WIDE_GAMUT_RGB = RGB_Colourspace('Adobe Wide Gamut RGB', [[ 0.7347, 0.2653], [ 0.1152, 0.8264], [ 0.1566, 0.0177]], [ 0.3457, 0.3585], 'D50', [[ 0.71650072, 0.10102057, 0.14677439], [ 0.25872824, 0.72468231, 0.01658944], [ 0. , 0.05121182, 0.77389278]], [[ 1.46230418, -0.18452564, -0.27338105], [-0.52286828, 1.4479884 , 0.06812617], [ 0.03460045, -0.09581963, 1.28766046]], functools.partial(<function gamma_function>, exponent=0.4547069271758437), functools.partial(<function gamma_function>, exponent=2.19921875), False, False)
```

Adobe Wide Gamut RGB colourspace.

References

[Wikipedia04d]

colour.models.RGB_COLOURSPACE_ARRI_WIDE_GAMUT_3

```
colour.models.RGB_COLOURSPACE_ARRI_WIDE_GAMUT_3 = RGB_Colourspace('ARRI Wide Gamut 3', [[ 0.684 , 0.313 ], [ 0.221 , 0.848 ], [ 0.0861, -0.102 ]], [ 0.3127, 0.329 ], 'D65', [[ 0.638008, 0.214704, 0.097744], [ 0.291954, 0.823841, -0.115795], [ 0.002798, -0.067034, 1.153294]], [[ 1.789066, -0.482534, -0.200076], [-0.639849, 1.3964 , 0.194432], [-0.041532, 0.082335, 0.878868]], log_encoding_ARRILogC3, log_decoding_ARRILogC3, False, False)
```

ARRI Wide Gamut 3 colourspace.

References

[ARRI12]

colour.models.RGB_COLOURSPACE_ARRI_WIDE_GAMUT_4

colour.models.**RGB_COLOURSPACE_ARRI_WIDE_GAMUT_4 = RGB_Colourspace('ARRI Wide Gamut 4', [[0.7347, 0.2653], [0.1424, 0.8576], [0.0991, -0.0308]], [0.3127, 0.329], 'D65', [[0.70485832, 0.1297603 , 0.11583731], [0.25452418, 0.78147773, -0.03600191], [0. , 0. , 1.08905775]], [[1.50921547, -0.25059735, -0.16881148], [-0.49154545, 1.36124555, 0.09728294], [0. , 0. , 0.91822495]], log_encoding_ARRILogC4, log_decoding_ARRILogC4, False, False)**

ARRI Wide Gamut 4 colourspace.

References

[CB22]

colour.models.RGB_COLOURSPACE_APPLE_RGB

colour.models.**RGB_COLOURSPACE_APPLE_RGB = RGB_Colourspace('Apple RGB', [[0.625, 0.34], [0.28 , 0.595], [0.155, 0.07]], [0.3127, 0.329], 'D65', [[0.44966162, 0.31625612, 0.18453819], [0.24461592, 0.67204425, 0.08333983], [0.02518105, 0.14118577, 0.92269093]], [[2.95197848, -1.2896043 , -0.47391531], [-1.08508357, 1.99080934, 0.03720168], [0.08547221, -0.26942971, 1.09102767]], functools.partial(<function gamma_function>, exponent=0.5555555555555556), functools.partial(<function gamma_function>, exponent=1.8), False, False)**

Apple RGB colourspace.

References

[SBS99]

colour.models.RGB_COLOURSPACE_BEST_RGB

colour.models.**RGB_COLOURSPACE_BEST_RGB = RGB_Colourspace('Best RGB', [[0.73519164, 0.26480836], [0.21533613, 0.77415966], [0.13012295, 0.03483607]], [0.3457, 0.3585], 'D50', [[0.6318944 , 0.20538793, 0.12701335], [0.22760177, 0.73839465, 0.03400357], [0. , 0.01001892, 0.81508568]], [[1.75737181, -0.48538023, -0.25359913], [-0.54199672, 1.50475404, 0.02168337], [0.00666215, -0.01849623, 1.22659836]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Best RGB colourspace.

References

[HutchColora]

colour.models.RGB_COLOURSPACE_BETA_RGB

```
colour.models.RGB_COLOURSPACE_BETA_RGB = RGB_Colourspace('Beta RGB', [[ 0.6888, 0.3112], [ 0.1986, 0.7551], [ 0.1265, 0.0352]], [ 0.3457, 0.3585], 'D50', [[ 6.71355903e-01, 1.74572381e-01, 1.18367393e-01], [ 3.03318753e-01, 6.63744233e-01, 3.29370136e-02], [ 5.41053122e-17, 4.06983949e-02, 7.84406208e-01]], [[ 1.68297071, -0.42817109, -0.23598255], [-0.77107152, 1.70666472, 0.04469277], [ 0.04000653, -0.08854917, 1.27253082]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)
```

Beta RGB colourspace.

References

[Lin14]

colour.models.RGB_COLOURSPACE_BLACKMAGIC_WIDE_GAMUT

```
colour.models.RGB_COLOURSPACE_BLACKMAGIC_WIDE_GAMUT = RGB_Colourspace('Blackmagic Wide Gamut', [[ 0.7177215, 0.3171181], [ 0.228041 , 0.861569 ], [ 0.1005841, -0.0820452]], [ 0.312717 , 0.3290312], 'Blackmagic Wide Gamut', [[ 0.60653037, 0.2204081 , 0.123479 ], [ 0.26798941, 0.83273088, -0.10072029], [-0.02944217, -0.08661061, 1.20486076]], [[ 1.86638234, -0.51839734, -0.23460981], [-0.60034249, 1.37814896, 0.17673183], [ 0.00245199, 0.08639967, 0.83694271]], oetf_BlackmagicFilmGeneration5, oetf_inverse_BlackmagicFilmGeneration5, True, True)
```

Blackmagic Wide Gamut colourspace.

References

[BlackmagicDesign21]

colour.models.RGB_COLOURSPACE_BT470_525

```
colour.models.RGB_COLOURSPACE_BT470_525 = RGB_Colourspace('ITU-R BT.470 - 525', [[ 0.67, 0.33], [ 0.21, 0.71], [ 0.14, 0.08]], [ 0.31006, 0.31616], 'C', [[ 6.06863809e-01, 1.73507281e-01, 2.00334881e-01], [ 2.98903070e-01, 5.86619855e-01, 1.14477075e-01], [ -5.02801622e-17, 6.60980118e-02, 1.11615148e+00]], [[ 1.91008143, -0.53247794, -0.28822201], [-0.98463135, 1.99910001, -0.02830719], [ 0.05830945, -0.11838584, 0.89761208]], functools.partial(<function gamma_function>, exponent=0.35714285714285715), functools.partial(<function gamma_function>, exponent=2.8), False, False)
```

Recommendation ITU-R BT.470 - 525 colourspace.

References

[InternationalTUnion98b]

colour.models.RGB_COLOURSPACE_BT470_625

colour.models.**RGB_COLOURSPACE_BT470_625 = RGB_Colourspace('ITU-R BT.470 - 625', [[0.64, 0.33], [0.29, 0.6], [0.15, 0.06]], [0.3127, 0.329], 'D65', [[0.43055381, 0.3415498 , 0.17835231], [0.22200431, 0.70665477, 0.07134092], [0.02018221, 0.12955337, 0.93932217]], [[3.06336109, -1.39339017, -0.47582374], [-0.96924364, 1.8759675 , 0.04155506], [0.06786105, -0.22879927, 1.06908962]], functools.partial(<function gamma_function>, exponent=0.35714285714285715), functools.partial(<function gamma_function>, exponent=2.8), False, False)**

Recommendation ITU-R BT.470 - 625 colourspace.

References

[InternationalTUnion98b]

colour.models.RGB_COLOURSPACE_BT709

colour.models.**RGB_COLOURSPACE_BT709 = RGB_Colourspace('ITU-R BT.709', [[0.64, 0.33], [0.3 , 0.6], [0.15, 0.06]], [0.3127, 0.329], 'D65', [[0.4123908 , 0.35758434, 0.18048079], [0.21263901, 0.71516868, 0.07219232], [0.01933082, 0.11919478, 0.95053215]], [[3.24096994, -1.53738318, -0.49861076], [-0.96924364, 1.8759675 , 0.04155506], [0.05563008, -0.20397696, 1.05697151]], oetf_BT709, oetf_inverse_BT709, False, False)**

Recommendation ITU-R BT.709 colourspace.

References

[InternationalTUnion15b]

colour.models.RGB_COLOURSPACE_BT2020

colour.models.**RGB_COLOURSPACE_BT2020 = RGB_Colourspace('ITU-R BT.2020', [[0.708, 0.292], [0.17 , 0.797], [0.131, 0.046]], [0.3127, 0.329], 'D65', [[6.36958048e-01, 1.44616904e-01, 1.68880975e-01], [2.62700212e-01, 6.77998072e-01, 5.93017165e-02], [4.99410657e-17, 2.80726930e-02, 1.06098506e+00]], [[1.71665119, -0.35567078, -0.25336628], [-0.66668435, 1.61648124, 0.01576855], [0.01763986, -0.04277061, 0.94210312]], oetf_BT2020, oetf_inverse_BT2020, False, False)**

RecommendationITU-R BT.2020 colourspace.

The wavelength of the *RecommendationITU-R BT.2020* primary colours are:

- 630nm for the red primary colour
- 532nm for the green primary colour
- 467nm for the blue primary colour.

References

[InternationalTUnion15a]

colour.models.RGB_COLOURSPACE_CIE_RGB

```
colour.models.RGB_COLOURSPACE_CIE_RGB = RGB_Colourspace('CIE RGB', [[ 0.73474284,
0.26525716], [ 0.27377903, 0.7174777 ], [ 0.16655563, 0.00891073]], [ 0.33333333,
0.33333333], 'E', [[ 0.49 , 0.31 , 0.2 ], [ 0.1769, 0.8124, 0.0107], [ 0. , 0.0099,
0.9901]], [[ 2.36449012, -0.89655263, -0.46793749], [-0.51493525, 1.42633279, 0.08860245],
[ 0.00514883, -0.01426189, 1.00911305]], functools.partial(<function gamma_function>,
exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2),
False, False)
```

CIE RGB colourspace.

References

[FBH97]

colour.models.RGB_COLOURSPACE_CINEMA_GAMUT

```
colour.models.RGB_COLOURSPACE_CINEMA_GAMUT = RGB_Colourspace('Cinema Gamut', [[ 0.74,
0.27], [ 0.17, 1.14], [ 0.08, -0.1 ]], [ 0.3127, 0.329 ], 'D65', [[ 0.71604965, 0.12968348,
0.1047228 ], [ 0.26126136, 0.86964215, -0.1309035 ], [-0.00967635, -0.23648164,
1.33521573]], [[ 1.48981827, -0.2608959 , -0.14242652], [-0.45816657, 1.26162778,
0.15962363], [-0.07034967, 0.22155767, 0.7761816 ]], linear_function, linear_function,
False, False)
```

Canon Cinema Gamut colourspace.

References

[Canon14]

colour.models.RGB_COLOURSPACE_COLOR_MATCH_RGB

```
colour.models.RGB_COLOURSPACE_COLOR_MATCH_RGB = RGB_Colourspace('ColorMatch RGB', [[ 0.63
, 0.34 ], [ 0.295, 0.605], [ 0.15 , 0.075]], [ 0.3457, 0.3585], 'D50', [[ 0.5094668 ,
0.32087954, 0.13394933], [ 0.27495034, 0.658075 , 0.06697467], [ 0.02426032, 0.10877273,
0.69207155]], [[ 2.64164976, -1.22313179, -0.39291946], [-1.11207173, 2.05919502,
0.01596275], [ 0.08218196, -0.28076676, 1.45620209]], functools.partial(<function
gamma_function>, exponent=0.5555555555555556), functools.partial(<function
gamma_function>, exponent=1.8), False, False)
```

ColorMatch RGB colourspace.

References

[Lin14]

colour.models.RGB_COLOURSPACE_DAVINCI_WIDE_GAMUT

colour.models.**RGB_COLOURSPACE_DAVINCI_WIDE_GAMUT = RGB_Colourspace('DaVinci Wide Gamut', [[0.8 , 0.313], [0.1682, 0.9877], [0.079 , -0.1155]], [0.3127, 0.329], 'D65', [[0.70062239, 0.14877482, 0.10105872], [0.27411851, 0.8736319 , -0.14775041], [-0.09896291, -0.13789533, 1.32591599]], [[1.51667204, -0.28147805, -0.14696363], [-0.4649171 , 1.25142378, 0.17488461], [0.06484905, 0.10913934, 0.76141462]], oetf_DaVinciIntermediate, oetf_inverse_DaVinciIntermediate, True, True)**

DaVinci Wide Gamut colourspace.

References

[BlackmagicDesign20b], [BlackmagicDesign20a]

colour.models.RGB_COLOURSPACE_DCDM_XYZ

colour.models.**RGB_COLOURSPACE_DCDM_XYZ = RGB_Colourspace('DCDM XYZ', [[1., 0.], [0., 1.], [0., 0.]], [0.33333333, 0.33333333], 'E', [[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]], [[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]], eotf_inverse_DCDM, eotf_DCDM, False, False)**

DCDM XYZ colourspace.

References

[DigitalCInitiatives07]

colour.models.RGB_COLOURSPACE_DCI_P3

colour.models.**RGB_COLOURSPACE_DCI_P3 = RGB_Colourspace('DCI-P3', [[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.06]], [0.314, 0.351], 'DCI-P3', [[4.45169816e-01, 2.77134409e-01, 1.72282670e-01], [2.09491678e-01, 7.21595254e-01, 6.89130679e-02], [-3.63410132e-17, 4.70605601e-02, 9.07355394e-01]], [[2.72539403, -1.01800301, -0.4401632], [-0.79516803, 1.68973205, 0.02264719], [0.04124189, -0.08763902, 1.10092938]], functools.partial(<function gamma_function>, exponent=0.3846153846153846), functools.partial(<function gamma_function>, exponent=2.6), False, False)**

DCI-P3 colourspace.

References

[DigitalCInitiatives07], [HewlettPDCompany09]

colour.models.RGB_COLOURSPACE_DCI_P3_P

```
colour.models.RGB_COLOURSPACE_DCI_P3_P = RGB_Colourspace('DCI-P3-P', [[ 0.74, 0.27], [
0.22, 0.78], [ 0.09, -0.09]], [ 0.314, 0.351], 'DCI-P3', [[ 0.55907356, 0.24893595,
0.08657739], [ 0.2039863 , 0.88259109, -0.08657739], [-0.00755505, 0. , 0.961971 ]], [[
1.99040349, -0.56139586, -0.22966194], [-0.45849279, 1.262346 , 0.15487549], [ 0.01563207,
-0.00440904, 1.03772867]], functools.partial(<function gamma_function>,
exponent=0.3846153846153846), functools.partial(<function gamma_function>, exponent=2.6),
False, False)
```

DCI-P3+ colourspace.

Notes

- The actual *DCI-P3+* colourspace name is *DCI-P3-P* to avoid canonical key collisions in the *colour.utilities.CanonicalMapping* class.

References

[Canon14]

colour.models.RGB_COLOURSPACE_DISPLAY_P3

```
colour.models.RGB_COLOURSPACE_DISPLAY_P3 = RGB_Colourspace('Display P3', [[ 0.68 , 0.32 ],
[ 0.265, 0.69 ], [ 0.15 , 0.06 ]], [ 0.3127, 0.329 ], 'D65', [[ 4.86570949e-01,
2.65667693e-01, 1.98217285e-01], [ 2.28974564e-01, 6.91738522e-01, 7.92869141e-02], [
-3.97207552e-17, 4.51133819e-02, 1.04394437e+00]], [[ 2.49349691, -0.93138362,
-0.40271078], [-0.82948897, 1.76266406, 0.02362469], [ 0.03584583, -0.07617239,
0.95688452]], eotf_inverse_sRGB, eotf_sRGB, False, False)
```

Display P3 colourspace.

References

[AppleInc19]

colour.models.RGB_COLOURSPACE_DON_RGB_4

```
colour.models.RGB_COLOURSPACE_DON_RGB_4 = RGB_Colourspace('Don RGB 4', [[ 0.69612069,
0.29956897], [ 0.21468298, 0.76529477], [ 0.12993763, 0.03534304]], [ 0.3457, 0.3585],
'D50', [[ 0.64631888, 0.19296024, 0.12501655], [ 0.27813723, 0.68785827, 0.0340045 ], [
0.00400197, 0.01799629, 0.80310634]], [[ 1.75819127, -0.48659205, -0.25308814],
[-0.7112839 , 1.65225302, 0.04076449], [ 0.00717743, -0.03459953, 1.24551283]],
functools.partial(<function gamma_function>, exponent=0.45454545454545453),
functools.partial(<function gamma_function>, exponent=2.2), False, False)
```

Don RGB 4 colourspace.

References

[HutchColorb]

colour.models.RGB_COLOURSPACE_EBU_3213_E

```
colour.models.RGB_COLOURSPACE_EBU_3213_E = RGB_Colourspace('EBU Tech. 3213-E', [[ 0.64, 0.33], [ 0.29, 0.6 ], [ 0.15, 0.06]], [ 0.313, 0.329], 'D65', [[ 0.43194331, 0.341235 , 0.17818948], [ 0.22272077, 0.70600344, 0.07127579], [ 0.02024734, 0.12943396, 0.93846459]], [[ 3.05350675, -1.38890786, -0.47429309], [-0.97013781, 1.87769818, 0.04159339], [ 0.06792306, -0.22900835, 1.07006656]], linear_function, linear_function, False, False)
```

EBU Tech. 3213-E colourspace.

References

[EuropeanBUnion75]

colour.models.RGB_COLOURSPACE_ECI_RGB_V2

```
colour.models.RGB_COLOURSPACE_ECI_RGB_V2 = RGB_Colourspace('ECI RGB v2', [[ 0.67010309, 0.32989691], [ 0.20990566, 0.70990566], [ 0.14006179, 0.08032956]], [ 0.3457, 0.3585], 'D50', [[ 0.65032438, 0.177949 , 0.13602229], [ 0.3201597 , 0.60182752, 0.07801279], [ 0. , 0.06798052, 0.75712409]], [[ 1.78215602, -0.49656317, -0.26901095], [-0.95923427, 1.94844461, -0.02843173], [ 0.08612755, -0.17494658, 1.32334029]], functools.partial(<function _scale_domain_0_100_range_0_1>, callable_=<function lightness_CIE1976>), functools.partial(<function _scale_domain_0_100_range_0_1>, callable_=<function luminance_CIE1976>), False, False)
```

ECI RGB v2 colourspace.

References

[EuropeanCInitiative02]

colour.models.RGB_COLOURSPACE_EKTA_SPACE_PS_5

```
colour.models.RGB_COLOURSPACE_EKTA_SPACE_PS_5 = RGB_Colourspace('Ekta Space PS 5', [[ 0.69473684, 0.30526316], [ 0.26 , 0.7 ], [ 0.10972851, 0.00452489]], [ 0.3457, 0.3585], 'D50', [[ 0.59433686, 0.27294481, 0.09701401], [ 0.26114801, 0.73485141, 0.00400058], [ 0. , 0.04199151, 0.78311309]], [[ 2.00336603, -0.73013869, -0.24445204], [-0.71215462, 1.62076569, 0.07994372], [ 0.03818663, -0.08690749, 1.27266809]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)
```

Ekta Space PS 5 colourspace.

References

[Hol]

colour.models.RGB_COLOURSPACE_F_GAMUT

colour.models.**RGB_COLOURSPACE_F_GAMUT = RGB_Colourspace('F-Gamut', [[0.708, 0.292], [0.17 , 0.797], [0.131, 0.046]], [0.3127, 0.329], 'D65', [[6.36958048e-01, 1.44616904e-01, 1.68880975e-01], [2.62700212e-01, 6.77998072e-01, 5.93017165e-02], [4.99410657e-17, 2.80726930e-02, 1.06098506e+00]], [[1.71665119, -0.35567078, -0.25336628], [-0.66668435, 1.61648124, 0.01576855], [0.01763986, -0.04277061, 0.94210312]], log_encoding_FLog, log_decoding_FLog, False, False)**

Fujifilm F-Gamut colourspace.

References

[Fujifilm22b]

colour.models.RGB_COLOURSPACE_H273_GENERIC_FILM

colour.models.**RGB_COLOURSPACE_H273_GENERIC_FILM = RGB_Colourspace('ITU-T H.273 - Generic Film', [[0.681, 0.319], [0.243, 0.692], [0.145, 0.049]], [0.31 , 0.316], 'C', [[5.41353080e-01, 2.38201725e-01, 2.01457854e-01], [2.53585363e-01, 6.78335776e-01, 6.80788609e-02], [-4.41279481e-17, 6.37165107e-02, 1.11982779e+00]], [[2.19248548, -0.73706449, -0.34962064], [-0.82433417, 1.75978548, 0.04131385], [0.04690337, -0.10012914, 0.89064375]], linear_function, linear_function, False, False)**

Recommendation ITU-T H.273 Generic Film (colour filters using Illuminant C) colourspace.

References

[InternationalTUnion21]

colour.models.RGB_COLOURSPACE_H273_22_UNSPECIFIED

colour.models.**RGB_COLOURSPACE_H273_22_UNSPECIFIED = RGB_Colourspace('ITU-T H.273 - 22 Unspecified', [[0.63 , 0.34], [0.295, 0.605], [0.155, 0.077]], [0.3127, 0.329], 'D65', [[0.42942013, 0.3277917 , 0.1932441], [0.23175055, 0.67225077, 0.09599868], [0.02044858, 0.11111583, 0.95749334]], [[3.13288278, -1.44707454, -0.48720324], [-1.08850877, 2.01538781, 0.01762239], [0.05941301, -0.20297883, 1.05275352]], linear_function, linear_function, False, False)**

Recommendation ITU-T H.273 row *22* colourspace as given in *Table 2 - Interpretation of colour primaries (ColourPrimaries) value*.

References

[InternationalTUnion21]

colour.models.RGB_COLOURSPACE_PROTUNE_NATIVE

colour.models.**RGB_COLOURSPACE_PROTUNE_NATIVE = RGB_Colourspace('Protune Native', [[0.69848046, 0.19302645], [0.32955538, 1.02459662], [0.10844263, -0.03467857]], [0.3127, 0.329], 'D65', [[0.50225719, 0.29296671, 0.15523203], [0.13879976, 0.91084146, -0.04964122], [0.07801426, -0.31483251, 1.325876]], [[2.2668965 , -0.83163359, -0.29654225], [-0.35733783, 1.24337315, 0.08838899], [-0.21823445, 0.34417515, 0.79265501]], log_encoding_Protune, log_decoding_Protune, False, False)**

Protune Native colourspace.

References

[GoProDM16], [Man15]

colour.models.RGB_COLOURSPACE_MAX_RGB

colour.models.**RGB_COLOURSPACE_MAX_RGB = RGB_Colourspace('Max RGB', [[0.73413379, 0.26586621], [0.10039113, 0.89960887], [0.03621495, 0.]], [0.3457, 0.3585], 'D50', [[0.85630404, 0.07698771, 0.03100393], [0.31011011, 0.68988989, 0.], [0. , 0. , 0.8251046]], [[1.2169928 , -0.13580933, -0.04572942], [-0.54704638, 1.51055387, 0.02055568], [0. , 0. , 1.21196755]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Max RGB colourspace.

References

[HutchColorc]

colour.models.RGB_COLOURSPACE_NTSC1953

colour.models.**RGB_COLOURSPACE_NTSC1953 = RGB_Colourspace('NTSC (1953)', [[0.67, 0.33], [0.21, 0.71], [0.14, 0.08]], [0.31006, 0.31616], 'C', [[6.06863809e-01, 1.73507281e-01, 2.00334881e-01], [2.98903070e-01, 5.86619855e-01, 1.14477075e-01], [-5.02801622e-17, 6.60980118e-02, 1.11615148e+00]], [[1.91008143, -0.53247794, -0.28822201], [-0.98463135, 1.99910001, -0.02830719], [0.05830945, -0.11838584, 0.89761208]], functools.partial(<function gamma_function>, exponent=0.35714285714285715), functools.partial(<function gamma_function>, exponent=2.8), False, False)**

NTSC (1953) colourspace.

References

[InternationalTUnion98b]

colour.models.RGB_COLOURSPACE_NTSC1987

```
colour.models.RGB_COLOURSPACE_NTSC1987 = RGB_Colourspace('NTSC (1987)', [[ 0.63 , 0.34 ], [
0.31 , 0.595], [ 0.155, 0.07 ]], [ 0.3127, 0.329 ], 'D65', [[ 0.3935209 , 0.36525808,
0.19167695], [ 0.21237636, 0.70105986, 0.08656378], [ 0.01873909, 0.11193393, 0.95838473]],
[[ 3.50600328, -1.73979073, -0.54405827], [-1.06904756, 1.97777888, 0.03517142], [
0.05630659, -0.19697565, 1.04995233]], functools.partial(<function gamma_function>,
exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2),
False, False)
```

NTSC (1987) colourspace.

References

[SocietyoMPaTEngineers04]

colour.models.RGB_COLOURSPACE_P3_D65

```
colour.models.RGB_COLOURSPACE_P3_D65 = RGB_Colourspace('P3-D65', [[ 0.68 , 0.32 ], [ 0.265,
0.69 ], [ 0.15 , 0.06 ]], [ 0.3127, 0.329 ], 'D65', [[ 4.86570949e-01, 2.65667693e-01,
1.98217285e-01], [ 2.28974564e-01, 6.91738522e-01, 7.92869141e-02], [ -3.97207552e-17,
4.51133819e-02, 1.04394437e+00]], [[ 2.49349691, -0.93138362, -0.40271078], [-0.82948897,
1.76266406, 0.02362469], [ 0.03584583, -0.07617239, 0.95688452]],
functools.partial(<function gamma_function>, exponent=0.3846153846153846),
functools.partial(<function gamma_function>, exponent=2.6), False, False)
```

P3-D65 colourspace.

colour.models.RGB_COLOURSPACE_PAL_SECAM

```
colour.models.RGB_COLOURSPACE_PAL_SECAM = RGB_Colourspace('Pal/Secam', [[ 0.64, 0.33], [
0.29, 0.6 ], [ 0.15, 0.06]], [ 0.3127, 0.329 ], 'D65', [[ 0.43055381, 0.3415498 ,
0.17835231], [ 0.22200431, 0.70665477, 0.07134092], [ 0.02018221, 0.12955337, 0.93932217]],
[[ 3.06336109, -1.39339017, -0.47582374], [-0.96924364, 1.8759675 , 0.04155506], [
0.06786105, -0.22879927, 1.06908962]], functools.partial(<function gamma_function>,
exponent=0.35714285714285715), functools.partial(<function gamma_function>, exponent=2.8),
False, False)
```

Pal/Secam colourspace.

References

[InternationalTUnion98b]

colour.models.RGB_COLOURSPACE_RED_COLOR

colour.models.**RGB_COLOURSPACE_RED_COLOR = RGB_Colourspace('REDcolor', [[0.70105856, 0.33018098], [0.29881132, 0.62516925], [0.13503868, 0.03526178]], [0.3127, 0.329], 'D65', [[0.42302331, 0.36210731, 0.16532531], [0.19923335, 0.75759632, 0.04317033], [-0.01885014, 0.09212233, 1.01578557]], [[2.99433635, -1.37906534, -0.42873703], [-0.79472663, 1.69283865, 0.0574019], [0.12764085, -0.17911636, 0.97129776]], log_encoding_REDLogFilm, log_decoding_REDLogFilm, False, False)**

REDcolor colourspace.

References

[Man15], [SonyImageworks12]

colour.models.RGB_COLOURSPACE_RED_COLOR_2

colour.models.**RGB_COLOURSPACE_RED_COLOR_2 = RGB_Colourspace('REDcolor2', [[0.89740722, 0.33077623], [0.29602209, 0.68463555], [0.09979951, -0.02300051]], [0.3127, 0.329], 'D65', [[0.44957762, 0.3734296 , 0.12744871], [0.16571026, 0.86366248, -0.02937275], [-0.11431396, 0.02440023, 1.17897148]], [[2.55060735, -1.09426927, -0.30298724], [-0.48063394, 1.36324834, 0.0859211], [0.2572561 , -0.13431523, 0.81704083]], log_encoding_REDLogFilm, log_decoding_REDLogFilm, False, False)**

REDcolor2 colourspace.

References

[Man15], [SonyImageworks12]

colour.models.RGB_COLOURSPACE_RED_COLOR_3

colour.models.**RGB_COLOURSPACE_RED_COLOR_3 = RGB_Colourspace('REDcolor3', [[0.70259866, 0.33018559], [0.29578224, 0.68974826], [0.11109053, -0.00433232]], [0.3127, 0.329], 'D65', [[0.47986312, 0.33439883, 0.13619398], [0.22551123, 0.77980008, -0.00531131], [-0.02239109, 0.01635861, 1.09509023]], [[2.58673915, -1.10240102, -0.32705386], [-0.74762558, 1.6008681 , 0.10074495], [0.06405867, -0.04645456, 0.90497461]], log_encoding_REDLogFilm, log_decoding_REDLogFilm, False, False)**

REDcolor3 colourspace.

References

[Man15], [SonyImageworks12]

colour.models.RGB_COLOURSPACE_RED_COLOR_4

colour.models.**RGB_COLOURSPACE_RED_COLOR_4 = RGB_Colourspace('REDcolor4', [[0.70259815, 0.3301851], [0.29578233, 0.68974825], [0.14445924, 0.05083772]], [0.3127, 0.329], 'D65', [[0.44431783, 0.30962925, 0.19650885], [0.20880659, 0.72203852, 0.06915489], [-0.02073188, 0.0151468 , 1.09464284]], [[2.78855329, -1.18687705, -0.42561558], [-0.81255797, 1.73265028, 0.03640786], [0.06405707, -0.04645378, 0.9049753]], log_encoding_REDLogFilm, log_decoding_REDLogFilm, False, False)**

REDcolor4 colourspace.

References

[Man15], [SonyImageworks12]

colour.models.RGB_COLOURSPACE_RED_WIDE_GAMUT_RGB

colour.models.**RGB_COLOURSPACE_RED_WIDE_GAMUT_RGB = RGB_Colourspace('REDWideGamutRGB', [[0.780308, 0.304253], [0.121595, 1.493994], [0.095612, -0.084589]], [0.3127, 0.329], 'D65', [[0.735275, 0.068609, 0.146571], [0.286694, 0.842979, -0.129673], [-0.079681, -0.347343, 1.516082]], [[1.41280661, -0.17752237, -0.15177038], [-0.48620319, 1.29069621, 0.15740028], [-0.03713878, 0.28637576, 0.68767961]], log_encoding_Log3G10, log_decoding_Log3G10, False, False)**

REDWideGamutRGB colourspace.

References

[Man15], [Nat16], [SonyImageworks12]

colour.models.RGB_COLOURSPACE_DRAGON_COLOR

colour.models.**RGB_COLOURSPACE_DRAGON_COLOR = RGB_Colourspace('DRAGONcolor', [[0.75865589, 0.33035535], [0.29492362, 0.70805324], [0.0859616 , -0.04587944]], [0.3127, 0.329], 'D65', [[0.49831915, 0.34905932, 0.10307746], [0.21699218, 0.83802234, -0.05501452], [-0.05846657, -0.00352329, 1.15104761]], [[2.41407671, -1.00664042, -0.26429553], [-0.61715986, 1.45087355, 0.12461203], [0.12073206, -0.04669048, 0.85573054]], log_encoding_REDLogFilm, log_decoding_REDLogFilm, False, False)**

DRAGONcolor colourspace.

References

[Man15], [SonyImageworks12]

colour.models.RGB_COLOURSPACE_DRAGON_COLOR_2

colour.models.**RGB_COLOURSPACE_DRAGON_COLOR_2 = RGB_Colourspace('DRAGONcolor2', [[0.75865621, 0.33035584], [0.29492389, 0.70805336], [0.14416873, 0.05035738]], [0.3127, 0.329], 'D65', [[0.43856251, 0.30720212, 0.2046913], [0.19097146, 0.73753094, 0.0714976], [-0.05145591, -0.0031012 , 1.14361486]], [[2.72655873, -1.13744045, -0.41690486], [-0.71770143, 1.654923 , 0.02499461], [0.12073281, -0.04669036, 0.85572978]], log_encoding_REDLogFilm, log_decoding_REDLogFilm, False, False)**

DRAGONcolor2 colourspace.

References

[Man15], [SonyImageworks12]

colour.models.RGB_COLOURSPACE_ROMM_RGB

colour.models.**RGB_COLOURSPACE_ROMM_RGB = RGB_Colourspace('ROMM RGB', [[7.34700000e-01, 2.65300000e-01], [1.59600000e-01, 8.40400000e-01], [3.66000000e-02, 1.00000000e-04]], [0.3457, 0.3585], 'D50', [[7.97700000e-01, 1.35200000e-01, 3.13000000e-02], [2.88000000e-01, 7.11900000e-01, 1.00000000e-04], [0.00000000e+00, 0.00000000e+00, 8.24900000e-01]], [[1.346 , -0.2556, -0.0511], [-0.5446, 1.5082, 0.0205], [0. , 0. , 1.2123]], cctf_encoding_ROMMRGB, cctf_decoding_ROMMRGB, False, False)**

ROMM RGB colourspace.

References

[ANSI03], [SWG00]

colour.models.RGB_COLOURSPACE_RIMM_RGB

colour.models.**RGB_COLOURSPACE_RIMM_RGB = RGB_Colourspace('RIMM RGB', [[7.34700000e-01, 2.65300000e-01], [1.59600000e-01, 8.40400000e-01], [3.66000000e-02, 1.00000000e-04]], [0.3457, 0.3585], 'D50', [[7.97700000e-01, 1.35200000e-01, 3.13000000e-02], [2.88000000e-01, 7.11900000e-01, 1.00000000e-04], [0.00000000e+00, 0.00000000e+00, 8.24900000e-01]], [[1.346 , -0.2556, -0.0511], [-0.5446, 1.5082, 0.0205], [0. , 0. , 1.2123]], cctf_encoding_RIMMRGB, cctf_decoding_RIMMRGB, False, False)**

RIMM RGB colourspace. In cases in which it is necessary to identify a specific precision level, the notation *RIMM8 RGB*, *RIMM12 RGB* and *RIMM16 RGB* is used.

References

[SWG00]

colour.models.RGB_COLOURSPACE_ERIMM_RGB

colour.models.**RGB_COLOURSPACE_ERIMM_RGB = RGB_Colourspace('ERIMM RGB', [[7.34700000e-01, 2.65300000e-01], [1.59600000e-01, 8.40400000e-01], [3.66000000e-02, 1.00000000e-04]], [0.3457, 0.3585], 'D50', [[7.97700000e-01, 1.35200000e-01, 3.13000000e-02], [2.88000000e-01, 7.11900000e-01, 1.00000000e-04], [0.00000000e+00, 0.00000000e+00, 8.24900000e-01]], [[1.346 , -0.2556, -0.0511], [-0.5446, 1.5082, 0.0205], [0. , 0. , 1.2123]], log_encoding_ERIMMRGB, log_decoding_ERIMMRGB, False, False)**

ERIMM RGB colourspace.

References

[SWG00]

colour.models.RGB_COLOURSPACE_PROPHOTO_RGB

colour.models.**RGB_COLOURSPACE_PROPHOTO_RGB = RGB_Colourspace('ProPhoto RGB', [[7.34700000e-01, 2.65300000e-01], [1.59600000e-01, 8.40400000e-01], [3.66000000e-02, 1.00000000e-04]], [0.3457, 0.3585], 'D50', [[7.97700000e-01, 1.35200000e-01, 3.13000000e-02], [2.88000000e-01, 7.11900000e-01, 1.00000000e-04], [0.00000000e+00, 0.00000000e+00, 8.24900000e-01]], [[1.346 , -0.2556, -0.0511], [-0.5446, 1.5082, 0.0205], [0. , 0. , 1.2123]], cctf_encoding_ROMMRGB, cctf_decoding_ROMMRGB, False, False)**

ProPhoto RGB colourspace, an alias colourspace for *ROMM RGB*.

References

[ANSI03], [SWG00]

colour.models.RGB_COLOURSPACE_PLASA_ANSI_E154

colour.models.**RGB_COLOURSPACE_PLASA_ANSI_E154 = RGB_Colourspace('PLASA ANSI E1.54', [[7.34700000e-01, 2.65300000e-01], [1.59600000e-01, 8.40400000e-01], [3.66000000e-02, 1.00000000e-04]], [0.4254, 0.4044], 'PLASA ANSI E1.54', [[9.08326466e-01, 1.27611601e-01, 1.59907169e-02], [3.27996477e-01, 6.71959833e-01, 4.36904833e-05], [0.00000000e+00, 0.00000000e+00, 4.20870425e-01]], [[1.18198169, -0.22446963, -0.04488538], [-0.57694792, 1.59775212, 0.02175492], [0. , 0. , 2.3760282]], linear_function, linear_function, False, False)**

PLASA ANSI E1.54 colourspace.

Notes

The *[0.4254, 0.4044]* whitepoint chromaticity coordinates are described by [Woo14] to be that of a "2° Planckian source at 3,200 K". However, we can show that the chromaticity coordinates should be *[0.4234, 0.3990]*:

```
sd = colour.sd_blackbody(3200)
colour.XYZ_to_xy(
    colour.sd_to_XYZ(
        sd, colour.MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
    )
).round(4)
```

References

[PLASANAmerica15], [Woo14]

colour.models.RGB_COLOURSPACE_RUSSELL_RGB

```
colour.models.RGB_COLOURSPACE_RUSSELL_RGB = RGB_Colourspace('Russell RGB', [[ 0.69, 0.31], [ 0.18, 0.77], [ 0.1 , 0.02]], [ 0.33243, 0.34744], 'D55', [[ 7.01583746e-01, 1.55416218e-01, 9.97983328e-02], [ 3.15204292e-01, 6.64836042e-01, 1.99596666e-02], [ 5.64430745e-17, 4.31711716e-02, 8.78225329e-01]], [[ 1.58699918, -0.35980738, -0.17216338], [-0.75352154, 1.67719311, 0.04750942], [ 0.03704107, -0.08244626, 1.13632451]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)
```

Russell RGB colourspace.

References

[Cot]

colour.models.RGB_COLOURSPACE_SMPTE_240M

```
colour.models.RGB_COLOURSPACE_SMPTE_240M = RGB_Colourspace('SMPTE 240M', [[ 0.63 , 0.34 ], [ 0.31 , 0.595], [ 0.155, 0.07 ]], [ 0.3127, 0.329 ], 'D65', [[ 0.3935209 , 0.36525808, 0.19167695], [ 0.21237636, 0.70105986, 0.08656378], [ 0.01873909, 0.11193393, 0.95838473]], [[ 3.50600328, -1.73979073, -0.54405827], [-1.06904756, 1.97777888, 0.03517142], [ 0.05630659, -0.19697565, 1.04995233]], oetf_SMPTE240M, eotf_SMPTE240M, False, False)
```

SMPTE 240M colourspace.

References

[SocietyoMPaTEngineers99],

colour.models.RGB_COLOURSPACE_SMPTE_C

```
colour.models.RGB_COLOURSPACE_SMPTE_C = RGB_Colourspace('SMPTE C', [[ 0.63 , 0.34 ], [ 0.31 , 0.595], [ 0.155, 0.07 ]], [ 0.3127, 0.329 ], 'D65', [[ 0.3935209 , 0.36525808, 0.19167695], [ 0.21237636, 0.70105986, 0.08656378], [ 0.01873909, 0.11193393, 0.95838473]], [[ 3.50600328, -1.73979073, -0.54405827], [-1.06904756, 1.97777888, 0.03517142], [ 0.05630659, -0.19697565, 1.04995233]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)
```

SMPTE C colourspace.

References

[SocietyoMPaTEngineers04]

colour.models.RGB_COLOURSPACE_S_GAMUT

```
colour.models.RGB_COLOURSPACE_S_GAMUT = RGB_Colourspace('S-Gamut', [[ 0.73 , 0.28 ], [ 0.14
, 0.855], [ 0.1 , -0.05 ]], [ 0.3127, 0.329 ], 'D65', [[ 0.70648271, 0.12880105,
0.11517216], [ 0.27097967, 0.78660641, -0.05758608], [-0.00967785, 0.00460004,
1.09413556]], [[ 1.5073999 , -0.24582214, -0.17161168], [-0.51815173, 1.35539124,
0.12587867], [ 0.0155117 , -0.00787277, 0.91191637]], log_encoding_SLog2,
log_decoding_SLog2, False, False)
```

S-Gamut colourspace.

References

[GDY+], [SonyCorporationb]

colour.models.RGB_COLOURSPACE_S_GAMUT3

```
colour.models.RGB_COLOURSPACE_S_GAMUT3 = RGB_Colourspace('S-Gamut3', [[ 0.73 , 0.28 ], [
0.14 , 0.855], [ 0.1 , -0.05 ]], [ 0.3127, 0.329 ], 'D65', [[ 0.70648271, 0.12880105,
0.11517216], [ 0.27097967, 0.78660641, -0.05758608], [-0.00967785, 0.00460004,
1.09413556]], [[ 1.5073999 , -0.24582214, -0.17161168], [-0.51815173, 1.35539124,
0.12587867], [ 0.0155117 , -0.00787277, 0.91191637]], log_encoding_SLog3,
log_decoding_SLog3, False, False)
```

S-Gamut3 colourspace.

References

[SonyCorporationc]

colour.models.RGB_COLOURSPACE_S_GAMUT3_CINE

```
colour.models.RGB_COLOURSPACE_S_GAMUT3_CINE = RGB_Colourspace('S-Gamut3.Cine', [[ 0.766,
0.275], [ 0.225, 0.8 ], [ 0.089, -0.087]], [ 0.3127, 0.329 ], 'D65', [[ 0.59908392,
0.24892552, 0.10244649], [ 0.21507582, 0.8850685 , -0.10014432], [-0.03206585, -0.02765839,
1.14878199]], [[ 1.84677897, -0.52598612, -0.21054521], [-0.44415326, 1.2594429 ,
0.14939997], [ 0.04085542, 0.01564089, 0.86820725]], log_encoding_SLog3,
log_decoding_SLog3, False, False)
```

S-Gamut3.Cine colourspace.

References

[SonyCorporationa]

colour.models.RGB_COLOURSPACE_VENICE_S_GAMUT3

colour.models.**RGB_COLOURSPACE_VENICE_S_GAMUT3 = RGB_Colourspace('Venice S-Gamut3', [[0.74046426, 0.27936437], [0.08924115, 0.89380953], [0.11048824, -0.05257933]], [0.3127, 0.329], 'D65', [[0.74422299, 0.07790652, 0.12832642], [0.28078248, 0.78028572, -0.0610682], [-0.01992929, 0.01479657, 1.09419047]], [[1.39026398, -0.13557353, -0.17061639], [-0.49777193, 1.32876782, 0.13253885], [0.03205319, -0.02043803, 0.9090178]], log_encoding_SLog3, log_decoding_SLog3, False, False)**

Venice S-Gamut3 colourspace.

References

[SonyECorporation20b]

colour.models.RGB_COLOURSPACE_VENICE_S_GAMUT3_CINE

colour.models.**RGB_COLOURSPACE_VENICE_S_GAMUT3_CINE = RGB_Colourspace('Venice S-Gamut3.Cine', [[0.77590187, 0.27450239], [0.1886829 , 0.82868494], [0.10133738, -0.08918752]], [0.3127, 0.329], 'D65', [[0.63226084, 0.20037001, 0.11782508], [0.22368436, 0.88001406, -0.10369842], [-0.04107303, -0.01844361, 1.14857439]], [[1.70701129, -0.39308248, -0.21060088], [-0.42750858, 1.23694441, 0.1555323], [0.05417788, 0.00580601, 0.86561094]], log_encoding_SLog3, log_decoding_SLog3, False, False)**

Venice S-Gamut3.Cine colourspace.

References

[SonyECorporation20a]

colour.models.RGB_COLOURSPACE_sRGB

colour.models.**RGB_COLOURSPACE_sRGB = RGB_Colourspace('sRGB', [[0.64, 0.33], [0.3 , 0.6], [0.15, 0.06]], [0.3127, 0.329], 'D65', [[0.4124, 0.3576, 0.1805], [0.2126, 0.7152, 0.0722], [0.0193, 0.1192, 0.9505]], [[3.2406, -1.5372, -0.4986], [-0.9689, 1.8758, 0.0415], [0.0557, -0.204 , 1.057]], eotf_inverse_sRGB, eotf_sRGB, False, False)**

Smits (1999) colourspace.

References

[Smi99],

colour.models.RGB_COLOURSPACE_V_GAMUT

colour.models.**RGB_COLOURSPACE_V_GAMUT = RGB_Colourspace('V-Gamut', [[0.73 , 0.28], [0.165, 0.84], [0.1 , -0.03]], [0.3127, 0.329], 'D65', [[0.679644, 0.152211, 0.1186], [0.260686, 0.774894, -0.03558], [-0.00931 , -0.004612, 1.10298]], [[1.589012, -0.313204, -0.180965], [-0.534053, 1.396011, 0.102458], [0.011179, 0.003194, 0.905535]], log_encoding_VLog, log_decoding_VLog, False, False)**

Panasonic V-Gamut colourspace.

References

[Panasonic14]

colour.models.RGB_COLOURSPACE_XTREME_RGB

colour.models.**RGB_COLOURSPACE_XTREME_RGB = RGB_Colourspace('Xtreme RGB', [[1., 0.], [0., 1.], [0., 0.]], [0.3457, 0.3585], 'D50', [[0.96429568, 0. , 0.], [0. , 1. , 0.], [0. , 0. , 0.8251046]], [[1.03702632, 0. , 0.], [0. , 1. , 0.], [0. , 0. , 1.21196755]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Xtreme RGB colourspace.

References

[HutchColord]

Colour Component Transfer Functions

colour

cctf_encoding(value[, function])	Encode linear RGB values to non-linear $R'G'B'$ values using given encoding colour component transfer function (Encoding CCTF).
CCTF_ENCODINGS	Supported encoding colour component transfer functions (Encoding CCTFs), a collection of the functions defined by colour.LOG_ENCODINGS, colour.OETFS, colour.EOTF_INVERSES attributes, the colour.models.cctf_encoding_ProPhotoRGB(), colour.models.cctf_encoding_RIMMRGB(), colour.models.cctf_encoding_ROMMRGB() definitions and 3 gamma encoding functions (1 / 2.2, 1 / 2.4, 1 / 2.6).
cctf_decoding(value[, function])	Decode non-linear $R'G'B'$ values to linear RGB values using given decoding colour component transfer function (Decoding CCTF).
CCTF_DECODINGS	Supported decoding colour component transfer functions (Decoding CCTFs), a collection of the functions defined by colour.LOG_DECODINGS, colour.EOTFS, colour.OETF_INVERSES attributes, the colour.models.cctf_decoding_ProPhotoRGB(), colour.models.cctf_decoding_RIMMRGB(), colour.models.cctf_decoding_ROMMRGB() definitions and 3 gamma decoding functions (2.2, 2.4, 2.6).
gamma_function(a[, exponent, ...])	Define a typical gamma encoding / decoding function.
linear_function(a)	Define a typical linear encoding / decoding function, essentially a pass-through function.

colour.cctf_encoding

colour.**cctf_encoding**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[num bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'ARIB STD-B67', 'Blackmagic Film Generation 5', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'DCDM', 'DICOM GSDF', 'DaVinci Intermediate', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'Gamma 2.2', 'Gamma 2.4', 'Gamma 2.6', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'ProPhoto RGB', 'Protune', 'REDLog', 'REDLogFilm', 'RIMM RGB', 'ROMM RGB', 'S-Log', 'S-Log2', 'S-Log3', 'SMPTE 240M', 'ST 2084', 'T-Log', 'V-Log', 'ViperLog', 'sRGB'], str] = 'sRGB', **kwargs: Any*) → Union[numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]], numpy.ndarray[*Any*, numpy.dtype[Union[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64]]]]

Encode linear RGB values to non-linear $R'G'B'$ values using given encoding colour component transfer function (Encoding CCTF).

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy.

_typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Linear *RGB* values.

- **function** (Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'ARIB STD-B67', 'Blackmagic Film Generation 5', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'DCDM', 'DICOM GSDF', 'DaVinci Intermediate', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'Gamma 2.2', 'Gamma 2.4', 'Gamma 2.6', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'ProPhoto RGB', 'Protune', 'REDLog', 'REDLogFilm', 'RIMM RGB', 'ROMM RGB', 'S-Log', 'S-Log2', 'S-Log3', 'SMPTE 240M', 'ST 2084', 'T-Log', 'V-Log', 'ViperLog', 'sRGB'], str]) – {colour.CCTF_ENCODINGS}, Encoding colour component transfer function.
- **kwargs** (Any) – Keywords arguments for the relevant encoding *CCTF* of the colour.CCTF_ENCODINGS attribute collection.

Return type *Union*[numpy.ndarray[Any, numpy.dtype[*Union*[numpy.float16, numpy.float32, numpy.float64]]], numpy.ndarray[Any, numpy.dtype[*Union*[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64]]]]

> **Warning:** For *ITU-R BT.2100*, only the inverse electro-optical transfer functions (EOTFs / EOCFs) are exposed by this definition, See the colour.oetf() definition for the opto-electronic transfer functions (OETF).

Returns Non-linear $R'G'B'$ values.

Return type numpy.ndarray

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) –
- **function** (Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'ARIB STD-B67', 'Blackmagic Film Generation 5', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'DCDM', 'DICOM GSDF', 'DaVinci Intermediate', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'Gamma 2.2', 'Gamma 2.4', 'Gamma 2.6', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'ProPhoto RGB', 'Protune', 'REDLog', 'REDLogFilm', 'RIMM RGB', 'ROMM RGB', 'S-Log', 'S-Log2', 'S-Log3', 'SMPTE 240M', 'ST 2084', 'T-Log', 'V-Log', 'ViperLog', 'sRGB'], str]) –
- **kwargs** (Any) –

Examples

```
>>> cctf_encoding(0.18, function="PLog", log_reference=400)
...
0.3910068...
>>> cctf_encoding(0.18, function="ST 2084", L_p=1000)
...
0.1820115...
>>> cctf_encoding(
...     0.11699185725296059, function="ITU-R BT.1886"
... )
0.4090077...
```

colour.CCTF_ENCODINGS

colour.**CCTF_ENCODINGS = CanonicalMapping({'Gamma 2.2': ..., 'Gamma 2.4': ..., 'Gamma 2.6': ..., 'ProPhoto RGB': ..., 'RIMM RGB': ..., 'ROMM RGB': ..., 'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ARRI LogC3': ..., 'ARRI LogC4': ..., 'Canon Log 2': ..., 'Canon Log 3': ..., 'Canon Log': ..., 'Cineon': ..., 'D-Log': ..., 'ERIMM RGB': ..., 'F-Log': ..., 'F-Log2': ..., 'Filmic Pro 6': ..., 'L-Log': ..., 'Log2': ..., 'Log3G10': ..., 'Log3G12': ..., 'N-Log': ..., 'PLog': ..., 'Panalog': ..., 'Protune': ..., 'REDLog': ..., 'REDLogFilm': ..., 'S-Log': ..., 'S-Log2': ..., 'S-Log3': ..., 'T-Log': ..., 'V-Log': ..., 'ViperLog': ..., 'ARIB STD-B67': ..., 'Blackmagic Film Generation 5': ..., 'DaVinci Intermediate': ..., 'ITU-R BT.2020': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'ITU-T H.273 Log': ..., 'ITU-T H.273 Log Sqrt': ..., 'ITU-T H.273 IEC 61966-2': ..., 'SMPTE 240M': ..., 'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-T H.273 ST.428-1': ..., 'ST 2084': ..., 'sRGB': ...})**

Supported encoding colour component transfer functions (Encoding CCTFs), a collection of the functions defined by `colour.LOG_ENCODINGS`, `colour.OETFS`, `colour.EOTF_INVERSES` attributes, the `colour.models.cctf_encoding_ProPhotoRGB()`, `colour.models.cctf_encoding_RIMMRGB()`, `colour.models.cctf_encoding_ROMMRGB()` definitions and 3 gamma encoding functions (1 / 2.2, 1 / 2.4, 1 / 2.6).

> **Warning:** For *ITU-R BT.2100*, only the inverse electro-optical transfer functions (EOTFs / EOCFs) are exposed by this attribute, See the `colour.OETFS` attribute for the opto-electronic transfer functions (OETF).

colour.cctf_decoding

colour.**cctf_decoding**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[num bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'ARIB STD-B67', 'Blackmagic Film Generation 5', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'DCDM', 'DICOM GSDF', 'DaVinci Intermediate', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'Gamma 2.2', 'Gamma 2.4', 'Gamma 2.6', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'ProPhoto RGB', 'Protune', 'REDLog', 'REDLogFilm', 'RIMM RGB', 'ROMM RGB', 'S-Log', 'S-Log2', 'S-Log3', 'SMPTE 240M', 'ST 2084', 'T-Log', 'V-Log', 'ViperLog', 'sRGB'], str] = 'sRGB', **kwargs: Any*) → numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]

Decode non-linear $R'G'B'$ values to linear RGB values using given decoding colour component transfer function (Decoding CCTF).

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Non-linear $R'G'B'$ values.
- **function** (Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'ARIB STD-B67', 'Blackmagic Film Generation 5', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'DCDM', 'DICOM GSDF', 'DaVinci Intermediate', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'Gamma 2.2', 'Gamma 2.4', 'Gamma 2.6', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'ProPhoto RGB', 'Protune', 'REDLog', 'REDLogFilm', 'RIMM RGB', 'ROMM RGB', 'S-Log', 'S-Log2', 'S-Log3', 'SMPTE 240M', 'ST 2084', 'T-Log', 'V-Log', 'ViperLog', 'sRGB'], str]) – {colour.CCTF_DECODINGS}, Decoding colour component transfer function.
- **kwargs** (Any) – Keywords arguments for the relevant decoding *CCTF* of the colour.CCTF_DECODINGS attribute collection.

Return type numpy.ndarray[Any, numpy.dtype[*Union*[numpy.float16, numpy.float32, numpy.float64]]]

> **Warning:** For *ITU-R BT.2100*, only the electro-optical transfer functions (EOTFs / EOCFs) are exposed by this definition, See the colour.oetf_inverse() definition for the inverse opto-electronic transfer functions (OETF).

Returns Linear RGB values.

Return type numpy.ndarray

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) –
- **function** (Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'ARIB STD-B67', 'Blackmagic Film Generation 5', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'DCDM', 'DICOM GSDF', 'DaVinci Intermediate', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'Gamma 2.2', 'Gamma 2.4', 'Gamma 2.6', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'ProPhoto RGB', 'Protune', 'REDLog', 'REDLogFilm', 'RIMM RGB', 'ROMM RGB', 'S-Log', 'S-Log2', 'S-Log3', 'SMPTE 240M', 'ST 2084', 'T-Log', 'V-Log', 'ViperLog', 'sRGB'], str]) –
- **kwargs** (Any) –

Examples

```
>>> cctf_decoding(0.391006842619746, function="PLog", log_reference=400)
...
0.1...
>>> cctf_decoding(0.182011532850008, function="ST 2084", L_p=1000)
...
0.1...
>>> cctf_decoding(
...     0.461356129500442, function="ITU-R BT.1886"
... )
0.1...
```

colour.CCTF_DECODINGS

colour.**CCTF_DECODINGS = CanonicalMapping({'Gamma 2.2': ..., 'Gamma 2.4': ..., 'Gamma 2.6': ..., 'ProPhoto RGB': ..., 'RIMM RGB': ..., 'ROMM RGB': ..., 'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ARRI LogC3': ..., 'ARRI LogC4': ..., 'Canon Log 2': ..., 'Canon Log 3': ..., 'Canon Log': ..., 'Cineon': ..., 'D-Log': ..., 'ERIMM RGB': ..., 'F-Log': ..., 'F-Log2': ..., 'Filmic Pro 6': ..., 'L-Log': ..., 'Log2': ..., 'Log3G10': ..., 'Log3G12': ..., 'N-Log': ..., 'PLog': ..., 'Panalog': ..., 'Protune': ..., 'REDLog': ..., 'REDLogFilm': ..., 'S-Log': ..., 'S-Log2': ..., 'S-Log3': ..., 'T-Log': ..., 'V-Log': ..., 'ViperLog': ..., 'ARIB STD-B67': ..., 'Blackmagic Film Generation 5': ..., 'DaVinci Intermediate': ..., 'ITU-R BT.2020': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'ITU-T H.273 Log': ..., 'ITU-T H.273 Log Sqrt': ..., 'ITU-T H.273 IEC 61966-2': ..., 'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-T H.273 ST.428-1': ..., 'SMPTE 240M': ..., 'ST 2084': ..., 'sRGB': ...})**

Supported decoding colour component transfer functions (Decoding CCTFs), a collection of the functions defined by `colour.LOG_DECODINGS`, `colour.EOTFS`, `colour.OETF_INVERSES` attributes, the `colour.models.cctf_decoding_ProPhotoRGB()`, `colour.models.cctf_decoding_RIMMRGB()`, `colour.models.cctf_decoding_ROMMRGB()` definitions and 3 gamma decoding functions (2.2, 2.4, 2.6).

> **Warning:** For *ITU-R BT.2100*, only the electro-optical transfer functions (EOTFs / EOCFs) are exposed by this attribute, See the `colour.OETF_INVERSES` attribute for the inverse opto-electronic transfer functions (OETF).

Notes

- The order by which this attribute is defined and updated is critically important to ensure that *ITU-R BT.2100* definitions are reciprocal.

colour.gamma_function

colour.**gamma_function**(*a: ArrayLike*, *exponent: ArrayLike = 1*, *negative_number_handling: Union[Literal['Clamp', 'Indeterminate', 'Mirror', 'Preserve'], str] = 'Indeterminate'*) → NDArrayFloat

Define a typical gamma encoding / decoding function.

Parameters

- **a** (ArrayLike) – Array to encode / decode.
- **exponent** (ArrayLike) – Encoding / decoding exponent.
- **negative_number_handling** (Union[Literal['Clamp', 'Indeterminate', 'Mirror', 'Preserve'], str]) – 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 numpy.ndarray

Examples

```
>>> gamma_function(0.18, 2.2)
0.0229932...
>>> gamma_function(-0.18, 2.0)
0.0323999...
>>> gamma_function(-0.18, 2.2)
nan
>>> gamma_function(-0.18, 2.2, "Mirror")
-0.0229932...
>>> gamma_function(-0.18, 2.2, "Preserve")
-0.1...
>>> gamma_function(-0.18, 2.2, "Clamp")
0.0
```

colour.linear_function

colour.**linear_function**(*a: ArrayLike*) → NDArrayFloat

Define a typical linear encoding / decoding function, essentially a pass-through function.

Parameters **a** (ArrayLike) – Array to encode / decode.

Returns Encoded / decoded array.

Return type numpy.ndarray

Examples

```
>>> linear_function(0.18)
0.1799999...
```

colour.models

cctf_encoding_ROMMRGB(X[, bit_depth, out_int])	Define the *ROMM RGB* encoding colour component transfer function (Encoding CCTF).
cctf_decoding_ROMMRGB(X_p[, bit_depth, in_int])	Define the *ROMM RGB* decoding colour component transfer function (Encoding CCTF).
cctf_encoding_RIMMRGB(X[, bit_depth, ...])	Define the *RIMM RGB* encoding colour component transfer function (Encoding CCTF).
cctf_decoding_RIMMRGB(X_p[, bit_depth, ...])	Define the *RIMM RGB* decoding colour component transfer function (Encoding CCTF).

colour.models.cctf_encoding_ROMMRGB

colour.models.**cctf_encoding_ROMMRGB**(*X: ArrayLike*, *bit_depth: int = 8*, *out_int: bool = False*) → NDArrayReal

Define the *ROMM RGB* encoding colour component transfer function (Encoding CCTF).

Parameters

- **X** (ArrayLike) – Linear data X_{ROMM}.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_int** (bool) – Whether to return value as int code value or float equivalent of a code value at a given bit-depth.

Returns Non-linear data X'_{ROMM}.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

* This definition has an output int switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANSI03], [SWG00]

Examples

```
>>> cctf_encoding_ROMMRGB(0.18)
0.3857114...
>>> cctf_encoding_ROMMRGB(0.18, out_int=True)
98
```

colour.models.cctf_decoding_ROMMRGB

colour.models.**cctf_decoding_ROMMRGB**(*X_p: ArrayLike*, *bit_depth: int = 8*, *in_int: bool = False*) → NDArrayFloat

Define the *ROMM RGB* decoding colour component transfer function (Encoding CCTF).

Parameters

- **X_p** (ArrayLike) – Non-linear data X'_{ROMM}.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_int** (bool) – Whether to treat the input value as int code value or float equivalent of a code value at a given bit-depth.

Returns Linear data X_{ROMM}.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

* This definition has an input int switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANSI03], [SWG00]

Examples

```
>>> cctf_decoding_ROMMRGB(0.385711424751138)
0.1...
>>> cctf_decoding_ROMMRGB(98, in_int=True)
0.1...
```

colour.models.cctf_encoding_RIMMRGB

colour.models.**cctf_encoding_RIMMRGB**(*X: ArrayLike*, *bit_depth: int = 8*, *out_int: bool = False*, *E_clip: float = 2.0*) → NDArrayReal

Define the *RIMM RGB* encoding colour component transfer function (Encoding CCTF).

RIMM RGB encoding non-linearity is based on that specified by *Recommendation ITU-R BT.709-6*.

Parameters

- **X** (ArrayLike) – Linear data X_{RIMM}.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_int** (bool) – Whether to return value as int code value or float equivalent of a code value at a given bit-depth.
- **E_clip** (float) – Maximum exposure level.

Returns Non-linear data X'_{RIMM}.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

* This definition has an output int switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> cctf_encoding_RIMMRGB(0.18)
0.2916737...
>>> cctf_encoding_RIMMRGB(0.18, out_int=True)
74
```

colour.models.cctf_decoding_RIMMRGB

`colour.models.cctf_decoding_RIMMRGB(X_p: ArrayLike, bit_depth: int = 8, in_int: bool = False, E_clip: float = 2.0) → NDArrayFloat`

Define the *RIMM RGB* decoding colour component transfer function (Encoding CCTF).

Parameters

- **X_p** (ArrayLike) – Non-linear data X'_{RIMM}.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_int** (bool) – Whether to treat the input value as int code value or float equivalent of a code value at a given bit-depth.
- **E_clip** (float) – Maximum exposure level.

Returns Linear data X_{RIMM}.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

* This definition has an input int switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> cctf_decoding_RIMMRGB(0.291673732475746)
0.1...
>>> cctf_decoding_RIMMRGB(74, in_int=True)
0.1...
```

Aliases

colour.models

cctf_encoding_ProPhotoRGB(X[, bit_depth, ...])	Define the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).
cctf_decoding_ProPhotoRGB(X_p[, bit_depth, ...])	Define the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).

colour.models.cctf_encoding_ProPhotoRGB

colour.models.**cctf_encoding_ProPhotoRGB**(*X*, *bit_depth=8*, *out_int=False*)

Define the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).

Parameters

- **X** – Linear data X_{ROMM}.
- **bit_depth** – Bit-depth used for conversion.
- **out_int** – Whether to return value as int code value or float equivalent of a code value at a given bit-depth.

Returns Non-linear data X'_{ROMM}.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

* This definition has an output int switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANSI03], [SWG00]

Examples

```
>>> cctf_encoding_ROMMRGB(0.18)
0.3857114...
>>> cctf_encoding_ROMMRGB(0.18, out_int=True)
98
```

colour.models.cctf_decoding_ProPhotoRGB

colour.models.**cctf_decoding_ProPhotoRGB**(*X_p*, *bit_depth=8*, *in_int=False*)

Define the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).

Parameters

- **X_p** – Non-linear data X'_{ROMM}.
- **bit_depth** – Bit-depth used for conversion.
- **in_int** – Whether to treat the input value as int code value or float equivalent of a code value at a given bit-depth.

Returns Linear data X_{ROMM}.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

* This definition has an input int switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANSI03], [SWG00]

Examples

```
>>> cctf_decoding_ROMMRGB(0.385711424751138)
0.1...
>>> cctf_decoding_ROMMRGB(98, in_int=True)
0.1...
```

Ancillary Objects

colour.models

exponent_function_basic(x[, exponent, style])	Define the *basic* exponent transfer function.
exponent_function_monitor_curve(x[, ...])	Define the *Monitor Curve* exponent transfer function.
logarithmic_function_basic(x[, style, base])	Define the basic logarithmic function.
logarithmic_function_quasilog(x[, style, ...])	Define the quasilog logarithmic function.
logarithmic_function_camera(x[, style, ...])	Define the camera logarithmic function.

colour.models.exponent_function_basic

colour.models.**exponent_function_basic**(*x: ArrayLike, exponent: ArrayLike = 1, style: Union[Literal['basicFwd', 'basicRev', 'basicMirrorFwd', 'basicMirrorRev', 'basicPassThruFwd', 'basicPassThruRev'], str] = 'basicFwd'*) → NDArrayFloat

Define the *basic* exponent transfer function.

Parameters

- **x** (ArrayLike) – Data to undergo the basic exponent conversion.
- **exponent** (ArrayLike) – Exponent value used for the conversion.
- **style** (Union[Literal['basicFwd', 'basicRev', 'basicMirrorFwd', 'basicMirrorRev', 'basicPassThruFwd', 'basicPassThruRev'], str]) – Defines the behaviour for the transfer function to operate:
  - *basicFwd*: *Basic Forward* exponential behaviour where the definition applies a basic power law using the exponent. Values less than zero are clamped.
  - *basicRev*: *Basic Reverse* exponential behaviour where the definition applies a basic power law using the exponent. Values less than zero are clamped.

  - *basicMirrorFwd*: *Basic Mirror Forward* exponential behaviour where the definition applies a basic power law using the exponent for values greater than or equal to zero and mirrors the function for values less than zero (i.e. rotationally symmetric around the origin).
  - *basicMirrorRev*: *Basic Mirror Reverse* exponential behaviour where the definition applies a basic power law using the exponent for values greater than or equal to zero and mirrors the function for values less than zero (i.e. rotationally symmetric around the origin).
  - *basicPassThruFwd*: *Basic Pass Forward* exponential behaviour where the definition applies a basic power law using the exponent for values greater than or equal to zero and passes values less than zero unchanged.
  - *basicPassThruRev*: *Basic Pass Reverse* exponential behaviour where the definition applies a basic power law using the exponent for values greater than or equal to zero and passes values less than zero unchanged.

Returns Exponentially converted data.

Return type `numpy.ndarray`

Examples

```
>>> exponent_function_basic(0.18, 2.2)
0.0229932...
>>> exponent_function_basic(-0.18, 2.2)
0.0
>>> exponent_function_basic(0.18, 2.2, "basicRev")
0.4586564...
>>> exponent_function_basic(-0.18, 2.2, "basicRev")
0.0
>>> exponent_function_basic(
...     0.18, 2.2, "basicMirrorFwd"
... )
0.0229932...
>>> exponent_function_basic(
...     -0.18, 2.2, "basicMirrorFwd"
... )
-0.0229932...
>>> exponent_function_basic(
...     0.18, 2.2, "basicMirrorRev"
... )
0.4586564...
>>> exponent_function_basic(
...     -0.18, 2.2, "basicMirrorRev"
... )
-0.4586564...
>>> exponent_function_basic(
...     0.18, 2.2, "basicPassThruFwd"
... )
0.0229932...
>>> exponent_function_basic(
...     -0.18, 2.2, "basicPassThruFwd"
... )
-0.1799999...
>>> exponent_function_basic(
...     0.18, 2.2, "basicPassThruRev"
... )
```

(continues on next page)

(continued from previous page)

```
0.4586564...
>>> exponent_function_basic(
...     -0.18, 2.2, "basicPassThruRev"
... )
-0.1799999...
```

colour.models.exponent_function_monitor_curve

colour.models.**exponent_function_monitor_curve**(*x: ArrayLike, exponent: ArrayLike = 1, offset: ArrayLike = 0, style: Union[Literal['monCurveFwd', 'monCurveRev', 'monCurveMirrorFwd', 'monCurveMirrorRev'], str] = 'monCurveFwd'*) → NDArrayFloat

Define the *Monitor Curve* exponent transfer function.

Parameters

- **x** (ArrayLike) – Data to undergo the monitor curve exponential conversion.
- **exponent** (ArrayLike) – Exponent value used for the conversion.
- **offset** (ArrayLike) – Offset value used for the conversion.
- **style** (Union[Literal['monCurveFwd', 'monCurveRev', 'monCurveMirrorFwd', 'monCurveMirrorRev'], str]) – Defines the behaviour for the transfer function to operate:
  - *monCurveFwd*: *Monitor Curve Forward* exponential behaviour where the definition applies a power law function with a linear segment near the origin.
  - *monCurveRev*: *Monitor Curve Reverse* exponential behaviour where the definition applies a power law function with a linear segment near the origin.
  - *monCurveMirrorFwd*: *Monitor Curve Mirror Forward* exponential behaviour where the definition applies a power law function with a linear segment near the origin and mirrors the function for values less than zero (i.e. rotationally symmetric around the origin).
  - *monCurveMirrorRev*: *Monitor Curve Mirror Reverse* exponential behaviour where the definition applies a power law function with a linear segment near the origin and mirrors the function for values less than zero (i.e. rotationally symmetric around the origin).

Returns Exponentially converted data.

Return type numpy.ndarray

Examples

```
>>> exponent_function_monitor_curve(0.18, 2.2, 0.001)
0.0232240...
>>> exponent_function_monitor_curve(
...     -0.18, 2.2, 0.001
... )
-0.0002054...
>>> exponent_function_monitor_curve(
...     0.18, 2.2, 0.001, "monCurveRev"
... )
0.4581151...
```

(continues on next page)

(continued from previous page)

```
>>> exponent_function_monitor_curve(
...     -0.18, 2.2, 0.001, "monCurveRev"
... )
-157.7302795...
>>> exponent_function_monitor_curve(
...     0.18, 2.2, 2, "monCurveMirrorFwd"
... )
0.1679399...
>>> exponent_function_monitor_curve(
...     -0.18, 2.2, 0.001, "monCurveMirrorFwd"
... )
-0.0232240...
>>> exponent_function_monitor_curve(
...     0.18, 2.2, 0.001, "monCurveMirrorRev"
... )
0.4581151...
>>> exponent_function_monitor_curve(
...     -0.18, 2.2, 0.001, "monCurveMirrorRev"
... )
-0.4581151...
```

colour.models.logarithmic_function_basic

colour.models.**logarithmic_function_basic**(*x: ArrayLike, style: Union[Literal['log10', 'antiLog10', 'log2', 'antiLog2', 'logB', 'antiLogB'], str] = 'log2', base: int = 2*) → NDArrayFloat

Define the basic logarithmic function.

Parameters

- **x** (ArrayLike) – The data to undergo basic logarithmic conversion.
- **style** (Union[Literal['log10', 'antiLog10', 'log2', 'antiLog2', 'logB', 'antiLogB'], str]) – Defines the behaviour for the logarithmic function to operate:
  - *log10*: Applies a base 10 logarithm to the passed value.
  - *antiLog10*: Applies a base 10 anti-logarithm to the passed value.
  - *log2*: Applies a base 2 logarithm to the passed value.
  - *antiLog2*: Applies a base 2 anti-logarithm to the passed value.
  - *logB*: Applies an arbitrary base logarithm to the passed value.
  - *antiLogB*: Applies an arbitrary base anti-logarithm to the passed value.
- **base** (int) – Logarithmic base used for the conversion.

Returns Logarithmically converted data.

Return type numpy.ndarray

Examples

The basic logarithmic function *styles* operate as follows:

```
>>> logarithmic_function_basic(0.18)
-2.4739311...
>>> logarithmic_function_basic(0.18, "log10")
-0.7447274...
>>> logarithmic_function_basic(0.18, "logB", 3)
-1.5608767...
>>> logarithmic_function_basic(
...     -2.473931188332412, "antiLog2"
... )
0.18000000...
>>> logarithmic_function_basic(
...     -0.7447274948966939, "antiLog10"
... )
0.18000000...
>>> logarithmic_function_basic(
...     -1.5608767950073117, "antiLogB", 3
... )
0.18000000...
```

colour.models.logarithmic_function_quasilog

colour.models.**logarithmic_function_quasilog**(*x: ArrayLike, style: Union[Literal['linToLog', 'logToLin'], str] = 'linToLog', base: int = 2, log_side_slope: float = 1, lin_side_slope: float = 1, log_side_offset: float = 0, lin_side_offset: float = 0*) → NDArrayFloat

Define the quasilog logarithmic function.

Parameters

- **x** (ArrayLike) – Linear/non-linear data to undergo encoding/decoding.
- **style** (Union[Literal['linToLog', 'logToLin'], str]) – Defines the behaviour for the logarithmic function to operate:
  - *linToLog*: Applies a logarithm to convert linear data to logarithmic data.
  - *logToLin*: Applies an anti-logarithm to convert logarithmic data to linear data.
- **base** (int) – Logarithmic base used for the conversion.
- **log_side_slope** (float) – Slope (or gain) applied to the log side of the logarithmic function. The default value is 1.
- **lin_side_slope** (float) – Slope of the linear side of the logarithmic function. The default value is 1.
- **log_side_offset** (float) – Offset applied to the log side of the logarithmic function. The default value is 0.
- **lin_side_offset** (float) – Offset applied to the linear side of the logarithmic function. The default value is 0.

Returns Encoded/Decoded data.

Return type numpy.ndarray

Examples

```
>>> logarithmic_function_quasilog(0.18, "linToLog")
-2.4739311...
>>> logarithmic_function_quasilog(
...     -2.473931188332412, "logToLin"
... )
0.18000000...
```

colour.models.logarithmic_function_camera

colour.models.**logarithmic_function_camera**(*x: ArrayLike*, *style: Union[Literal['cameraLinToLog', 'cameraLogToLin'], str] = 'cameraLinToLog'*, *base: int = 2*, *log_side_slope: float = 1*, *lin_side_slope: float = 1*, *log_side_offset: float = 0*, *lin_side_offset: float = 0*, *lin_side_break: float = 0.005*, *linear_slope: float | None = None*) → NDArrayFloat

Define the camera logarithmic function.

Parameters

- **x** (ArrayLike) – Linear/non-linear data to undergo encoding/decoding.
- **style** (Union[Literal['cameraLinToLog', 'cameraLogToLin'], str]) – Defines the behaviour for the logarithmic function to operate:
  - *cameraLinToLog*: Applies a piece-wise function with logarithmic and linear segments on linear values, converting them to non-linear values.
  - *cameraLogToLin*: Applies a piece-wise function with logarithmic and linear segments on non-linear values, converting them to linear values.
- **base** (int) – Logarithmic base used for the conversion.
- **log_side_slope** (float) – Slope (or gain) applied to the log side of the logarithmic segment. The default value is 1.
- **lin_side_slope** (float) – Slope of the linear side of the logarithmic segment. The default value is 1.
- **log_side_offset** (float) – Offset applied to the log side of the logarithmic segment. The default value is 0.
- **lin_side_offset** (float) – Offset applied to the linear side of the logarithmic segment. The default value is 0.
- **lin_side_break** (float) – Break-point, defined in linear space, at which the piece-wise function transitions between the logarithmic and linear segments.
- **linear_slope** (float | None) – Slope of the linear portion of the curve. The default value is *None*.

Returns Encoded/Decoded data.

Return type numpy.ndarray

Examples

```
>>> logarithmic_function_camera(
...     0.18, "cameraLinToLog"
... )
-2.4739311...
>>> logarithmic_function_camera(
...     -2.4739311883324122, "cameraLogToLin"
... )
0.1800000...
```

Opto-Electronic Transfer Functions

colour

oetf(value[, function])	Encode estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given opto-electronic transfer function (OETF).
OETFS	Supported opto-electrical transfer functions (OETFs / OECFs).
oetf_inverse(value[, function])	Decode $R'G'B'$ video component signal value to tristimulus values at the display using given inverse opto-electronic transfer function (OETF).
OETF_INVERSES	Supported inverse opto-electrical transfer functions (OETFs / OECFs).

colour.oetf

colour.**oetf**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[A bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['ARIB STD-B67', 'Blackmagic Film Generation 5', 'DaVinci Intermediate', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'ITU-T H.273 Log', 'ITU-T H.273 Log Sqrt', 'ITU-T H.273 IEC 61966-2', 'SMPTE 240M'], str] = 'ITU-R BT.709', **kwargs: Any*) → numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]

Encode estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given opto-electronic transfer function (OETF).

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Value.
- **function** (Union[Literal['ARIB STD-B67', 'Blackmagic Film Generation 5', 'DaVinci Intermediate', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'ITU-T H.273 Log', 'ITU-T H.273 Log Sqrt', 'ITU-T H.273 IEC 61966-2', 'SMPTE 240M'], str]) – Opto-electronic transfer function (OETF).

- **kwargs** (Any) – {colour.models.oetf_ARIBSTDB67(), colour.models.oetf_BlackmagicFilmGeneration5(), colour.models.oetf_DaVinciIntermediate(), colour.models.oetf_BT2020(), colour.models.oetf_BT2100_HLG(), colour.models.oetf_BT2100_PQ(), colour.models.oetf_BT601(), colour.models.oetf_BT709(), colour.models.oetf_SMPTE240M()}, See the documentation of the previously listed definitions.

Returns $R'G'B'$ video component signal value.

Return type numpy.ndarray

Examples

```
>>> oetf(0.18)
0.4090077...
>>> oetf(0.18, function="ITU-R BT.601")
0.4090077...
```

colour.OETFS

colour.OETFS = CanonicalMapping({'ARIB STD-B67': ..., 'Blackmagic Film Generation 5': ..., 'DaVinci Intermediate': ..., 'ITU-R BT.2020': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'ITU-T H.273 Log': ..., 'ITU-T H.273 Log Sqrt': ..., 'ITU-T H.273 IEC 61966-2': ..., 'SMPTE 240M': ...})

Supported opto-electrical transfer functions (OETFs / OECFs).

colour.oetf_inverse

colour.**oetf_inverse**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['ARIB STD-B67', 'Blackmagic Film Generation 5', 'DaVinci Intermediate', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'ITU-T H.273 Log', 'ITU-T H.273 Log Sqrt', 'ITU-T H.273 IEC 61966-2'], str] = 'ITU-R BT.709', **kwargs: Any*) → numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]

Decode $R'G'B'$ video component signal value to tristimulus values at the display using given inverse opto-electronic transfer function (OETF).

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Value.
- **function** (Union[Literal['ARIB STD-B67', 'Blackmagic Film Generation 5', 'DaVinci Intermediate', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'ITU-T H.273 Log', 'ITU-T H.273 Log Sqrt', 'ITU-T H.273 IEC 61966-2'], str]) – Inverse opto-electronic transfer function (OETF).

- **kwargs** (Any) – {colour.models.oetf_inverse_ARIBSTDB67(), colour.models.oetf_inverse_BlackmagicFilmGeneration5(), colour.models.oetf_inverse_DaVinciIntermediate(), colour.models.oetf_inverse_BT2020(), colour.models.oetf_inverse_BT2100_HLG(), colour.models.oetf_inverse_BT2100_PQ(), colour.models.oetf_inverse_BT601(), colour.models.oetf_inverse_BT709()}, See the documentation of the previously listed definitions.

Returns Tristimulus values at the display.

Return type numpy.ndarray

Examples

```
>>> oetf_inverse(0.409007728864150)
0.1...
>>> oetf_inverse(
...     0.409007728864150, function="ITU-R BT.601"
... )
0.1...
```

colour.OETF_INVERSES

colour.OETF_INVERSES = CanonicalMapping({'ARIB STD-B67': ..., 'Blackmagic Film Generation 5': ..., 'DaVinci Intermediate': ..., 'ITU-R BT.2020': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'ITU-T H.273 Log': ..., 'ITU-T H.273 Log Sqrt': ..., 'ITU-T H.273 IEC 61966-2': ...})

Supported inverse opto-electrical transfer functions (OETFs / OECFs).

colour.models

oetf_ARIBSTDB67(E[, r, constants])	Define *ARIB STD-B67 (Hybrid Log-Gamma)* opto-electrical transfer function (OETF).
oetf_inverse_ARIBSTDB67(E_p[, r, constants])	Define *ARIB STD-B67 (Hybrid Log-Gamma)* inverse opto-electrical transfer function (OETF).
oetf_BlackmagicFilmGeneration5(x[, constants])	Define the *Blackmagic Film Generation 5* opto-electronic transfer function (OETF).
oetf_inverse_BlackmagicFilmGeneration5(y[, ...])	Define the *Blackmagic Film Generation 5* inverse opto-electronic transfer function (OETF).
oetf_DaVinciIntermediate(L[, constants])	Define the *DaVinci Intermediate* opto-electronic transfer function.
oetf_inverse_DaVinciIntermediate(V[, constants])	Define the *DaVinci Intermediate* inverse opto-electronic transfer function (OETF).
oetf_BT2020(E[, is_12_bits_system, constants])	Define *Recommendation ITU-R BT.2020* opto-electronic transfer function (OETF).
oetf_inverse_BT2020(E_p[, ...])	Define *Recommendation ITU-R BT.2020* inverse opto-electronic transfer function (OETF).
oetf_BT2100_HLG(E[, constants])	Define *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF).
oetf_inverse_BT2100_HLG(E_p[, constants])	Define *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-electrical transfer function (OETF).
oetf_BT2100_PQ(E)	Define *Recommendation ITU-R BT.2100 Reference PQ* opto-electrical transfer function (OETF).
oetf_inverse_BT2100_PQ(E_p)	Define *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-electrical transfer function (OETF).
oetf_BT601(L)	Define *Recommendation ITU-R BT.601-7* opto-electronic transfer function (OETF).
oetf_inverse_BT601(E)	Define *Recommendation ITU-R BT.601-7* inverse opto-electronic transfer function (OETF).
oetf_BT709(L)	Define *Recommendation ITU-R BT.709-6* opto-electronic transfer function (OETF).
oetf_inverse_BT709(V)	Define *Recommendation ITU-R BT.709-6* inverse opto-electronic transfer function (OETF).
oetf_H273_Log(L_c)	Define *Recommendation ITU-T H.273* opto-electronic transfer function (OETF) for logarithmic encoding (100:1 range).
oetf_inverse_H273_Log(V)	Define *Recommendation ITU-T H.273* inverse-opto-electronic transfer function (OETF) for logarithmic encoding (100:1 range).
oetf_H273_LogSqrt(L_c)	Define *Recommendation ITU-T H.273* opto-electronic transfer function (OETF) for logarithmic encoding (100*Sqrt(10):1 range).
oetf_inverse_H273_LogSqrt(V)	Define *Recommendation ITU-T H.273* inverse-opto-electronic transfer function (OETF) for logarithmic encoding (100*Sqrt(10):1 range).
oetf_H273_IEC61966_2(L_c)	Define *Recommendation ITU-T H.273* opto-electronic transfer function (OETF) for *IEC 61966-2* family of transfer functions (*2-1 sRGB*, *2-1 sYCC*, *2-4 xvYCC*).
oetf_inverse_H273_IEC61966_2(V)	Define *Recommendation ITU-T H.273* inverse opto-electronic transfer function (OETF) for *IEC 61966-2* family of transfer functions (*2-1 sRGB*, *2-1 sYCC*, *2-4 xvYCC*).
oetf_SMPTE240M(L_c)	Define *SMPTE 240M* opto-electrical transfer function (OETF).

colour.models.oetf_ARIBSTDB67

colour.models.**oetf_ARIBSTDB67**(*E: ArrayLike, r: ArrayLike = 0.5, constants: colour.utilities.data_structures.Structure = CONSTANTS_ARIBSTDB67*) → NDArrayFloat

Define *ARIB STD-B67 (Hybrid Log-Gamma)* opto-electrical transfer function (OETF).

Parameters

- **E** (ArrayLike) – Voltage normalised by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera color channel R, G, B.
- **r** (ArrayLike) – Video level corresponding to reference white level.
- **constants** (colour.utilities.data_structures.Structure) – *ARIB STD-B67 (Hybrid Log-Gamma)* constants.

Returns Resulting non-linear signal E'.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

- This definition uses the *mirror* negative number handling mode of `colour.models.gamma_function()` definition to the sign of negative numbers.

References

[AssociationoRIaBusinesses15]

Examples

```
>>> oetf_ARIBSTDB67(0.18)
0.2121320...
```

colour.models.oetf_inverse_ARIBSTDB67

colour.models.**oetf_inverse_ARIBSTDB67**(*E_p: ArrayLike, r: ArrayLike = 0.5, constants: colour.utilities.data_structures.Structure = CONSTANTS_ARIBSTDB67*) → NDArrayFloat

Define *ARIB STD-B67 (Hybrid Log-Gamma)* inverse opto-electrical transfer function (OETF).

Parameters

- **E_p** (ArrayLike) – Non-linear signal E'.
- **r** (ArrayLike) – Video level corresponding to reference white level.

- **constants** (colour.utilities.data_structures.Structure) – *ARIB STD-B67 (Hybrid Log-Gamma)* constants.

Returns Voltage E normalised by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera color channel R, G, B.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

- This definition uses the *mirror* negative number handling mode of `colour.models.gamma_function()` definition to the sign of negative numbers.

References

[AssociationoRIaBusinesses15]

Examples

```
>>> oetf_inverse_ARIBSTDB67(0.212132034355964)
0.1799999...
```

colour.models.oetf_BlackmagicFilmGeneration5

colour.models.**oetf_BlackmagicFilmGeneration5**(*x: ArrayLike*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_BLACKMAGIC_FILM_GENERATION_5*) → NDArrayFloat

Define the *Blackmagic Film Generation 5* opto-electronic transfer function (OETF).

Parameters

- **x** (ArrayLike) – Linear light value :math\`x\`.
- **constants** (colour.utilities.data_structures.Structure) – *Blackmagic Film Generation 5* constants.

Returns Encoded value y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[BlackmagicDesign21]

Examples

```
>>> oetf_BlackmagicFilmGeneration5(0.18)
0.3835616...
```

colour.models.oetf_inverse_BlackmagicFilmGeneration5

colour.models.**oetf_inverse_BlackmagicFilmGeneration5**(*y: ArrayLike, constants:* colour.utilities.data_structures.Structure = *CONSTANTS_BLACKMAGIC_FILM_GENERATION_5*) → NDArrayFloat

Define the *Blackmagic Film Generation 5* inverse opto-electronic transfer function (OETF).

Parameters

- **y** (ArrayLike) – Encoded value y.
- **constants** (colour.utilities.data_structures.Structure) – *Blackmagic Film Generation 5* constants.

Returns Linear light value :math`x`.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[BlackmagicDesign21]

Examples

```
>>> oetf_inverse_BlackmagicFilmGeneration5(0.38356164383561653)
...
0.1799999...
```

colour.models.oetf_DaVinciIntermediate

colour.models.**oetf_DaVinciIntermediate**(*L: ArrayLike*, *constants:* colour.utilities.data_structures.Structure = *CONSTANTS_DAVINCI_INTERMEDIATE*) → NDArrayFloat

Define the *DaVinci Intermediate* opto-electronic transfer function.

Parameters

- **L** (ArrayLike) – Linear light value :math`L`.
- **constants** (colour.utilities.data_structures.Structure) – *DaVinci Intermediate* colour component transfer function constants.

Returns Encoded value V.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[BlackmagicDesign20a]

Examples

```
>>> oetf_DaVinciIntermediate(0.18)
0.3360432...
```

colour.models.oetf_inverse_DaVinciIntermediate

colour.models.**oetf_inverse_DaVinciIntermediate**(*V: ArrayLike*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_DAVINCI_INTERMEDIATE*) → NDArrayFloat

Define the *DaVinci Intermediate* inverse opto-electronic transfer function (OETF).

Parameters

- **V** (ArrayLike) – Encoded value V.
- **constants** (colour.utilities.data_structures.Structure) – *DaVinci Intermediate* colour component transfer function constants.

Returns Linear light value :math`L`.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[BlackmagicDesign20a]

Examples

```
>>> oetf_inverse_DaVinciIntermediate(0.336043272384855)
...
0.1799999...
```

colour.models.oetf_BT2020

colour.models.**oetf_BT2020**(*E: ArrayLike*, *is_12_bits_system: bool = False*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_BT2020*) → NDArrayFloat

Define *Recommendation ITU-R BT.2020* opto-electronic transfer function (OETF).

Parameters

- **E** (ArrayLike) – Voltage E normalised by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera colour channel R, G, B.
- **is_12_bits_system** (bool) – *BT.709 alpha* and *beta* constants are used if system is not 12-bit.
- **constants** (colour.utilities.data_structures.Structure) – *Recommendation ITU-R BT.2020* constants.

Returns Resulting non-linear signal E'.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[InternationalTUnion15a]

Examples

```
>>> oetf_BT2020(0.18)
0.4090077...
```

colour.models.oetf_inverse_BT2020

colour.models.**oetf_inverse_BT2020**(*E_p: ArrayLike, is_12_bits_system: bool = False, constants: colour.utilities.data_structures.Structure = CONSTANTS_BT2020*) → NDArrayFloat

Define *Recommendation ITU-R BT.2020* inverse opto-electronic transfer function (OETF).

Parameters

- **E_p** (ArrayLike) – Non-linear signal E'.
- **is_12_bits_system** (bool) – *BT.709 alpha* and *beta* constants are used if system is not 12-bit.
- **constants** (colour.utilities.data_structures.Structure) – *Recommendation ITU-R BT.2020* constants.

Returns Resulting voltage E.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[InternationalTUnion15a]

Examples

```
>>> oetf_inverse_BT2020(0.705515089922121)
0.4999999...
```

colour.models.oetf_BT2100_HLG

colour.models.**oetf_BT2100_HLG**(*E: ArrayLike*, *constants:* colour.utilities.data_structures.Structure = *CONSTANTS_BT2100_HLG*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF).

The OETF maps relative scene linear light into the non-linear *HLG* signal value.

Parameters

- **E** (ArrayLike) – E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.
- **constants** (colour.utilities.data_structures.Structure) – *Recommendation ITU-R BT.2100 Reference HLG* constants.

Returns E' is the resulting non-linear signal R', G', B'.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> oetf_BT2100_HLG(0.18 / 12)
0.2121320...
```

colour.models.oetf_inverse_BT2100_HLG

colour.models.**oetf_inverse_BT2100_HLG**(*E_p: ArrayLike, constants:* colour.utilities.data_structures.Structure = *CONSTANTS_BT2100_HLG*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-electrical transfer function (OETF).

Parameters

- **E_p** (ArrayLike) – E' is the resulting non-linear signal R', G', B'.
- **constants** (colour.utilities.data_structures.Structure) – *Recommendation ITU-R BT.2100 Reference HLG* constants.

Returns E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> oetf_inverse_BT2100_HLG(0.212132034355964)
0.0149999...
```

colour.models.oetf_BT2100_PQ

colour.models.**oetf_BT2100_PQ**(*E: ArrayLike*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference PQ* opto-electrical transfer function (OETF).

The OETF maps relative scene linear light into the non-linear *PQ* signal value.

Parameters **E** (ArrayLike) – $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure.

Returns E' is the resulting non-linear signal (R', G', B').

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	UN	UN

Range	Scale - Reference	Scale - 1
E_p	UN	UN

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> oetf_BT2100_PQ(0.1)
0.7247698...
```

colour.models.oetf_inverse_BT2100_PQ

colour.models.**oetf_inverse_BT2100_PQ**(*E_p: ArrayLike*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-electrical transfer function (OETF).

Parameters **E_p** (ArrayLike) – E' is the resulting non-linear signal (R', G', B').

Returns $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	UN	UN

Range	Scale - Reference	Scale - 1
E	UN	UN

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> oetf_inverse_BT2100_PQ(0.724769816665726)
0.0999999...
```

colour.models.oetf_BT601

colour.models.**oetf_BT601**(*L: ArrayLike*) → NDArrayFloat

Define *Recommendation ITU-R BT.601-7* opto-electronic transfer function (OETF).

Parameters **L** (ArrayLike) – *Luminance* L of the image.

Returns Corresponding electrical signal E.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[InternationalTUnion11b]

Examples

```
>>> oetf_BT601(0.18)
0.4090077...
```

colour.models.oetf_inverse_BT601

colour.models.**oetf_inverse_BT601**(*E: ArrayLike*) → NDArrayFloat

Define *Recommendation ITU-R BT.601-7* inverse opto-electronic transfer function (OETF).

Parameters **E** (ArrayLike) – Electrical signal E.

Returns Corresponding *luminance* L of the image.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[InternationalTUnion11b]

Examples

```
>>> oetf_inverse_BT601(0.409007728864150)
0.1...
```

colour.models.oetf_BT709

colour.models.**oetf_BT709**(*L: ArrayLike*) → NDArrayFloat

Define *Recommendation ITU-R BT.709-6* opto-electronic transfer function (OETF).

Parameters **L** (ArrayLike) – *Luminance* L of the image.

Returns Corresponding electrical signal V.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[InternationalTUnion15b]

Examples

```
>>> oetf_BT709(0.18)
0.4090077...
```

colour.models.oetf_inverse_BT709

colour.models.**oetf_inverse_BT709**(*V: ArrayLike*) → NDArrayFloat

Define *Recommendation ITU-R BT.709-6* inverse opto-electronic transfer function (OETF).

Parameters **V** (ArrayLike) – Electrical signal V.

Returns Corresponding *luminance* L of the image.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[InternationalTUnion15b]

Examples

```
>>> oetf_inverse_BT709(0.409007728864150)
0.1...
```

colour.models.oetf_H273_Log

colour.models.**oetf_H273_Log**(*L_c*)

Define *Recommendation ITU-T H.273* opto-electronic transfer function (OETF) for logarithmic encoding (100:1 range).

Parameters **L_c** – Scene *Luminance* L_c.

Returns Corresponding electrical signal V.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[InternationalTUnion21]

Warning:

- The function is clamped to domain [0.01, np.inf].

Examples

```
>>> oetf_H273_Log(0.18)
0.6276362525516...
>>> oetf_H273_Log(0.01)
0.0
>>> oetf_H273_Log(0.001)
0.0
>>> oetf_H273_Log(1.0)
1.0
```

colour.models.oetf_inverse_H273_Log

colour.models.**oetf_inverse_H273_Log**(*V*)

Define *Recommendation ITU-T H.273* inverse-opto-electronic transfer function (OETF) for logarithmic encoding (100:1 range).

Parameters **V** – Electrical signal V.

Returns Corresponding scene *Luminance* L_c.

Return type numpy.ndarray

Notes

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Domain	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

References

[InternationalTUnion21]

Warning:

- The function is clamped to domain [colour.models.oetf_H273_Log() (0.01), np.inf].

Examples

```
>>> oetf_inverse_H273_Log(0.6276362525516)
0.17999999...
>>> oetf_inverse_H273_Log(0.0)
0.01
>>> oetf_inverse_H273_Log(1.0)
1.0
```

colour.models.oetf_H273_LogSqrt

colour.models.**oetf_H273_LogSqrt**(*L_c*)

Define *Recommendation ITU-T H.273* opto-electronic transfer function (OETF) for logarithmic encoding (100*Sqrt(10):1 range).

Parameters **L_c** – Scene *Luminance* L_c.

Returns Corresponding electrical signal V.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[InternationalTUnion21]

Warning:

- The function is clamped to domain [colour.models.oetf_H273_LogSqrt() (sqrt(10) / 1000), np.inf].

Examples

```
>>> oetf_H273_LogSqrt(0.18)
0.702109002041...
>>> oetf_H273_LogSqrt(0.003162277660168)
0.0
>>> oetf_H273_LogSqrt(0.0001)
0.0
>>> oetf_H273_LogSqrt(1.0)
1.0
```

colour.models.oetf_inverse_H273_LogSqrt

colour.models.**oetf_inverse_H273_LogSqrt**(*V*)

Define *Recommendation ITU-T H.273* inverse-opto-electronic transfer function (OETF) for logarithmic encoding (100*Sqrt(10):1 range).

Parameters **V** – Electrical signal V.

Returns Corresponding scene *Luminance* L_c.

Return type numpy.ndarray

Notes

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Domain	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

References

[InternationalTUnion21]

Warning:

- The function is clamped to domain [sqrt(10) / 1000, np.inf].

Examples

```
>>> oetf_inverse_H273_LogSqrt(0.702109002041)
0.1799999999...
>>> oetf_inverse_H273_LogSqrt(0.0)
0.00316227766...
>>> oetf_inverse_H273_LogSqrt(1.0)
1.0
```

colour.models.oetf_H273_IEC61966_2

colour.models.**oetf_H273_IEC61966_2**(*L_c*)

Define *Recommendation ITU-T H.273* opto-electronic transfer function (OETF) for *IEC 61966-2* family of transfer functions (*2-1 sRGB*, *2-1 sYCC*, *2-4 xvYCC*).

Parameters **L_c** – Scene *Luminance* L_c.

Returns Corresponding electrical signal V.

Return type numpy.ndarray

Notes

Usage in [InternationalTUnion21] is as follows:

- For *IEC 61966-2-1 sRGB (MatrixCoefficients=0)*, the function is only defined for L_c in [0-1] range.
- For *IEC 61966-2-1 sYCC (MatrixCoefficients=5)* and *IEC 61966-2-4 xvYCC*, the function is defined for any real-valued L_c.

Domain	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

- [InternationalTUnion21]

Examples

```
>>> oetf_H273_IEC61966_2(0.18)
0.4613561295004...
>>> oetf_H273_IEC61966_2(-0.18)
-0.4613561295004...
```

colour.models.oetf_inverse_H273_IEC61966_2

colour.models.**oetf_inverse_H273_IEC61966_2**(*V*)

Define *Recommendation ITU-T H.273* inverse opto-electronic transfer function (OETF) for *IEC 61966-2* family of transfer functions (*2-1 sRGB*, *2-1 sYCC*, *2-4 xvYCC*).

Parameters **V** – Electrical signal V.

Returns Corresponding scene luminance L_c.

Return type numpy.ndarray

Notes

Usage in [InternationalTUnion21] is as follows:

- For *IEC 61966-2-1 sRGB (MatrixCoefficients=0)*, the function is only defined for L_c in [0-1] range.
- For *IEC 61966-2-1 sYCC (MatrixCoefficients=5)* and *IEC 61966-2-4 xvYCC*, the function is defined for any real-valued L_c.

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Domain	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

References

- [InternationalTUnion21]

Examples

```
>>> oetf_inverse_H273_IEC61966_2(0.461356129500)
0.1799999999...
>>> oetf_inverse_H273_IEC61966_2(-0.461356129500)
-0.1799999999...
```

colour.models.oetf_SMPTE240M

colour.models.**oetf_SMPTE240M**(*L_c: ArrayLike*) → NDArrayFloat

Define *SMPTE 240M* opto-electrical transfer function (OETF).

Parameters **L_c** (ArrayLike) – Light input L_c to the reference camera normalised to the system reference white.

Returns Video signal output V_c of the reference camera normalised to the system reference white.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V_c	[0, 1]	[0, 1]

References

[SocietyoMPaTEngineers99]

Examples

```
>>> oetf_SMPTE240M(0.18)
0.4022857...
```

Electro-Optical Transfer Functions

colour

eotf(value[, function])	Decode $R'G'B'$ video component signal value to tristimulus values at the display using given electro-optical transfer function (EOTF).
EOTFS	Supported electro-optical transfer functions (EOTFs / EOCFs).
eotf_inverse(value[, function])	Encode estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given inverse electro-optical transfer function (EOTF).
EOTF_INVERSES	Supported inverse electro-optical transfer functions (EOTFs / EOCFs).

colour.eotf

colour.**eotf**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[A bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['DCDM', 'DICOM GSDF', 'ITU-R BT.1886', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-T H.273 ST.428-1', 'SMPTE 240M', 'ST 2084', 'sRGB'], str] = 'ITU-R BT.1886', **kwargs: Any*) → numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]

Decode $R'G'B'$ video component signal value to tristimulus values at the display using given electro-optical transfer function (EOTF).

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Value.
- **function** (Union[Literal['DCDM', 'DICOM GSDF', 'ITU-R BT.1886', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-T H.273 ST.428-1', 'SMPTE 240M', 'ST 2084', 'sRGB'], str]) – Electro-optical transfer function (EOTF).
- **kwargs** (Any) – {colour.models.eotf_DCDM(), colour.models.eotf_DICOMGSDF(), colour.models.eotf_BT1886(), colour.models.eotf_BT2100_HLG(), colour.models.eotf_BT2100_PQ(), colour.models.eotf_SMPTE240M(), colour.models.eotf_ST2084(), colour.models.eotf_sRGB()}, See the documentation of the previously listed definitions.

Returns Tristimulus values at the display.

Return type numpy.ndarray

Examples

```
>>> eotf(0.461356129500442)
0.1...
>>> eotf(0.182011532850008, function="ST 2084", L_p=1000)
...
0.1...
```

colour.EOTFS

colour.**EOTFS = CanonicalMapping({'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-T H.273 ST.428-1': ..., 'SMPTE 240M': ..., 'ST 2084': ..., 'sRGB': ...})**

Supported electro-optical transfer functions (EOTFs / EOCFs).

colour.eotf_inverse

colour.**eotf_inverse**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[nump bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['DCDM', 'DICOM GSDF', 'ITU-R BT.1886', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-T H.273 ST.428-1', 'ST 2084', 'sRGB'], str] = 'ITU-R BT.1886', **kwargs*) → Union[numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]], numpy.ndarray[*Any*, numpy.dtype[Union[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64]]]]

Encode estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given inverse electro-optical transfer function (EOTF).

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Value.
- **function** (Union[Literal['DCDM', 'DICOM GSDF', 'ITU-R BT.1886', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-T H.273 ST.428-1', 'ST 2084', 'sRGB'], str]) – Inverse electro-optical transfer function (EOTF).
- **kwargs** – {colour.models.eotf_inverse_DCDM(), colour.models.eotf_inverse_DICOMGSDF(), colour.models.eotf_inverse_BT1886(), colour.models.eotf_inverse_BT2100_HLG(), colour.models.eotf_inverse_BT2100_PQ(), colour.models.eotf_inverse_ST2084(), colour.models.eotf_inverse_sRGB()}, See the documentation of the previously listed definitions.

Returns $R'G'B'$ video component signal value.

Return type numpy.ndarray

Examples

```
>>> eotf_inverse(0.11699185725296059)
0.4090077...
>>> eotf_inverse(
...     0.11699185725296059, function="ITU-R BT.1886"
... )
0.4090077...
```

colour.EOTF_INVERSES

colour.EOTF_INVERSES = CanonicalMapping({'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-T H.273 ST.428-1': ..., 'ST 2084': ..., 'sRGB': ...})

Supported inverse electro-optical transfer functions (EOTFs / EOCFs).

colour.models

eotf_DCDM(XYZ_p[, in_int])	Define the *DCDM* electro-optical transfer function (EOTF).
eotf_inverse_DCDM(XYZ[, out_int])	Define the *DCDM* inverse electro-optical transfer function (EOTF).
eotf_DICOMGSDF(J[, in_int, constants])	Define the *DICOM - Grayscale Standard Display Function* electro-optical transfer function (EOTF).
eotf_inverse_DICOMGSDF(L[, out_int, constants])	Define the *DICOM - Grayscale Standard Display Function* inverse electro-optical transfer function (EOTF).
eotf_BT1886(V[, L_B, L_W])	Define *Recommendation ITU-R BT.1886* electro-optical transfer function (EOTF).
eotf_inverse_BT1886(L[, L_B, L_W])	Define *Recommendation ITU-R BT.1886* inverse electro-optical transfer function (EOTF).
BT2100_HLG_EOTF_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF).
eotf_BT2100_HLG(E_p[, L_B, L_W, gamma, ...])	Define *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF).
BT2100_HLG_EOTF_INVERSE_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF).
eotf_inverse_BT2100_HLG(F_D[, L_B, L_W, ...])	Define *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF).
eotf_BT2100_PQ(E_p)	Define *Recommendation ITU-R BT.2100 Reference PQ* electro-optical transfer function (EOTF).
eotf_inverse_BT2100_PQ(F_D)	Define *Recommendation ITU-R BT.2100 Reference PQ* inverse electro-optical transfer function (EOTF).
eotf_H273_ST428_1(V)	Define the *SMPTE ST 428-1 (2019)* electro-optical transfer function (EOTF).
eotf_inverse_H273_ST428_1(L_o)	Define *Recommendation ITU-T H.273* inverse electro-optical transfer function (EOTF) for *SMPTE ST 428-1 (2019)*.
eotf_SMPTE240M(V_r)	Define *SMPTE 240M* electro-optical transfer function (EOTF).
eotf_ST2084(N[, L_p, constants])	Define *SMPTE ST 2084:2014* optimised perceptual electro-optical transfer function (EOTF).
eotf_inverse_ST2084(C[, L_p, constants])	Define *SMPTE ST 2084:2014* optimised perceptual inverse electro-optical transfer function (EOTF).
eotf_sRGB(V)	Define the *IEC 61966-2-1:1999 sRGB* electro-optical transfer function (EOTF).
eotf_inverse_sRGB(L)	Define the *IEC 61966-2-1:1999 sRGB* inverse electro-optical transfer function (EOTF).

colour.models.eotf_DCDM

`colour.models.eotf_DCDM(XYZ_p: ArrayLike, in_int: bool = False) → NDArrayFloat`

Define the *DCDM* electro-optical transfer function (EOTF).

Parameters

- **XYZ_p** (`ArrayLike`) – Non-linear *CIE XYZ'* tristimulus values.
- **in_int** (`bool`) – Whether to treat the input value as int code value or float equivalent of a code value at a given bit-depth.

Returns *CIE XYZ* tristimulus values.

Return type `numpy.ndarray`

Warning: *DCDM* is an absolute transfer function.

Notes

- *DCDM* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain *	Scale - Reference	Scale - 1
XYZ_p	UN	UN

Range *	Scale - Reference	Scale - 1
XYZ	UN	UN

* This definition has an input int switch, thus the domain-range scale information is only given for the floating point mode.

References

[DigitalCInitiatives07]

Examples

```
>>> eotf_DCDM(0.11281860951766724)
0.18...
>>> eotf_DCDM(462, in_int=True)
0.18...
```

colour.models.eotf_inverse_DCDM

colour.models.**eotf_inverse_DCDM**(*XYZ: ArrayLike*, *out_int: bool = False*) → NDArrayReal

Define the *DCDM* inverse electro-optical transfer function (EOTF).

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **out_int** (bool) – Whether to return value as int code value or float equivalent of a code value at a given bit-depth.

Returns Non-linear *CIE XYZ'* tristimulus values.

Return type numpy.ndarray

Warning: *DCDM* is an absolute transfer function.

Notes

- *DCDM* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain *	Scale - Reference	Scale - 1
XYZ	UN	UN

Range *	Scale - Reference	Scale - 1
XYZ_p	UN	UN

* This definition has an output int switch, thus the domain-range scale information is only given for the floating point mode.

References

[DigitalCInitiatives07]

Examples

```
>>> eotf_inverse_DCDM(0.18)
0.1128186...
>>> eotf_inverse_DCDM(0.18, out_int=True)
462
```

colour.models.eotf_DICOMGSDF

colour.models.**eotf_DICOMGSDF**(*J: ArrayLike*, *in_int: bool = False*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_DICOMGSDF*) → NDArrayFloat

Define the *DICOM - Grayscale Standard Display Function* electro-optical transfer function (EOTF).

Parameters

- **J** (ArrayLike) – Just-Noticeable Difference (JND) Index, j.
- **in_int** (bool) – Whether to treat the input value as int code value or float equivalent of a code value at a given bit-depth.
- **constants** (colour.utilities.data_structures.Structure) – *DICOM - Grayscale Standard Display Function* constants.

Returns Corresponding *luminance* L.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
J	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[NationalEMAssociation04]

Examples

```
>>> eotf_DICOMGSDF(0.500486263438448)
130.0628647...
>>> eotf_DICOMGSDF(512, in_int=True)
130.0652840...
```

colour.models.eotf_inverse_DICOMGSDF

colour.models.**eotf_inverse_DICOMGSDF**(*L: ArrayLike*, *out_int: bool = False*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_DICOMGSDF*) → NDArrayReal

Define the *DICOM - Grayscale Standard Display Function* inverse electro-optical transfer function (EOTF).

Parameters

- **L** (ArrayLike) – *Luminance* L.
- **out_int** (bool) – Whether to return value as int code value or float equivalent of a code value at a given bit-depth.

- **constants** (colour.utilities.data_structures.Structure) – *DICOM - Grayscale Standard Display Function* constants.

Returns Just-Noticeable Difference (JND) Index, j.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
J	[0, 1]	[0, 1]

References

[NationalEMAssociation04]

Examples

```
>>> eotf_inverse_DICOMGSDF(130.0662)
0.5004862...
>>> eotf_inverse_DICOMGSDF(130.0662, out_int=True)
512
```

colour.models.eotf_BT1886

colour.models.**eotf_BT1886**(*V: ArrayLike, L_B: float = 0, L_W: float = 1*) → NDArrayFloat

Define *Recommendation ITU-R BT.1886* electro-optical transfer function (EOTF).

Parameters

- **V** (ArrayLike) – Input video signal level (normalised, black at $V = 0$, to white at $V = 1$. For content mastered per *Recommendation ITU-R BT.709*, 10-bit digital code values D map into values of V per the following equation: $V = (D - 64)/876$
- **L_B** (float) – Screen luminance for black.
- **L_W** (float) – Screen luminance for white.

Returns Screen luminance in cd/m^2.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[InternationalTUnion11a]

Examples

```
>>> eotf_BT1886(0.409007728864150)
0.1169918...
```

colour.models.eotf_inverse_BT1886

colour.models.**eotf_inverse_BT1886**(*L: ArrayLike*, *L_B: float = 0*, *L_W: float = 1*) → NDArrayFloat

Define *Recommendation ITU-R BT.1886* inverse electro-optical transfer function (EOTF).

Parameters

- **L** (ArrayLike) – Screen luminance in cd/m^2.
- **L_B** (float) – Screen luminance for black.
- **L_W** (float) – Screen luminance for white.

Returns Input video signal level (normalised, black at $V = 0$, to white at $V = 1$.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[InternationalTUnion11a]

Examples

```
>>> eotf_inverse_BT1886(0.11699185725296059)
0.4090077...
```

colour.models.BT2100_HLG_EOTF_METHODS

colour.models.**BT2100_HLG_EOTF_METHODS = CanonicalMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2': ...})**

Supported *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF).

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

colour.models.eotf_BT2100_HLG

colour.models.**eotf_BT2100_HLG**(*E_p: ArrayLike, L_B: float = 0, L_W: float = 1000, gamma: float | None = None, constants: colour.utilities.data_structures.Structure = CONSTANTS_BT2100_HLG, method: Union[Literal['ITU-R BT.2100-1', 'ITU-R BT.2100-2'], str] = 'ITU-R BT.2100-2'*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF).

The EOTF maps the non-linear *HLG* signal into display light.

Parameters

- **E_p** (ArrayLike) – E' denotes a non-linear colour value R', G', B' or L', M', S' in *HLG* space.
- **L_B** (float) – L_B is the display luminance for black in cd/m^2.
- **L_W** (float) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (float | None) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **constants** (colour.utilities.data_structures.Structure) – *Recommendation ITU-R BT.2100 Reference HLG* constants.
- **method** (Union[Literal['ITU-R BT.2100-1', 'ITU-R BT.2100-2'], str]) – Computation method.

Returns Luminance F_D of a displayed linear component R_D, G_D, B_D or Y_D or I_D, in cd/m^2.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

Examples

```
>>> eotf_BT2100_HLG(0.212132034355964)
6.4760398...
>>> eotf_BT2100_HLG(0.212132034355964, method="ITU-R BT.2100-1")
...
6.4760398...
>>> eotf_BT2100_HLG(0.212132034355964, 0.01)
...
7.3321975...
```

colour.models.BT2100_HLG_EOTF_INVERSE_METHODS

colour.models.**BT2100_HLG_EOTF_INVERSE_METHODS = CanonicalMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2': ...})**

Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF).

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

colour.models.eotf_inverse_BT2100_HLG

colour.models.**eotf_inverse_BT2100_HLG**(*F_D: ArrayLike, L_B: float = 0, L_W: float = 1000, gamma: float | None = None, constants: colour.utilities.data_structures.Structure = CONSTANTS_BT2100_HLG, method: Union[Literal['ITU-R BT.2100-1', 'ITU-R BT.2100-2'], str] = 'ITU-R BT.2100-2'*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF).

Parameters

- **F_D** (ArrayLike) – 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** (float) – L_B is the display luminance for black in cd/m^2.

- **L_W** (`float`) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (`float | None`) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **constants** (`colour.utilities.data_structures.Structure`) – *Recommendation ITU-R BT.2100 Reference HLG* constants.
- **method** (`Union[Literal['ITU-R BT.2100-1', 'ITU-R BT.2100-2'], str]`) – Computation method.

Returns E' denotes a non-linear colour value R', G', B' or L', M', S' in *HLG* space.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

Examples

```
>>> eotf_inverse_BT2100_HLG(6.476039825649814)
0.2121320...
>>> eotf_inverse_BT2100_HLG(6.476039825649814, method="ITU-R BT.2100-1")
...
0.2121320...
>>> eotf_inverse_BT2100_HLG(7.332197528353875, 0.01)
0.2121320...
```

colour.models.eotf_BT2100_PQ

colour.models.**eotf_BT2100_PQ**(*E_p: ArrayLike*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference PQ* electro-optical transfer function (EOTF).

The EOTF maps the non-linear *PQ* signal into display light.

Parameters E_p (`ArrayLike`) – 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 `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
E_p	UN	UN

Range	Scale - Reference	Scale - 1
F_D	UN	UN

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> eotf_BT2100_PQ(0.724769816665726)
779.9883608...
```

colour.models.eotf_inverse_BT2100_PQ

colour.models.**eotf_inverse_BT2100_PQ**(*F_D: ArrayLike*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference PQ* inverse electro-optical transfer function (EOTF).

Parameters **F_D** (ArrayLike) – 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 numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	UN	UN

Range	Scale - Reference	Scale - 1
E_p	UN	UN

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> eotf_inverse_BT2100_PQ(779.988360834085370)
0.7247698...
```

colour.models.eotf_H273_ST428_1

colour.models.**eotf_H273_ST428_1**(*V*)

Define the *SMPTE ST 428-1 (2019)* electro-optical transfer function (EOTF).

Parameters V – Electrical signal V.

Returns Corresponding output display *Luminance* L_o of the image.

Return type numpy.ndarray

Notes

- The function given in [InternationalTUnion21] divides L_o by 48 contrary to what is given in [SocietyoMPaTEngineers19] and colour.models.eotf_DCDM().

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Domain	Scale - Reference	Scale - 1
L_o	[0, 1]	[0, 1]

References

- [InternationalTUnion21], [SocietyoMPaTEngineers19]

Examples

```
>>> eotf_H273_ST428_1(0.5000483377172)
0.1799999...
```

colour.models.eotf_inverse_H273_ST428_1

colour.models.**eotf_inverse_H273_ST428_1**(*L_o*)

Define *Recommendation ITU-T H.273* inverse electro-optical transfer function (EOTF) for *SMPTE ST 428-1 (2019)*.

Parameters L_o – Output display *Luminance* L_o of the image.

Returns Corresponding electrical signal V.

Return type numpy.ndarray

Notes

- The function given in [InternationalTUnion21] multiplies L_o by 48 contrary to what is given in [SocietyoMPaTEngineers19] and `colour.models.eotf_inverse_DCDM()`.

Domain	Scale - Reference	Scale - 1
`L_o`	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
`V`	[0, 1]	[0, 1]

References

- [InternationalTUnion21], [SocietyoMPaTEngineers19]

Examples

```
>>> eotf_inverse_H273_ST428_1(0.18)
0.5000483...
```

colour.models.eotf_SMPTE240M

`colour.models.eotf_SMPTE240M`(*V_r: ArrayLike*) → NDArrayFloat

Define *SMPTE 240M* electro-optical transfer function (EOTF).

Parameters **V_r** (`ArrayLike`) – Video signal level V_r driving the reference reproducer normalised to the system reference white.

Returns Light output L_r from the reference reproducer normalised to the system reference white.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
`V_c`	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
`L_c`	[0, 1]	[0, 1]

References

[SocietyoMPaTEngineers99]

Examples

```
>>> eotf_SMPTE240M(0.402285796753870)
0.1...
```

colour.models.eotf_ST2084

colour.models.**eotf_ST2084**(*N: ArrayLike*, *L_p: float = 10000*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_ST2084*) → NDArrayFloat

Define *SMPTE ST 2084:2014* optimised perceptual electro-optical transfer function (EOTF).

This perceptual quantizer (PQ) has been modeled by Dolby Laboratories using *Barten (1999)* contrast sensitivity function.

Parameters

- **N** (ArrayLike) – Color value abbreviated as N, that is directly proportional to the encoded signal representation, and which is not directly proportional to the optical output of a display device.
- **L_p** (float) – System peak luminance cd/m^2, this parameter should stay at its default $10000cd/m^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.
- **constants** (colour.utilities.data_structures.Structure) – *SMPTE ST 2084:2014* constants.

Returns Target optical output C in cd/m^2 of the ideal reference display.

Return type numpy.ndarray

Warning: *SMPTE ST 2084:2014* is an absolute transfer function.

Notes

- *SMPTE ST 2084:2014* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
N	UN	UN

Range	Scale - Reference	Scale - 1
C	UN	UN

References

[Mil14], [SocietyoMPaTEngineers14]

Examples

```
>>> eotf_ST2084(0.508078421517399)
100.0000000...
```

colour.models.eotf_inverse_ST2084

colour.models.**eotf_inverse_ST2084**(*C: ArrayLike*, *L_p: float = 10000*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_ST2084*) → NDArrayFloat

Define *SMPTE ST 2084:2014* optimised perceptual inverse electro-optical transfer function (EOTF).

Parameters

- **C** (ArrayLike) – Target optical output C in cd/m^2 of the ideal reference display.
- **L_p** (float) – System peak luminance cd/m^2, this parameter should stay at its default $10000 cd/m^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.
- **constants** (colour.utilities.data_structures.Structure) – *SMPTE ST 2084:2014* constants.

Returns Color value abbreviated as N, that is directly proportional to the encoded signal representation, and which is not directly proportional to the optical output of a display device.

Return type numpy.ndarray

Warning: *SMPTE ST 2084:2014* is an absolute transfer function.

Notes

- *SMPTE ST 2084:2014* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations. The effective domain of *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) is [0.0001, 10000].

Domain	Scale - Reference	Scale - 1
C	UN	UN

Range	Scale - Reference	Scale - 1
N	UN	UN

References

[Mil14], [SocietyoMPaTEngineers14]

Examples

```
>>> eotf_inverse_ST2084(100)
0.5080784...
```

colour.models.eotf_sRGB

colour.models.**eotf_sRGB**(*V: ArrayLike*) → NDArrayFloat

Define the *IEC 61966-2-1:1999 sRGB* electro-optical transfer function (EOTF).

Parameters **V** (ArrayLike) – Electrical signal V.

Returns Corresponding *luminance* L of the image.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[InternationalECommission99], [InternationalTUnion15b]

Examples

```
>>> eotf_sRGB(0.461356129500442)
0.1...
```

colour.models.eotf_inverse_sRGB

colour.models.**eotf_inverse_sRGB**(*L: ArrayLike*) → NDArrayFloat

Define the *IEC 61966-2-1:1999 sRGB* inverse electro-optical transfer function (EOTF).

Parameters **L** (ArrayLike) – *Luminance* L of the image.

Returns Corresponding electrical signal V.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[InternationalECommission99], [InternationalTUnion15b]

Examples

```
>>> eotf_inverse_sRGB(0.18)
0.4613561...
```

Opto-Optical Transfer Functions

colour

ootf(value[, function])	Map relative scene linear light to display linear light using given opto-optical transfer function (OOTF / OOCF).
OOTFS	Supported opto-optical transfer functions (OOTFs / OOCFs).
ootf_inverse(value[, function])	Map relative display linear light to scene linear light using given inverse opto-optical transfer function (OOTF / OOCF).
OOTF_INVERSES	Supported inverse opto-optical transfer functions (OOTFs / OOCFs).

colour.ootf

colour.**ootf**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'], str] = 'ITU-R BT.2100 PQ', **kwargs: Any*) → numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]

Map relative scene linear light to display linear light using given opto-optical transfer function (OOTF / OOCF).

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Value.

- **function** (Union[Literal['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'], str]) – Opto-optical transfer function (OOTF / OOCF).
- **kwargs** (Any) – {colour.models.ootf_BT2100_HLG(), colour.models.ootf_BT2100_PQ()}, See the documentation of the previously listed definitions.

Returns Luminance of a displayed linear component.

Return type numpy.ndarray

Examples

```
>>> ootf(0.1)
779.9883608...
>>> ootf(0.1, function="ITU-R BT.2100 HLG")
63.0957344...
```

colour.OOTFS

colour.**OOTFS = CanonicalMapping({'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ...})**

Supported opto-optical transfer functions (OOTFs / OOCFs).

colour.ootf_inverse

colour.**ootf_inverse**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'], str] = 'ITU-R BT.2100 PQ', **kwargs: Any*) → numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]

Map relative display linear light to scene linear light using given inverse opto-optical transfer function (OOTF / OOCF).

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Value.
- **function** (Union[Literal['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'], str]) – Inverse opto-optical transfer function (OOTF / OOCF).
- **kwargs** (Any) – {colour.models.ootf_inverse_BT2100_HLG(), colour.models.ootf_inverse_BT2100_PQ()}, See the documentation of the previously listed definitions.

Returns Luminance of scene linear light.

Return type numpy.ndarray

Examples

```
>>> ootf_inverse(779.988360834115840)
0.1000000...
>>> ootf_inverse(
...     63.095734448019336, function="ITU-R BT.2100 HLG"
... )
0.1000000...
```

colour.OOTF_INVERSES

colour.**OOTF_INVERSES = CanonicalMapping({'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ...})**

Supported inverse opto-optical transfer functions (OOTFs / OOCFs).

colour.models

BT2100_HLG_OOTF_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).
ootf_BT2100_HLG(E[, L_B, L_W, gamma, method])	Define *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).
BT2100_HLG_OOTF_INVERSE_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).
ootf_inverse_BT2100_HLG(F_D[, L_B, L_W, ...])	Define *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).
ootf_BT2100_PQ(E)	Define *Recommendation ITU-R BT.2100 Reference PQ* opto-optical transfer function (OOTF / OOCF).
ootf_inverse_BT2100_PQ(F_D)	Define *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-optical transfer function (OOTF / OOCF).

colour.models.BT2100_HLG_OOTF_METHODS

colour.models.**BT2100_HLG_OOTF_METHODS = CanonicalMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2': ...})**

Supported *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

colour.models.ootf_BT2100_HLG

colour.models.**ootf_BT2100_HLG**(*E: ArrayLike, L_B: float = 0, L_W: float = 1000, gamma: float | None = None, method: Union[Literal['ITU-R BT.2100-1', 'ITU-R BT.2100-2'], str] = 'ITU-R BT.2100-2'*) → NDArrayFloat

Define *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** (ArrayLike) – E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.
- **L_B** (float) – L_B is the display luminance for black in cd/m^2.
- **L_W** (float) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (float | None) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **method** (Union[Literal['ITU-R BT.2100-1', 'ITU-R BT.2100-2'], str]) – Computation method.

Returns F_D is the luminance of a displayed linear component $R_D, G_D, or B_D$, in cd/m^2.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> ootf_BT2100_HLG(0.1)
63.0957344...
>>> ootf_BT2100_HLG(0.1, 0.01, method="ITU-R BT.2100-1")
...
63.1051034...
```

colour.models.BT2100_HLG_OOTF_INVERSE_METHODS

colour.models.**BT2100_HLG_OOTF_INVERSE_METHODS = CanonicalMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2': ...})**

Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

colour.models.ootf_inverse_BT2100_HLG

colour.models.**ootf_inverse_BT2100_HLG**(*F_D: ArrayLike, L_B: float = 0, L_W: float = 1000, gamma: float | None = None, method: Union[Literal['ITU-R BT.2100-1', 'ITU-R BT.2100-2'], str] = 'ITU-R BT.2100-2'*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).

Parameters

- **F_D** (ArrayLike) – F_D is the luminance of a displayed linear component $R_D, G_D, or B_D$, in cd/m^2.
- **L_B** (float) – L_B is the display luminance for black in cd/m^2.
- **L_W** (float) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (float | None) – System gamma value, 1.2 at the nominal display peak luminance of $1000 cd/m^2$.
- **method** (Union[Literal['ITU-R BT.2100-1', 'ITU-R BT.2100-2'], str]) – Computation method.

Returns E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

Examples

```
>>> ootf_inverse_BT2100_HLG(63.095734448019336)
0.1000000...
>>> ootf_inverse_BT2100_HLG(
...     63.105103490674857, 0.01, method="ITU-R BT.2100-1"
... )
...
0.0999999...
```

colour.models.ootf_BT2100_PQ

`colour.models.`**`ootf_BT2100_PQ`**(*E: ArrayLike*) → NDArrayFloat

Define *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** (ArrayLike) – $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure.

Returns F_D is the luminance of a displayed linear component (R_D, G_D, B_D; Y_D; or I_D).

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	UN	UN

Range	Scale - Reference	Scale - 1
F_D	UN	UN

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> ootf_BT2100_PQ(0.1)
779.9883608...
```

colour.models.ootf_inverse_BT2100_PQ

colour.models.**ootf_inverse_BT2100_PQ**(*F_D: ArrayLike*) → NDArrayFloat

Define *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-optical transfer function (OOTF / OOCF).

Parameters F_D (ArrayLike) – F_D is the luminance of a displayed linear component (R_D, G_D, B_D; Y_D; or I_D).

Returns $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	UN	UN

Range	Scale - Reference	Scale - 1
E	UN	UN

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> ootf_inverse_BT2100_PQ(779.988360834115840)
0.1000000...
```

Log Encoding and Decoding

colour

log_encoding(value[, function])	Encode *scene-referred* exposure values to $R'G'B'$ video component signal value using given *log* encoding function.
LOG_ENCODINGS	Supported *log* encoding functions.
log_decoding(value[, function])	Decode $R'G'B'$ video component signal value to *scene-referred* exposure values using given *log* decoding function.
LOG_DECODINGS	Supported *log* decoding functions.

colour.log_encoding

colour.**log_encoding**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[nump bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'L-Log', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'], str] = 'Cineon', **kwargs: Any*) → Union[numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]], numpy.ndarray[*Any*, numpy.dtype[Union[numpy.int8, numpy.int16, numpy.int32, numpy.int64, numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64]]]]

Encode *scene-referred* exposure values to $R'G'B'$ video component signal value using given *log* encoding function.

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – *Scene-referred* exposure values.
- **function** (Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'L-Log', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'], str]) – *Log* encoding function.
- **kwargs** (Any) – {colour.models.log_encoding_ACEScc(), colour.models.log_encoding_ACEScct(), colour.models.log_encoding_ACESproxy(), colour.models.log_encoding_ARRILogC3(), colour.models.log_encoding_ARRILogC4(), colour.models.log_encoding_CanonLog2(), colour.models.log_encoding_CanonLog3(), colour.models.log_encoding_CanonLog(), colour.models.log_encoding_Cineon(), colour.models.log_encoding_DJIDLog(), colour.models.log_encoding_ERIMMRGB(), colour.models.log_encoding_FLog(), colour.models.log_encoding_FLog2(), colour.models.log_encoding_FilmicPro6(), colour.models.log_encoding_LLog(), colour.models.log_encoding_Log2(), colour.models.log_encoding_Log3G10(), colour.models.log_encoding_Log3G12(), colour.models.log_encoding_NLog(), colour.models.log_encoding_PivotedLog(), colour.models.log_encoding_Panalog(), colour.models.log_encoding_Protune(), colour.models.log_encoding_REDLog(), colour.models.log_encoding_REDLogFilm(), colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2(), colour.models.log_encoding_SLog3(), colour.models.log_encoding_FilmLightTLog(), colour.models.log_encoding_VLog(), colour.models.log_encoding_ViperLog()}, See the documentation of the previously listed definitions.

Returns *Log* values.

Return type numpy.ndarray

Examples

```
>>> log_encoding(0.18)
0.4573196...
>>> log_encoding(0.18, function="ACEScc")
0.4135884...
>>> log_encoding(0.18, function="PLog", log_reference=400)
...
0.3910068...
>>> log_encoding(0.18, function="S-Log")
0.3849708...
```

colour.LOG_ENCODINGS

colour.**LOG_ENCODINGS = CanonicalMapping({'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ARRI LogC3': ..., 'ARRI LogC4': ..., 'Canon Log 2': ..., 'Canon Log 3': ..., 'Canon Log': ..., 'Cineon': ..., 'D-Log': ..., 'ERIMM RGB': ..., 'F-Log': ..., 'F-Log2': ..., 'Filmic Pro 6': ..., 'L-Log': ..., 'Log2': ..., 'Log3G10': ..., 'Log3G12': ..., 'N-Log': ..., 'PLog': ..., 'Panalog': ..., 'Protune': ..., 'REDLog': ..., 'REDLogFilm': ..., 'S-Log': ..., 'S-Log2': ..., 'S-Log3': ..., 'T-Log': ..., 'V-Log': ..., 'ViperLog': ...})**

Supported *log* encoding functions.

colour.log_decoding

colour.**log_decoding**(*value: Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]], function: Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'L-Log', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'], str] = 'Cineon', **kwargs: Any*) → numpy.ndarray[*Any*, numpy.dtype[Union[numpy.float16, numpy.float32, numpy.float64]]]

Decode $R'G'B'$ video component signal value to *scene-referred* exposure values using given *log* decoding function.

Parameters

- **value** (Union[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]], numpy._typing._nested_sequence._NestedSequence[numpy._typing._array_like._SupportsArray[numpy.dtype[Any]]], bool, int, float, complex, str, bytes, numpy._typing._nested_sequence._NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – *Log* values.
- **function** (Union[Literal['ACEScc', 'ACEScct', 'ACESproxy', 'ARRI LogC3', 'ARRI LogC4', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'F-Log2', 'Filmic Pro 6', 'L-Log', 'Log2', 'Log3G10', 'Log3G12', 'N-Log', 'PLog', 'Panalog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'], str]) – *Log* decoding function.
- **kwargs** (Any) – {colour.models.log_decoding_ACEScc(), colour.models.log_decoding_ACEScct(), colour.models.log_decoding_ACESproxy(), colour.models.log_decoding_ARRILogC3(), colour.models.log_decoding_ARRILogC4(), colour.models.log_decoding_CanonLog2(),

colour.models.log_decoding_CanonLog3(), colour.models.log_decoding_CanonLog(), colour.models.log_decoding_Cineon(), colour.models.log_decoding_DJIDLog(), colour.models.log_decoding_ERIMMRGB(), colour.models.log_decoding_FLog(), colour.models.log_decoding_FLog2(), colour.models.log_decoding_FilmicPro6(), colour.models.log_decoding_LLog(), colour.models.log_decoding_Log2(), colour.models.log_decoding_Log3G10(), colour.models.log_decoding_Log3G12(), colour.models.log_decoding_NLog(), colour.models.log_decoding_PivotedLog(), colour.models.log_decoding_Panalog(), colour.models.log_decoding_Protune(), colour.models.log_decoding_REDLog(), colour.models.log_decoding_REDLogFilm(), colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2(), colour.models.log_decoding_SLog3(), colour.models.log_decoding_FilmLightTLog(), colour.models.log_decoding_VLog(), colour.models.log_decoding_ViperLog()}, See the documentation of the previously listed definitions.

Returns *Scene-referred* exposure values.

Return type numpy.ndarray

Examples

```
>>> log_decoding(0.457319613085418)
0.1...
>>> log_decoding(0.413588402492442, function="ACEScc")
...
0.1...
>>> log_decoding(0.391006842619746, function="PLog", log_reference=400)
...
0.1...
>>> log_decoding(0.376512722254600, function="S-Log")
...
0.1...
```

colour.LOG_DECODINGS

colour.**LOG_DECODINGS = CanonicalMapping({'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ARRI LogC3': ..., 'ARRI LogC4': ..., 'Canon Log 2': ..., 'Canon Log 3': ..., 'Canon Log': ..., 'Cineon': ..., 'D-Log': ..., 'ERIMM RGB': ..., 'F-Log': ..., 'F-Log2': ..., 'Filmic Pro 6': ..., 'L-Log': ..., 'Log2': ..., 'Log3G10': ..., 'Log3G12': ..., 'N-Log': ..., 'PLog': ..., 'Panalog': ..., 'Protune': ..., 'REDLog': ..., 'REDLogFilm': ..., 'S-Log': ..., 'S-Log2': ..., 'S-Log3': ..., 'T-Log': ..., 'V-Log': ..., 'ViperLog': ...})**

Supported *log* decoding functions.

colour.models

log_encoding_ACEScc(lin_AP1)	Define the *ACEScc* colourspace log encoding / opto-electronic transfer function.
log_decoding_ACEScc(ACEScc)	Define the *ACEScc* colourspace log decoding / electro-optical transfer function.
log_encoding_ACEScct(lin_AP1[, constants])	Define the *ACEScct* colourspace log encoding / opto-electronic transfer function.
log_decoding_ACEScct(ACEScct[, constants])	Define the *ACEScct* colourspace log decoding / electro-optical transfer function.
log_encoding_ACESproxy(lin_AP1[, bit_depth, ...])	Define the *ACESproxy* colourspace log encoding curve / opto-electronic transfer function.

continues on next page

Table 3 – continued from previous page

log_decoding_ACESproxy(ACESproxy[, ...])	Define the *ACESproxy* colourspace log decoding curve / electro-optical transfer function.
log_encoding_ARRILogC3(x[, firmware, method, EI])	Define the *ARRI LogC3* log encoding curve / opto-electronic transfer function.
log_decoding_ARRILogC3(t[, firmware, method, EI])	Define the *ARRI LogC3* log decoding curve / electro-optical transfer function.
log_encoding_CanonLog2(x[, bit_depth, ...])	Define the *Canon Log 2* log encoding curve / opto-electronic transfer function.
log_decoding_CanonLog2(clog2[, bit_depth, ...])	Define the *Canon Log 2* log decoding curve / electro-optical transfer function.
log_encoding_CanonLog3(x[, bit_depth, ...])	Define the *Canon Log 3* log encoding curve / opto-electronic transfer function.
log_decoding_CanonLog3(clog3[, bit_depth, ...])	Define the *Canon Log 3* log decoding curve / electro-optical transfer function.
log_encoding_CanonLog(x[, bit_depth, ...])	Define the *Canon Log* log encoding curve / opto-electronic transfer function.
log_decoding_CanonLog(clog[, bit_depth, ...])	Define the *Canon Log* log decoding curve / electro-optical transfer function.
log_encoding_Cineon(x[, black_offset])	Define the *Cineon* log encoding curve / opto-electronic transfer function.
log_decoding_Cineon(y[, black_offset])	Define the *Cineon* log decoding curve / electro-optical transfer function.
log_encoding_ERIMMRGB(X[, bit_depth, ...])	Define the *ERIMM RGB* log encoding curve / opto-electronic transfer function (OETF).
log_decoding_ERIMMRGB(X_p[, bit_depth, ...])	Define the *ERIMM RGB* log decoding curve / electro-optical transfer function (EOTF).
log_encoding_FLog(in_r[, bit_depth, ...])	Define the *Fujifilm F-Log* log encoding curve / opto-electronic transfer function.
log_decoding_FLog(out_r[, bit_depth, ...])	Define the *Fujifilm F-Log* log decoding curve / electro-optical transfer function.
log_encoding_FLog2(in_r[, bit_depth, ...])	Define the *Fujifilm F-Log2* log encoding curve / opto-electronic transfer function.
log_decoding_FLog2(out_r[, bit_depth, ...])	Define the *Fujifilm F-Log2* log decoding curve / electro-optical transfer function.
log_encoding_LLog(LSR[, bit_depth, ...])	Define the *Leica L-Log* log encoding curve / opto-electronic transfer function.
log_decoding_LLog(LLog[, bit_depth, ...])	Define the *Leica L-Log* log decoding curve / electro-optical transfer function.
log_encoding_Log2(lin[, middle_grey, ...])	Define the common *Log2* encoding function.
log_decoding_Log2(log_norm[, middle_grey, ...])	Define the common *Log2* decoding function.
LOG3G10_ENCODING_METHODS	Supported *Log3G10* log encoding curve / opto-electronic transfer function methods.
log_encoding_Log3G10(x[, method])	Define the *Log3G10* log encoding curve / opto-electronic transfer function.
LOG3G10_DECODING_METHODS	Supported *Log3G10* log decoding curve / electro-optical transfer function methods.
log_decoding_Log3G10(y[, method])	Define the *Log3G10* log decoding curve / electro-optical transfer function.
log_encoding_Log3G12(x)	Define the *Log3G12* log encoding curve / opto-electronic transfer function.
log_decoding_Log3G12(y)	Define the *Log3G12* log decoding curve / electro-optical transfer function.
log_encoding_NLog(y[, bit_depth, ...])	Define the *Nikon N-Log* log encoding curve / opto-electronic transfer function.

continues on next page

Table 3 – continued from previous page

log_decoding_NLog(x[, bit_depth, ...])	Define the *Nikon N-Log* log decoding curve / electro-optical transfer function.
log_encoding_Panalog(x[, black_offset])	Define the *Panalog* log encoding curve / opto-electronic transfer function.
log_decoding_Panalog(y[, black_offset])	Define the *Panalog* log decoding curve / electro-optical transfer function.
log_encoding_PivotedLog(x[, log_reference, ...])	Define the *Josh Pines* style *Pivoted Log* log encoding curve / opto-electronic transfer function.
log_decoding_PivotedLog(y[, log_reference, ...])	Define the *Josh Pines* style *Pivoted Log* log decoding curve / electro-optical transfer function.
log_encoding_Protune(x)	Define the *Protune* log encoding curve / opto-electronic transfer function.
log_decoding_Protune(y)	Define the *Protune* log decoding curve / electro-optical transfer function.
log_encoding_REDLog(x[, black_offset])	Define the *REDLog* log encoding curve / opto-electronic transfer function.
log_decoding_REDLog(y[, black_offset])	Define the *REDLog* log decoding curve / electro-optical transfer function.
log_encoding_REDLogFilm(x[, black_offset])	Define the *REDLogFilm* log encoding curve / opto-electronic transfer function.
log_decoding_REDLogFilm(y[, black_offset])	Define the *REDLogFilm* log decoding curve / electro-optical transfer function.
log_encoding_SLog(x[, bit_depth, ...])	Define the *Sony S-Log* log encoding curve / opto-electronic transfer function.
log_decoding_SLog(y[, bit_depth, ...])	Define the *Sony S-Log* log decoding curve / electro-optical transfer function.
log_encoding_SLog2(x[, bit_depth, ...])	Define the *Sony S-Log2* log encoding curve / opto-electronic transfer function.
log_decoding_SLog2(y[, bit_depth, ...])	Define the *Sony S-Log2* log decoding curve / electro-optical transfer function.
log_encoding_SLog3(x[, bit_depth, ...])	Define the *Sony S-Log3* log encoding curve / opto-electronic transfer function.
log_decoding_SLog3(y[, bit_depth, ...])	Define the *Sony S-Log3* log decoding curve / electro-optical transfer function.
log_encoding_VLog(L_in[, bit_depth, ...])	Define the *Panasonic V-Log* log encoding curve / opto-electronic transfer function.
log_decoding_VLog(V_out[, bit_depth, ...])	Define the *Panasonic V-Log* log decoding curve / electro-optical transfer function.
log_encoding_ViperLog(x)	Define the *Viper Log* log encoding curve / opto-electronic transfer function.
log_decoding_ViperLog(y)	Define the *Viper Log* log decoding curve / electro-optical transfer function.

colour.models.log_encoding_ACEScc

colour.models.**log_encoding_ACEScc**(*lin_AP1: ArrayLike*) → NDArrayFloat

Define the *ACEScc* colourspace log encoding / opto-electronic transfer function.

Parameters **lin_AP1** (ArrayLike) – *lin_AP1* value.

Returns *ACEScc* non-linear value.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
ACEScc	[0, 1]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

Examples

```
>>> log_encoding_ACEScc(0.18)
0.4135884...
```

colour.models.log_decoding_ACEScc

colour.models.**log_decoding_ACEScc**(*ACEScc: ArrayLike*) → NDArrayFloat

Define the *ACEScc* colourspace log decoding / electro-optical transfer function.

Parameters **ACEScc** (ArrayLike) – *ACEScc* non-linear value.

Returns *lin_AP1* value.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
ACEScc	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

Examples

```
>>> log_decoding_ACEScc(0.413588402492442)
0.1799999...
```

colour.models.log_encoding_ACEScct

colour.models.**log_encoding_ACEScct**(*lin_AP1: ArrayLike*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_ACES_CCT*) → NDArrayFloat

Define the *ACEScct* colourspace log encoding / opto-electronic transfer function.

Parameters

- **lin_AP1** (ArrayLike) – *lin_AP1* value.
- **constants** (colour.utilities.data_structures.Structure) – *ACEScct* constants.

Returns *ACEScct* non-linear value.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
ACEScct	[0, 1]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

Examples

```
>>> log_encoding_ACEScct(0.18)
0.4135884...
```

colour.models.log_decoding_ACEScct

colour.models.**log_decoding_ACEScct**(*ACEScct: ArrayLike*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_ACES_CCT*) → NDArrayFloat

Define the *ACEScct* colourspace log decoding / electro-optical transfer function.

Parameters

- **ACEScct** (ArrayLike) – *ACEScct* non-linear value.
- **constants** (colour.utilities.data_structures.Structure) – *ACEScct* constants.

Returns *lin_AP1* value.

Return type numpy.ndarray

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

Notes

Domain	Scale - Reference	Scale - 1
ACEScct	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Examples

```
>>> log_decoding_ACEScct(0.413588402492442)
0.1799999...
```

colour.models.log_encoding_ACESproxy

colour.models.**log_encoding_ACESproxy**(*lin_AP1: ArrayLike*, *bit_depth: Literal[10, 12] = 10*, *out_int: bool = False*, *constants: dict = CONSTANTS_ACES_PROXY*) → NDArrayFloat | NDArrayInt

Define the *ACESproxy* colourspace log encoding curve / opto-electronic transfer function.

Parameters

- **lin_AP1** (ArrayLike) – *lin_AP1* value.
- **bit_depth** (Literal[10, 12]) – *ACESproxy* bit-depth.
- **out_in** – Whether to return value as int code value or float equivalent of a code value at a given bit-depth.
- **constants** (dict) – *ACESproxy* constants.
- **out_int** (bool) –

Returns *ACESproxy* non-linear value.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
ACESproxy	[0, 1]	[0, 1]

* This definition has an output int switch, thus the domain-range scale information is only given for the floating point mode.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee13], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

Examples

```
>>> log_encoding_ACESproxy(0.18)
0.4164222...
>>> log_encoding_ACESproxy(0.18, out_int=True)
426
```

colour.models.log_decoding_ACESproxy

colour.models.**log_decoding_ACESproxy**(*ACESproxy: ArrayLike, bit_depth: Literal[10, 12] = 10, in_int: bool = False, constants: dict | None = None*) → NDArrayFloat

Define the *ACESproxy* colourspace log decoding curve / electro-optical transfer function.

Parameters

- **ACESproxy** (ArrayLike) – *ACESproxy* non-linear value.
- **bit_depth** (Literal[10, 12]) – *ACESproxy* bit-depth.
- **in_int** (bool) – Whether to treat the input value as int code value or float equivalent of a code value at a given bit-depth.
- **constants** (dict | None) – *ACESproxy* constants.

Returns *lin_AP1* value.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
ACESproxy	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

* This definition has an input int switch, thus the domain-range scale information is only given for the floating point mode.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee13], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea]

Examples

```
>>> log_decoding_ACESproxy(0.416422287390029)
0.1...
>>> log_decoding_ACESproxy(426, in_int=True)
0.1...
```

colour.models.log_encoding_ARRILogC3

`colour.models.log_encoding_ARRILogC3`(*x: ArrayLike, firmware: Union[Literal['SUP 2.x', 'SUP 3.x'], str] = 'SUP 3.x', method: Union[Literal['Linear Scene Exposure Factor', 'Normalised Sensor Signal'], str] = 'Linear Scene Exposure Factor', EI: Literal[160, 200, 250, 320, 400, 500, 640, 800, 1000, 1280, 1600] = 800*) → NDArrayFloat

Define the *ARRI LogC3* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **firmware** (`Union[Literal['SUP 2.x', 'SUP 3.x'], str]`) – Alexa firmware version.
- **method** (`Union[Literal['Linear Scene Exposure Factor', 'Normalised Sensor Signal'], str]`) – Conversion method.
- **EI** (`Literal[160, 200, 250, 320, 400, 500, 640, 800, 1000, 1280, 1600]`) – Exposure Index EI.

Returns *ARRI LogC3* encoded data t.

Return type `numpy.ndarray`

References

[ARRI12]

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
t	[0, 1]	[0, 1]

Examples

```
>>> log_encoding_ARRILogC3(0.18)
0.3910068...
```

colour.models.log_decoding_ARRILogC3

colour.models.**log_decoding_ARRILogC3**(*t: ArrayLike, firmware: Union[Literal['SUP 2.x', 'SUP 3.x'], str] = 'SUP 3.x', method: Union[Literal['Linear Scene Exposure Factor', 'Normalised Sensor Signal'], str] = 'Linear Scene Exposure Factor', EI: Literal[160, 200, 250, 320, 400, 500, 640, 800, 1000, 1280, 1600] = 800*) → NDArrayFloat

Define the *ARRI LogC3* log decoding curve / electro-optical transfer function.

Parameters

- **t** (ArrayLike) – *ARRI LogC3* encoded data t.
- **firmware** (Union[Literal['SUP 2.x', 'SUP 3.x'], str]) – Alexa firmware version.
- **method** (Union[Literal['Linear Scene Exposure Factor', 'Normalised Sensor Signal'], str]) – Conversion method.
- **EI** (Literal[160, 200, 250, 320, 400, 500, 640, 800, 1000, 1280, 1600]) – Exposure Index EI.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
t	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[ARRI12]

Examples

```
>>> log_decoding_ARRILogC3(0.391006832034084)
0.18...
```

colour.models.log_encoding_CanonLog2

colour.models.**log_encoding_CanonLog2**(*x: ArrayLike, bit_depth: int = 10, out_normalised_code_value: bool = True, in_reflection: bool = True, method: Union[Literal['v1', 'v1.2'], str] = 'v1.2'*) → NDArrayFloat

Define the *Canon Log 2* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the *Canon Log 2* non-linear data is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level x to a camera is reflection.
- **method** (Union[Literal['v1', 'v1.2'], str]) – Computation method.

Returns *Canon Log 2* non-linear data.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog2	[0, 1]	[0, 1]

References

[Canon16], [Canon20]

Examples

```
>>> log_encoding_CanonLog2(0.18) * 100
39.8254692...
```

colour.models.log_decoding_CanonLog2

colour.models.**log_decoding_CanonLog2**(*clog2: ArrayLike, bit_depth: int = 10, in_normalised_code_value: bool = True, out_reflection: bool = True, method: Union[Literal['v1', 'v1.2'], str] = 'v1.2'*) → NDArrayFloat

Define the *Canon Log 2* log decoding curve / electro-optical transfer function.

Parameters

- **clog2** (ArrayLike) – *Canon Log 2* non-linear data.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the *Canon Log 2* non-linear data is encoded with normalised code values.
- **out_reflection** (bool) – Whether the light level x to a camera is reflection.
- **method** (Union[Literal['v1', 'v1.2'], str]) – Computation method.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
clog2	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Canon16], [Canon20]

Examples

```
>>> log_decoding_CanonLog2(39.825469256149191 / 100)
0.1799999...
```

colour.models.log_encoding_CanonLog3

colour.models.**log_encoding_CanonLog3**(*x: ArrayLike*, *bit_depth: int = 10*, *out_normalised_code_value: bool = True*, *in_reflection: bool = True*, *method: Union[Literal['v1', 'v1.2'], str] = 'v1.2'*) → NDArrayFloat

Define the *Canon Log 3* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the *Canon Log 3* non-linear data is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level x to a camera is reflection.
- **method** (Union[Literal['v1', 'v1.2'], str]) – Computation method.

Returns *Canon Log 3* non-linear data.

Return type numpy.ndarray

Notes

- Introspection of the grafting points by Shaw, N. (2018) shows that the *Canon Log 3* v1 IDT was likely derived from its encoding curve as the latter is grafted at *+/-0.014*:

```
>>> clog3 = 0.04076162
>>> (clog3 - 0.073059361) / 2.3069815
-0.014000000000000002
>>> clog3 = 0.105357102
>>> (clog3 - 0.073059361) / 2.3069815
0.013999999999999997
```

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog2	[0, 1]	[0, 1]

References

[Canon16], [Canon20]

Examples

```
>>> log_encoding_CanonLog3(0.18) * 100
34.3389370...
```

colour.models.log_decoding_CanonLog3

colour.models.**log_decoding_CanonLog3**(*clog3: ArrayLike, bit_depth: int = 10, in_normalised_code_value: bool = True, out_reflection: bool = True, method: Union[Literal['v1', 'v1.2'], str] = 'v1.2'*) → NDArrayFloat

Define the *Canon Log 3* log decoding curve / electro-optical transfer function.

Parameters

- **clog3** (ArrayLike) – *Canon Log 3* non-linear data.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the *Canon Log 3* non-linear data is encoded with normalised code values.
- **out_reflection** (bool) – Whether the light level x to a camera is reflection.
- **method** (Union[Literal['v1', 'v1.2'], str]) – Computation method.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
clog2	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Canon16], [Canon20]

Examples

```
>>> log_decoding_CanonLog3(34.338937037393549 / 100)
0.1799999...
```

colour.models.log_encoding_CanonLog

colour.models.**log_encoding_CanonLog**(*x: ArrayLike, bit_depth: int = 10, out_normalised_code_value: bool = True, in_reflection: bool = True, method: Union[Literal['v1', 'v1.2'], str] = 'v1.2'*) → NDArrayFloat

Define the *Canon Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the *Canon Log* non-linear data is encoded as normalised code values.

- **in_reflection** (bool) – Whether the light level x to a camera is reflection.
- **method** (Union[Literal['v1', 'v1.2'], str]) – Computation method.

Returns *Canon Log* non-linear data.

Return type numpy.ndarray

References

[Canon16], [Canon20], [Tho12]

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog	[0, 1]	[0, 1]

Examples

```
>>> log_encoding_CanonLog(0.18) * 100
34.3389649...
>>> log_encoding_CanonLog(0.18, method="v1") * 100
34.3389651...
```

The values of *Table 2 Canon-Log Code Values* table in [Tho12] are obtained as follows:

```
>>> x = np.array([0, 2, 18, 90, 720]) / 100
>>> np.around(
...     log_encoding_CanonLog(x, method="v1") * (2**10 - 1)
... ).astype(np.int_)
array([ 128,  169,  351,  614, 1016])
>>> np.around(log_encoding_CanonLog(x, 10, False, method="v1") * 100, 1)
array([   7.3,   12. ,   32.8,   62.7,  108.7])
```

colour.models.log_decoding_CanonLog

colour.models.**log_decoding_CanonLog**(*clog: ArrayLike, bit_depth: int = 10, in_normalised_code_value: bool = True, out_reflection: bool = True, method: Union[Literal['v1', 'v1.2'], str] = 'v1.2'*) → NDArrayFloat

Define the *Canon Log* log decoding curve / electro-optical transfer function.

Parameters

- **clog** (ArrayLike) – *Canon Log* non-linear data.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the *Canon Log* non-linear data is encoded with normalised code values.
- **out_reflection** (bool) – Whether the light level x to a camera is reflection.
- **method** (Union[Literal['v1', 'v1.2'], str]) – Computation method.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
clog	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Canon16], [Canon20], [Tho12]

Examples

```
>>> log_decoding_CanonLog(34.338964929528061 / 100)
0.17999999...
>>> log_decoding_CanonLog(34.338965172606912 / 100, method="v1")
...
0.17999999...
```

colour.models.log_encoding_Cineon

colour.models.**log_encoding_Cineon**(*x: ArrayLike, black_offset: ArrayLike = 10 ** 95 - 685 / 300*) → NDArrayFloat

Define the *Cineon* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **black_offset** (ArrayLike) – Black offset.

Returns Non-linear data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_Cineon(0.18)
0.4573196...
```

colour.models.log_decoding_Cineon

colour.models.**log_decoding_Cineon**(*y: ArrayLike*, *black_offset: ArrayLike = 10 ** 95 - 685 / 300*) → NDArrayFloat

Define the *Cineon* log decoding curve / electro-optical transfer function.

Parameters

- **y** (ArrayLike) – Non-linear data y.
- **black_offset** (ArrayLike) – Black offset.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_Cineon(0.457319613085418)
0.1799999...
```

colour.models.log_encoding_ERIMMRGB

colour.models.**log_encoding_ERIMMRGB**(*X: ArrayLike*, *bit_depth: int = 8*, *out_int: bool = False*, *E_min: float = 0.001*, *E_clip: float = 316.2*) → NDArrayReal

Define the *ERIMM RGB* log encoding curve / opto-electronic transfer function (OETF).

Parameters

- **X** (ArrayLike) – Linear data X_{ERIMM}.
- **bit_depth** (int) – Bit-depth used for conversion.

- **out_int** (bool) – Whether to return value as int code value or float equivalent of a code value at a given bit-depth.
- **E_min** (float) – Minimum exposure limit.
- **E_clip** (float) – Maximum exposure limit.

Returns Non-linear data X'_{ERIMM}.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

* This definition has an output int switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> log_encoding_ERIMMRGB(0.18)
0.4100523...
>>> log_encoding_ERIMMRGB(0.18, out_int=True)
105
```

colour.models.log_decoding_ERIMMRGB

colour.models.**log_decoding_ERIMMRGB**(*X_p: ArrayLike, bit_depth: int = 8, in_int: bool = False, E_min: float = 0.001, E_clip: float = 316.2*) → NDArrayFloat

Define the *ERIMM RGB* log decoding curve / electro-optical transfer function (EOTF).

Parameters

- **X_p** (ArrayLike) – Non-linear data X'_{ERIMM}.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_int** (bool) – Whether to treat the input value as int code value or float equivalent of a code value at a given bit-depth.
- **E_min** (float) – Minimum exposure limit.
- **E_clip** (float) – Maximum exposure limit.

Returns Linear data X_{ERIMM}.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

* This definition has an input int switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> log_decoding_ERIMMRGB(0.410052389492129)
0.1...
>>> log_decoding_ERIMMRGB(105, in_int=True)
0.1...
```

colour.models.log_encoding_FLog

colour.models.**log_encoding_FLog**(*in_r: ArrayLike, bit_depth: int = 10, out_normalised_code_value: bool = True, in_reflection: bool = True, constants: colour.utilities.data_structures.Structure = CONSTANTS_FLOG*) → NDArrayFloat

Define the *Fujifilm F-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **in_r** (ArrayLike) – Linear reflection data :math`in`.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the non-linear *Fujifilm F-Log* data *out* is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level :math`in` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Fujifilm F-Log* constants.

Returns Non-linear data *out*.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
in_r	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
out_r	[0, 1]	[0, 1]

References

[Fujifilm22a]

Examples

```
>>> log_encoding_FLog(0.18)
0.4593184...
```

The values of *2-2. F-Log Code Value* table in [Fujifilm22a] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_FLog(x, 10, False) * 100, 1)
array([  3.5,  46.3,  73.2])
>>> np.around(log_encoding_FLog(x) * (2**10 - 1)).astype(np.int_)
array([ 95, 470, 705])
```

colour.models.log_decoding_FLog

colour.models.**log_decoding_FLog**(*out_r: ArrayLike, bit_depth: int = 10, in_normalised_code_value: bool = True, out_reflection: bool = True, constants: colour.utilities.data_structures.Structure = CONSTANTS_FLOG*) → NDArrayFloat

Define the *Fujifilm F-Log* log decoding curve / electro-optical transfer function.

Parameters

- **out_r** (ArrayLike) – Non-linear data *out*.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the non-linear *Fujifilm F-Log* data *out* is encoded as normalised code values.
- **out_reflection** (bool) – Whether the light level :math`in` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Fujifilm F-Log* constants.

Returns Linear reflection data :math`in`.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
out_r	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
in_r	[0, 1]	[0, 1]

References

[Fujifilm22a]

Examples

```
>>> log_decoding_FLog(0.45931845866162124)
0.1800000...
```

colour.models.log_encoding_FLog2

colour.models.**log_encoding_FLog2**(*in_r: ArrayLike, bit_depth: int = 10, out_normalised_code_value: bool = True, in_reflection: bool = True, constants: colour.utilities.data_structures.Structure = CONSTANTS_FLOG2*) → NDArrayFloat

Define the *Fujifilm F-Log2* log encoding curve / opto-electronic transfer function.

Parameters

- **in_r** (ArrayLike) – Linear reflection data :math`in`.
- **bit_depth** (int) – Bit depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the non-linear *Fujifilm F-Log2* data *out* is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level :math`in` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Fujifilm F-Log2* constants.

Returns Non-linear data *out*.

Return type numpy.floating or numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
in_r	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
out_r	[0, 1]	[0, 1]

References

[Fujifilm22b]

Examples

```
>>> log_encoding_FLog2(0.18)
0.3910072...
```

The values of *2-2. F-Log2 Code Value* table in [Fujifilm22b] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_FLog2(x, 10, False) * 100, 1)
array([  3.5,   38.4,   57.8])
>>> np.around(log_encoding_FLog2(x) * (2**10 - 1)).astype(np.int_)
array([ 95, 400, 570])
```

colour.models.log_decoding_FLog2

colour.models.**log_decoding_FLog2**(*out_r: ArrayLike*, *bit_depth: int = 10*, *in_normalised_code_value: bool = True*, *out_reflection: bool = True*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_FLOG2*) → NDArrayFloat

Define the *Fujifilm F-Log2* log decoding curve / electro-optical transfer function.

Parameters

- **out_r** (ArrayLike) – Non-linear data *out*.
- **bit_depth** (int) – Bit depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the non-linear *Fujifilm F-Log2* data *out* is encoded as normalised code values.
- **out_reflection** (bool) – Whether the light level :math`in` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Fujifilm F-Log2* constants.

Returns Linear reflection data :math`in`.

Return type numpy.floating or numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
out_r	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
in_r	[0, 1]	[0, 1]

References

[Fujifilm22b]

Examples

```
>>> log_decoding_FLog2(0.39100724189123004)
0.1799999...
```

colour.models.log_encoding_LLog

colour.models.**log_encoding_LLog**(*LSR: ArrayLike*, *bit_depth: int = 10*, *out_normalised_code_value: bool = True*, *in_reflection: bool = True*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_LLOG*) → NDArrayFloat

Define the *Leica L-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **LSR** (ArrayLike) – Linear scene reflection LSR values.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the non-linear *Leica L-Log* data $L - Log$ is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level :math`in` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Leica L-Log* constants.

Returns *L-Log* 10-bit equivalent code value $L - Log$.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
LSR	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
LLog	[0, 1]	[0, 1]

References

[LeicaCAG22]

Examples

```
>>> log_encoding_LLog(0.18)
0.4353139...
```

colour.models.log_decoding_LLog

colour.models.**log_decoding_LLog**(*LLog: ArrayLike*, *bit_depth: int = 10*, *in_normalised_code_value: bool = True*, *out_reflection: bool = True*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_LLOG*) → NDArrayFloat

Define the *Leica L-Log* log decoding curve / electro-optical transfer function.

Parameters

- **LLog** (ArrayLike) – *L-Log* 10-bit equivalent code value $L - Log$.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the non-linear *Leica L-Log* data $L - Log$ is encoded as normalised code values.
- **out_reflection** (bool) – Whether the light level :math`in` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Leica L-Log* constants.

Returns Linear scene reflection LSR values.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
LLog	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
LSR	[0, 1]	[0, 1]

References

[LeicaCAG22]

Examples

```
>>> log_decoding_LLog(0.43531390404392656)
0.1800000...
```

colour.models.log_encoding_Log2

colour.models.**log_encoding_Log2**(*lin: ArrayLike*, *middle_grey: float = 0.18*, *min_exposure: float = - 6.5*, *max_exposure: float = 6.5*) → NDArrayFloat

Define the common *Log2* encoding function.

Parameters

- **lin** (ArrayLike) – Linear data to undergo encoding.
- **middle_grey** (float) – *Middle Grey* exposure value.
- **min_exposure** (float) – Minimum exposure level.
- **max_exposure** (float) – Maximum exposure level.

Returns Non-linear *Log2* encoded data.

Return type numpy.ndarray

Notes

- The common *Log2* encoding function can be used to build linear to logarithmic shapers in the *ACES OCIO configuration*.
- A (48-nits OCIO) shaper having values in a linear domain, can be encoded to a logarithmic domain:

Shaper Domain	Shaper Range
[0.002, 16.291]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteeb]

Examples

```
>>> log_encoding_Log2(0.18)
0.5
```

colour.models.log_decoding_Log2

colour.models.**log_decoding_Log2**(*log_norm: ArrayLike*, *middle_grey: float = 0.18*, *min_exposure: float = - 6.5*, *max_exposure: float = 6.5*) → NDArrayFloat

Define the common *Log2* decoding function.

Parameters

- **log_norm** (ArrayLike) – Logarithmic data to undergo decoding.
- **middle_grey** (float) – *Middle Grey* exposure value.
- **min_exposure** (float) – Minimum exposure level.
- **max_exposure** (float) – Maximum exposure level.

Returns Linear *Log2* decoded data.

Return type numpy.ndarray

Notes

- The common *Log2* decoding function can be used to build logarithmic to linear shapers in the *ACES OCIO configuration*.
- The shaper with logarithmic encoded values can be decoded back to linear domain:

Shaper Range	Shaper Domain
[0, 1]	[0.002, 16.291]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteec]

Examples

```
>>> log_decoding_Log2(0.5)
0.1799999...
```

colour.models.LOG3G10_ENCODING_METHODS

colour.models.**LOG3G10_ENCODING_METHODS = CanonicalMapping({'v1': ..., 'v2': ..., 'v3': ...})**

Supported *Log3G10* log encoding curve / opto-electronic transfer function methods.

References

[Nat16], [REDDCinema17]

colour.models.log_encoding_Log3G10

colour.models.**log_encoding_Log3G10**(*x: ArrayLike*, *method: Union[Literal['v1', 'v2', 'v3'], str] = 'v3'*) → NDArrayFloat

Define the *Log3G10* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **method** (Union[Literal['v1', 'v2', 'v3'], str]) – Computation method.

Returns Non-linear data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

- The *Log3G10 v1* log encoding curve is the one used in *REDCINE-X Beta 42*. *Resolve 12.5.2* also uses the *v1* curve. *RED* is planning to use the *Log3G10 v2* log encoding curve in the release version of the *RED SDK*.
- The intent of the *Log3G10 v1* log encoding curve is that zero maps to zero, 0.18 maps to 1/3, and 10 stops above 0.18 maps to 1.0. The name indicates this in a similar way to the naming conventions of *Sony HyperGamma* curves.

  The constants used in the functions do not in fact quite hit these values, but rather than use corrected constants, the functions here use the official *RED* values, in order to match the output of the *RED SDK*.

  For those interested, solving for constants which exactly hit 1/3 and 1.0 yields the following values:

  ```
  B = 25 * (np.sqrt(4093.0) - 3) / 9
  A = 1 / np.log10(B * 184.32 + 1)
  ```

  where the function takes the form:

  ```
  Log3G10(x) = A * np.log10(B * x + 1)
  ```

  Similarly for *Log3G12*, the values which hit exactly 1/3 and 1.0 are:

  ```
  B = 25 * (np.sqrt(16381.0) - 3) / 9
  A = 1 / np.log10(B * 737.28 + 1)
  ```

References

[Nat16], [REDDCinema17]

Examples

```
>>> log_encoding_Log3G10(0.0)
0.09155148...
>>> log_encoding_Log3G10(0.18, method="v1")
0.3333336...
```

colour.models.LOG3G10_DECODING_METHODS

`colour.models.LOG3G10_DECODING_METHODS = CanonicalMapping({'v1': ..., 'v2': ..., 'v3': ...})`

Supported *Log3G10* log decoding curve / electro-optical transfer function methods.

References

[Nat16], [REDDCinema17]

colour.models.log_decoding_Log3G10

`colour.models.log_decoding_Log3G10`(*y*, *method: Union[Literal['v1', 'v2', 'v3'], str] = 'v3'*) → NDArrayFloat

Define the *Log3G10* log decoding curve / electro-optical transfer function.

Parameters

- **y** – Non-linear data y.
- **method** (`Union[Literal['v1', 'v2', 'v3'], str]`) – Computation method.

Returns Linear data x.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Nat16], [REDDCinema17]

Examples

```
>>> log_decoding_Log3G10(1.0)
184.3223476...
>>> log_decoding_Log3G10(1.0 / 3, method="v1")
0.1799994...
```

colour.models.log_encoding_Log3G12

colour.models.**log_encoding_Log3G12**(*x: ArrayLike*) → NDArrayFloat

Define the *Log3G12* log encoding curve / opto-electronic transfer function.

Parameters **x** (ArrayLike) – Linear data x.

Returns Non-linear data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[Nat16], [REDDCinema17]

Examples

```
>>> log_encoding_Log3G12(0.18)
0.3333326...
```

colour.models.log_decoding_Log3G12

colour.models.**log_decoding_Log3G12**(*y: ArrayLike*) → NDArrayFloat

Define the *Log3G12* log decoding curve / electro-optical transfer function.

Parameters **y** (ArrayLike) – Non-linear data y.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Nat16], [REDDCinema17]

Examples

```
>>> log_decoding_Log3G12(1.0 / 3)
0.1800015...
```

colour.models.log_encoding_NLog

colour.models.**log_encoding_NLog**(*y: ArrayLike*, *bit_depth: int = 10*, *out_normalised_code_value: bool = True*, *in_reflection: bool = True*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_NLOG*) → NDArrayFloat

Define the *Nikon N-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **y** (ArrayLike) – Reflectance y, "y = 0.18" is equivalent to Stop 0.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the non-linear *Nikon N-Log* data x is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level :math`in` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Nikon N-Log* constants.

Returns *N-Log* 10-bit equivalent code value x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Nikon18]

Examples

```
>>> log_encoding_NLog(0.18)
0.3636677...
```

colour.models.log_decoding_NLog

colour.models.**log_decoding_NLog**(*x: ArrayLike, bit_depth: int = 10, in_normalised_code_value: bool = True, out_reflection: bool = True, constants: colour.utilities.data_structures.Structure = CONSTANTS_NLOG*) → NDArrayFloat

Define the *Nikon N-Log* log decoding curve / electro-optical transfer function.

Parameters

- **x** (ArrayLike) – *N-Log* 10-bit equivalent code value x
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the non-linear *Nikon N-Log* data x is encoded as normalised code values.
- **out_reflection** (bool) – Whether the light level :math`in` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Nikon N-Log* constants.

Returns Reflectance y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[Nikon18]

Examples

```
>>> log_decoding_NLog(0.36366777011713869)
0.1799999...
```

colour.models.log_encoding_Panalog

colour.models.**log_encoding_Panalog**(*x: ArrayLike*, *black_offset: ArrayLike = 10 ** 64 - 681 / 444*) → NDArrayFloat

Define the *Panalog* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **black_offset** (ArrayLike) – Black offset.

Returns Non-linear data y.

Return type numpy.ndarray

Warning: These are estimations known to be close enough, the actual log encoding curves are not published.

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_Panalog(0.18)
0.3745767...
```

colour.models.log_decoding_Panalog

colour.models.**log_decoding_Panalog**(*y: ArrayLike*, *black_offset: ArrayLike = 10 ** 64 - 681 / 444*) → NDArrayFloat

Define the *Panalog* log decoding curve / electro-optical transfer function.

Parameters

- **y** (ArrayLike) – Non-linear data y.
- **black_offset** (ArrayLike) – Black offset.

Returns Linear data x.

Return type numpy.ndarray

Warning: These are estimations known to be close enough, the actual log encoding curves are not published.

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_Panalog(0.374576791382298)
0.1...
```

colour.models.log_encoding_PivotedLog

colour.models.**log_encoding_PivotedLog**(*x: ArrayLike*, *log_reference: float = 445*, *linear_reference: float = 0.18*, *negative_gamma: float = 0.6*, *density_per_code_value: float = 0.002*) → NDArrayFloat

Define the *Josh Pines* style *Pivoted Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **log_reference** (float) – Log reference.
- **linear_reference** (float) – Linear reference.
- **negative_gamma** (float) – Negative gamma.
- **density_per_code_value** (float) – Density per code value.

Returns Non-linear data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_PivotedLog(0.18)
0.4349951...
```

colour.models.log_decoding_PivotedLog

colour.models.**log_decoding_PivotedLog**(*y: ArrayLike*, *log_reference: float = 445*, *linear_reference: float = 0.18*, *negative_gamma: float = 0.6*, *density_per_code_value: float = 0.002*) → NDArrayFloat

Define the *Josh Pines* style *Pivoted Log* log decoding curve / electro-optical transfer function.

Parameters

- **y** (ArrayLike) – Non-linear data y.
- **log_reference** (float) – Log reference.
- **linear_reference** (float) – Linear reference.
- **negative_gamma** (float) – Negative gamma.
- **density_per_code_value** (float) – Density per code value.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_PivotedLog(0.434995112414467)
0.1...
```

colour.models.log_encoding_Protune

`colour.models.log_encoding_Protune(x: ArrayLike) → NDArrayFloat`

Define the *Protune* log encoding curve / opto-electronic transfer function.

Parameters **x** (ArrayLike) – Linear data x.

Returns Non-linear data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[GoProDM16]

Examples

```
>>> log_encoding_Protune(0.18)
0.6456234...
```

colour.models.log_decoding_Protune

`colour.models.log_decoding_Protune(y: ArrayLike) → NDArrayFloat`

Define the *Protune* log decoding curve / electro-optical transfer function.

Parameters **y** (ArrayLike) – Non-linear data y.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[GoProDM16]

Examples

```
>>> log_decoding_Protune(0.645623486803636)
0.1...
```

colour.models.log_encoding_REDLog

colour.models.**log_encoding_REDLog**(*x: ArrayLike*, *black_offset: ArrayLike = 10 ** 0 - 1023 / 511*) → NDArrayFloat

Define the *REDLog* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **black_offset** (ArrayLike) – Black offset.

Returns Non-linear data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_REDLog(0.18)
0.6376218...
```

colour.models.log_decoding_REDLog

colour.models.**log_decoding_REDLog**(*y: ArrayLike*, *black_offset: ArrayLike = 10 ** 0 - 1023 / 511*) → NDArrayFloat

Define the *REDLog* log decoding curve / electro-optical transfer function.

Parameters

- **y** (ArrayLike) – Non-linear data y.
- **black_offset** (ArrayLike) – Black offset.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_REDLog(0.637621845988175)
0.1...
```

colour.models.log_encoding_REDLogFilm

colour.models.**log_encoding_REDLogFilm**(*x: ArrayLike, black_offset: ArrayLike = 10 ** 95 - 685 / 300*) → NDArrayFloat

Define the *REDLogFilm* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Linear data x.
- **black_offset** (ArrayLike) – Black offset.

Returns Non-linear data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_REDLogFilm(0.18)
0.4573196...
```

colour.models.log_decoding_REDLogFilm

colour.models.**log_decoding_REDLogFilm**(*y: ArrayLike*, *black_offset: ArrayLike = 10 ** 95 - 685 / 300*) → NDArrayFloat

Define the *REDLogFilm* log decoding curve / electro-optical transfer function.

Parameters

- **y** (ArrayLike) – Non-linear data y.
- **black_offset** (ArrayLike) – Black offset.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_REDLogFilm(0.457319613085418)
0.1799999...
```

colour.models.log_encoding_SLog

colour.models.**log_encoding_SLog**(*x: ArrayLike*, *bit_depth: int = 10*, *out_normalised_code_value: bool = True*, *in_reflection: bool = True*) → NDArrayFloat

Define the *Sony S-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Reflection or $IRE/100$ input light level x to a camera.
- **bit_depth** (int) – Bit-depth used for conversion.

- **out_normalised_code_value** (bool) – Whether the non-linear *Sony S-Log* data y is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level x to a camera is reflection.

Returns Non-linear *Sony S-Log* data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_encoding_SLog(0.18)
0.3849708...
```

The values of *IRE and CV of S-Log2 @ISO800* table in [SonyCorporation12] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_SLog(x, 10, False) * 100).astype(np.int_)
array([ 3, 38, 65])
>>> np.around(log_encoding_SLog(x) * (2**10 - 1)).astype(np.int_)
array([ 90, 394, 636])
```

colour.models.log_decoding_SLog

colour.models.**log_decoding_SLog**(*y: ArrayLike, bit_depth: int = 10, in_normalised_code_value: bool = True, out_reflection: bool = True*) → NDArrayFloat

Define the *Sony S-Log* log decoding curve / electro-optical transfer function.

Parameters

- **y** (ArrayLike) – Non-linear *Sony S-Log* data y.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the non-linear *Sony S-Log* data y is encoded as normalised code values.
- **out_reflection** (bool) – Whether the light level x to a camera is reflection.

Returns Reflection or $IRE/100$ input light level x to a camera.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_decoding_SLog(0.384970815928670)
0.1...
```

colour.models.log_encoding_SLog2

colour.models.**log_encoding_SLog2**(*x: ArrayLike*, *bit_depth: int = 10*, *out_normalised_code_value: bool = True*, *in_reflection: bool = True*) → NDArrayFloat

Define the *Sony S-Log2* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Reflection or $IRE/100$ input light level x to a camera.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the non-linear *Sony S-Log2* data y is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level x to a camera is reflection.

Returns Non-linear *Sony S-Log2* data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_encoding_SLog2(0.18)
0.3395325...
```

The values of *IRE and CV of S-Log2 @ISO800* table in [SonyCorporation12] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_SLog2(x, 10, False) * 100).astype(np.int_)
array([ 3, 32, 59])
>>> np.around(log_encoding_SLog2(x) * (2**10 - 1)).astype(np.int_)
array([ 90, 347, 582])
```

colour.models.log_decoding_SLog2

colour.models.**log_decoding_SLog2**(*y: ArrayLike*, *bit_depth: int = 10*, *in_normalised_code_value: bool = True*, *out_reflection: bool = True*) → NDArrayFloat

Define the *Sony S-Log2* log decoding curve / electro-optical transfer function.

Parameters

- **y** (ArrayLike) – Non-linear *Sony S-Log2* data y.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the non-linear *Sony S-Log2* data y is encoded as normalised code values.
- **out_reflection** (bool) – Whether the light level x to a camera is reflection.

Returns Reflection or $IRE/100$ input light level x to a camera.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_decoding_SLog2(0.339532524633774)
0.1...
```

colour.models.log_encoding_SLog3

colour.models.**log_encoding_SLog3**(*x: ArrayLike*, *bit_depth: int = 10*, *out_normalised_code_value: bool = True*, *in_reflection: bool = True*) → NDArrayFloat

Define the *Sony S-Log3* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (ArrayLike) – Reflection or $IRE/100$ input light level x to a camera.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the non-linear *Sony S-Log3* data y is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level x to a camera is reflection.

Returns Non-linear *Sony S-Log3* data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyCorporationc]

Examples

```
>>> log_encoding_SLog3(0.18)
0.4105571...
```

The values of *S-Log3 10bit code values (18%, 90%)* table in [SonyCorporationc] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_SLog3(x, 10, False) * 100).astype(np.int_)
array([ 4, 41, 61])
>>> np.around(log_encoding_SLog3(x) * (2**10 - 1)).astype(np.int_)
array([ 95, 420, 598])
```

colour.models.log_decoding_SLog3

colour.models.**log_decoding_SLog3**(*y: ArrayLike*, *bit_depth: int = 10*, *in_normalised_code_value: bool = True*, *out_reflection: bool = True*) → NDArrayFloat

Define the *Sony S-Log3* log decoding curve / electro-optical transfer function.

Parameters

- **y** (ArrayLike) – Non-linear *Sony S-Log3* data y.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the non-linear *Sony S-Log3* data y is encoded as normalised code values.
- **out_reflection** (bool) – Whether the light level x to a camera is reflection.

Returns Reflection or $IRE/100$ input light level x to a camera.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyCorporationc]

Examples

```
>>> log_decoding_SLog3(0.410557184750733)
0.1...
```

colour.models.log_encoding_VLog

colour.models.**log_encoding_VLog**(*L_in: ArrayLike*, *bit_depth: int = 10*, *out_normalised_code_value: bool = True*, *in_reflection: bool = True*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_VLOG*) → NDArrayFloat

Define the *Panasonic V-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **L_in** (ArrayLike) – Linear reflection data :math`L_{in}`.
- **bit_depth** (int) – Bit-depth used for conversion.
- **out_normalised_code_value** (bool) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded as normalised code values.
- **in_reflection** (bool) – Whether the light level :math`L_{in}` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Panasonic V-Log* constants.

Returns Non-linear data V_{out}.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_in	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V_out	[0, 1]	[0, 1]

References

[Panasonic14]

Examples

```
>>> log_encoding_VLog(0.18)
0.4233114...
```

The values of *Fig.2.2 V-Log Code Value* table in [Panasonic14] are obtained as follows:

```
>>> L_in = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_VLog(L_in, 10, False) * 100).astype(np.int_)
array([ 7, 42, 61])
>>> np.around(log_encoding_VLog(L_in) * (2**10 - 1)).astype(np.int_)
array([128, 433, 602])
>>> np.around(log_encoding_VLog(L_in) * (2**12 - 1)).astype(np.int_)
array([ 512, 1733, 2409])
```

Note that some values in the last column values of *Fig.2.2 V-Log Code Value* table in [Panasonic14] are different by a code: [512, 1732, 2408].

colour.models.log_decoding_VLog

colour.models.**log_decoding_VLog**(*V_out: ArrayLike*, *bit_depth: int = 10*, *in_normalised_code_value: bool = True*, *out_reflection: bool = True*, *constants: colour.utilities.data_structures.Structure = CONSTANTS_VLOG*) → NDArrayFloat

Define the *Panasonic V-Log* log decoding curve / electro-optical transfer function.

Parameters

- **V_out** (ArrayLike) – Non-linear data V_{out}.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_normalised_code_value** (bool) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded as normalised code values.
- **out_reflection** (bool) – Whether the light level :math`L_{in}` to a camera is reflection.
- **constants** (colour.utilities.data_structures.Structure) – *Panasonic V-Log* constants.

Returns Linear reflection data :math`L_{in}`.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
V_out	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_in	[0, 1]	[0, 1]

References

[Panasonic14]

Examples

```
>>> log_decoding_VLog(0.423311448760136)
0.1799999...
```

colour.models.log_encoding_ViperLog

colour.models.**log_encoding_ViperLog**(*x: ArrayLike*) → NDArrayFloat

Define the *Viper Log* log encoding curve / opto-electronic transfer function.

Parameters x (ArrayLike) – Linear data x.

Returns Non-linear data y.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_ViperLog(0.18)
0.6360080...
```

colour.models.log_decoding_ViperLog

colour.models.**log_decoding_ViperLog**(*y: ArrayLike*) → NDArrayFloat

Define the *Viper Log* log decoding curve / electro-optical transfer function.

Parameters **y** (ArrayLike) – Non-linear data y.

Returns Linear data x.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_ViperLog(0.636008067010413)
0.1799999...
```

Colour Encodings

Y'CbCr Colour Encoding

colour

WEIGHTS_YCBCR	Implement a delimiter and case-insensitive dict-like object with support for slugs, i.e. *SEO* friendly and human-readable version of the keys but also canonical keys, i.e. slugified keys without delimiters.
matrix_YCbCr([K, bits, is_legal, is_int])	Compute the *Y'CbCr* to *R'G'B'* matrix for given weights, bit-depth, range legality and representation.
offset_YCbCr([bits, is_legal, is_int])	Compute the *R'G'B'* to *Y'CbCr* offsets for given bit-depth, range legality and representation.
RGB_to_YCbCr(RGB[, K, in_bits, in_legal, ...])	Convert 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, ...])	Convert an array of *Y'CbCr* colour encoding values to the corresponding *R'G'B'* values array.
RGB_to_YcCbcCrc(RGB[, out_bits, out_legal, ...])	Convert an array of *RGB* linear values to the corresponding *Yc'Cbc'Crc'* colour encoding values array.
YcCbcCrc_to_RGB(YcCbcCrc[, in_bits, ...])	Convert an array of *Yc'Cbc'Crc'* colour encoding values to the corresponding *RGB* array of linear values.

colour.WEIGHTS_YCBCR

colour.**WEIGHTS_YCBCR = CanonicalMapping({'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'ITU-R BT.2020': ..., 'SMPTE-240M': ...})**

Implement a delimiter and case-insensitive dict-like object with support for slugs, i.e. *SEO* friendly and human-readable version of the keys but also canonical keys, i.e. slugified keys without delimiters.

The item keys are expected to be str-like objects thus supporting the str.lower() method. Setting items is done by using the given keys. Retrieving or deleting an item and testing whether an item exist is done by transforming the item's key in a sequence as follows:

- *Original Key*
- *Lowercase Key*
- *Slugified Key*
- *Canonical Key*

For example, given the McCamy 1992 key:

- *Original Key* : McCamy 1992
- *Lowercase Key* : mccamy 1992

- *Slugified Key* : mccamy-1992
- *Canonical Key* : mccamy1992

Parameters

- **data** – Data to store into the delimiter and case-insensitive dict-like object at initialisation.
- **kwargs** – Key / value pairs to store into the mapping at initialisation.

Attributes

- data

Methods

- __init__()
- __repr__()
- __setitem__()
- __getitem__()
- __delitem__()
- __contains__()
- __iter__()
- __len__()
- __eq__()
- __ne__()
- copy()
- lower_keys()
- lower_items()
- slugified_keys()
- slugified_items()
- canonical_keys()
- canonical_items()

Examples

```
>>> methods = CanonicalMapping({"McCamy 1992": 1, "Hernandez 1999": 2})
>>> methods["mccamy 1992"]
1
>>> methods["MCCAMY 1992"]
1
>>> methods["mccamy-1992"]
1
>>> methods["mccamy1992"]
1
```

colour.matrix_YCbCr

`colour.matrix_YCbCr`(*K: NDArrayFloat = WEIGHTS_YCBCR['ITU-R BT.709'], bits: int = 8, is_legal: bool = False, is_int: bool = False*) → NDArrayFloat

Compute the *Y'CbCr* to *R'G'B'* matrix for given weights, bit-depth, range legality and representation.

The related offset for the *R'G'B'* to *Y'CbCr* matrix can be computed with the `colour.offset_YCbCr()` definition.

Parameters

- **K** (`NDArrayFloat`) – Luma weighting coefficients of red and blue. See `colour.WEIGHTS_YCBCR` for presets. Default is *(0.2126, 0.0722)*, the weightings for *ITU-R BT.709*.
- **bits** (`int`) – Bit-depth of the *Y'CbCr* colour encoding ranges array.
- **is_legal** (`bool`) – Whether the *Y'CbCr* colour encoding ranges array is legal.
- **is_int** (`bool`) – Whether the *Y'CbCr* colour encoding ranges array represents int code values.

Returns *Y'CbCr* matrix.

Return type `numpy.ndarray`

Examples

```
>>> matrix_YCbCr()
array([[  1.0000000...e+00,   ...,   1.5748000...e+00],
       [  1.0000000...e+00,  -1.8732427...e-01,  -4.6812427...e-01],
       [  1.0000000...e+00,   1.8556000...e+00,   ...]])
>>> matrix_YCbCr(K=WEIGHTS_YCBCR["ITU-R BT.601"])
array([[  1.0000000...e+00,   ...,   1.4020000...e+00],
       [  1.0000000...e+00,  -3.4413628...e-01,  -7.1413628...e-01],
       [  1.0000000...e+00,   1.7720000...e+00,   ...]])
>>> matrix_YCbCr(is_legal=True)
array([[  1.1643835...e+00,   ...,   1.7927410...e+00],
       [  1.1643835...e+00,  -2.1324861...e-01,  -5.3290932...e-01],
       [  1.1643835...e+00,   2.1124017...e+00,   ...]])
```

Matching the default output of the `colour.RGB_to_YCbCr()` is done as follows:

```
>>> from colour.algebra import vector_dot
>>> from colour.utilities import as_int_array
>>> RGB = np.array([1.0, 1.0, 1.0])
>>> RGB_to_YCbCr(RGB)
array([ 0.9215686...,  0.5019607...,  0.5019607...])
>>> YCbCr = vector_dot(np.linalg.inv(matrix_YCbCr(is_legal=True)), RGB)
>>> YCbCr += offset_YCbCr(is_legal=True)
>>> YCbCr
array([ 0.9215686...,  0.5019607...,  0.5019607...])
```

Matching the int output of the `colour.RGB_to_YCbCr()` is done as follows:

```
>>> RGB = np.array([102, 0, 51])
>>> RGB_to_YCbCr(RGB, in_bits=8, in_int=True, out_bits=8, out_int=True)
...
array([ 38, 140, 171])
>>> YCbCr = vector_dot(np.linalg.inv(matrix_YCbCr(is_legal=True)), RGB)
```

(continues on next page)

(continued from previous page)

```
>>> YCbCr += offset_YCbCr(is_legal=True, is_int=True)
>>> as_int_array(np.around(YCbCr))
...
array([ 38, 140, 171])
```

colour.offset_YCbCr

colour.**offset_YCbCr**(*bits: int = 8, is_legal: bool = False, is_int: bool = False*) → NDArrayFloat

Compute the *R'G'B'* to *Y'CbCr* offsets for given bit-depth, range legality and representation.

The related *R'G'B'* to *Y'CbCr* matrix can be computed with the `colour.matrix_YCbCr()` definition.

Parameters

- **bits** (int) – Bit-depth of the *Y'CbCr* colour encoding ranges array.
- **is_legal** (bool) – Whether the *Y'CbCr* colour encoding ranges array is legal.
- **is_int** (bool) – Whether the *Y'CbCr* colour encoding ranges array represents int code values.

Returns *Y'CbCr* matrix.

Return type `numpy.ndarray`

Examples

```
>>> offset_YCbCr()
array([ 0.,  0.,  0.])
>>> offset_YCbCr(is_legal=True)
array([ 0.0627451...,  0.5019607...,  0.5019607...])
```

colour.RGB_to_YCbCr

colour.**RGB_to_YCbCr**(*RGB: ArrayLike, K: NDArrayFloat = WEIGHTS_YCBCR['ITU-R BT.709'], in_bits: int = 10, in_legal: bool = False, in_int: bool = False, out_bits: int = 8, out_legal: bool = True, out_int: bool = False, clamp_int: bool = True, **kwargs: Any*) → NDArrayReal

Convert an array of *R'G'B'* values to the corresponding *Y'CbCr* colour encoding values array.

Parameters

- **RGB** (ArrayLike) – Input *R'G'B'* array of floats or int values.
- **K** (NDArrayFloat) – Luma weighting coefficients of red and blue. See `colour.WEIGHTS_YCBCR` for presets. Default is *(0.2126, 0.0722)*, the weightings for *ITU-R BT.709*.
- **in_bits** (int) – Bit-depth for int 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) – Whether to treat the input values as legal range. Default is *False*.
- **in_int** (bool) – Whether to treat the input values as `in_bits` int code values. Default is *False*.

- **out_bits** (int) – Bit-depth for int 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) – Whether to return legal range values. Default is *True*.
- **out_int** (bool) – Whether to return values as out_bits int code values. Default is *False*.
- **clamp_int** (bool) – Whether to clamp int output to allowable range for out_bits. Default is *True*.
- **in_range** – 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 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.ranges_YCbCr() definition.
- **kwargs** (Any) –

Returns *Y'CbCr* colour encoding array of int or float values.

Return type numpy.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

Domain *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
YCbCr	[0, 1]	[0, 1]

* This definition has input and output int switches, thus the domain-range scale information is only given for the floating point mode.

- The default arguments, **{'in_bits': 10, 'in_legal': False, 'in_int': False, 'out_bits': 8, 'out_legal': True, 'out_int': False} transform a float *R'G'B'* input array normalised to domain [0, 1] (in_bits is ignored) to a float *Y'CbCr* output array where *Y'* is normalised to range [16 / 255, 235 / 255] and *Cb* and *Cr* are normalised to range [16 / 255, 240./255]. The float values are calculated based on an [0, 255] int range, but no 8-bit quantisation or clamping are performed.

References

[InternationalTUnion11c], [InternationalTUnion15b], [SocietyoMPaTEngineers99], [Wikipedia04e]

Examples

```
>>> RGB = np.array([1.0, 1.0, 1.0])
>>> RGB_to_YCbCr(RGB)
array([ 0.9215686...,  0.5019607...,  0.5019607...])
```

Matching the 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 the 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 int 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 *Recommendation ITU-T T.871*

```
>>> RGB = np.array([102, 0, 51])
>>> RGB_to_YCbCr(
...     RGB,
...     K=WEIGHTS_YCBCR["ITU-R BT.601"],
...     in_range=(0, 255),
...     out_range=(0, 255, 0.5, 255.5),
...     out_int=True,
... )
...
array([ 36, 136, 175]...)
```

Note the use of [0.5, 255.5] for the *Cb / Cr* range, 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 int code value to represent achromatic colours. This does however create the possibility of output int codes with value of 256, which cannot be stored in 8-bit int representation. *Recommendation ITU-T T.871* specifies these should be clamped to 255, which is applied with the default `clamp_int=True`.

These *JFIF JPEG* ranges are also obtained as follows:

```
>>> RGB_to_YCbCr(
...     RGB,
...     K=WEIGHTS_YCBCR["ITU-R BT.601"],
...     in_bits=8,
...     in_int=True,
...     out_legal=False,
```

(continues on next page)

(continued from previous page)

```
...     out_int=True,
... )
...
array([ 36, 136, 175]...)
```

colour.YCbCr_to_RGB

colour.**YCbCr_to_RGB**(*YCbCr: ArrayLike, K: NDArrayFloat = WEIGHTS_YCBCR['ITU-R BT.709'], in_bits: int = 8, in_legal: bool = True, in_int: bool = False, out_bits: int = 10, out_legal: bool = False, out_int: bool = False, clamp_int: bool = True, **kwargs: Any*) → NDArrayReal

Convert an array of *Y'CbCr* colour encoding values to the corresponding *R'G'B'* values array.

Parameters

- **YCbCr** (ArrayLike) – Input *Y'CbCr* colour encoding array of int or float values.
- **K** (NDArrayFloat) – Luma weighting coefficients of red and blue. See colour.WEIGHTS_YCBCR for presets. Default is *(0.2126, 0.0722)*, the weightings for *ITU-R BT.709*.
- **in_bits** (int) – Bit-depth for int input, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Default is *8*.
- **in_legal** (bool) – Whether to treat the input values as legal range. Default is *True*.
- **in_int** (bool) – Whether to treat the input values as in_bits int code values. Default is *False*.
- **out_bits** (int) – Bit-depth for int 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) – Whether to return legal range values. Default is *False*.
- **out_int** (bool) – Whether to return values as out_bits int code values. Default is *False*.
- **clamp_int** (bool) – Whether to clamp int output to allowable range for out_bits. Default is *True*.
- **in_range** – 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.ranges_YCbCr() definition.
- **out_range** – 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.
- **kwargs** (Any) –

Returns *R'G'B'* array of int or float values.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
YCbCr	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

* This definition has input and output int switches, thus the domain-range scale information is only given for the floating point mode.

> **Warning:** For *Recommendation ITU-R BT.2020*, `colour.YCbCr_to_RGB()` definition is only applicable to the non-constant luminance implementation. `colour.YcCbcCrc_to_RGB()` definition should be used for the constant luminance case as per [InternationalTUnion15a].

References

[InternationalTUnion11c], [InternationalTUnion15b], [SocietyoMPaTEngineers99], [Wikipedia04e]

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.RGB_to_YcCbcCrc

`colour.RGB_to_YcCbcCrc`(*RGB: ArrayLike*, *out_bits: int = 10*, *out_legal: bool = True*, *out_int: bool = False*, *is_12_bits_system: bool = False*, ***kwargs: Any*) → NDArrayReal

Convert an array of *RGB* linear values to the corresponding *Yc'Cbc'Crc'* colour encoding values array.

Parameters

- **RGB** (ArrayLike) – Input *RGB* array of linear float values.
- **out_bits** (int) – Bit-depth for int 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) – Whether to return legal range values. Default is *True*.
- **out_int** (bool) – Whether to return values as `out_bits` int code values. Default is *False*.
- **is_12_bits_system** (bool) – *Recommendation ITU-R BT.2020* OETF (OECF) adopts different parameters for 10 and 12 bit systems. Default is *False*.
- **out_range** – 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.ranges_YCbCr()` definition.
- **kwargs** (Any) –

Returns *Yc'Cbc'Crc'* colour encoding array of int or float values.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
YcCbcCrc	[0, 1]	[0, 1]

* This definition has input and output int switches, thus the domain-range scale information is only given for the floating point mode.

Warning: This definition is specifically for usage with *Recommendation ITU-R BT.2020* when adopting the constant luminance implementation.

References

[InternationalTUnion15a], [Wikipedia04e]

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: ArrayLike, in_bits: int = 10, in_legal: bool = True, in_int: bool = False, is_12_bits_system: bool = False, **kwargs: Any*) → NDArrayFloat

Convert an array of *Yc'Cbc'Crc'* colour encoding values to the corresponding *RGB* array of linear values.

Parameters

- **YcCbcCrc** (ArrayLike) – Input *Yc'Cbc'Crc'* colour encoding array of linear float values.
- **in_bits** (int) – Bit-depth for int 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) – Whether to treat the input values as legal range. Default is *False*.

- **in_int** (bool) – Whether to treat the input values as in_bits int code values. Default is *False*.
- **is_12_bits_system** (bool) – *Recommendation ITU-R BT.2020* EOTF (EOCF) adopts different parameters for 10 and 12 bit systems. Default is *False*.
- **in_range** – 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.ranges_YCbCr() definition.
- **kwargs** (Any) –

Returns *RGB* array of linear float values.

Return type numpy.ndarray

Notes

Domain *	Scale - Reference	Scale - 1
YcCbcCrc	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

* This definition has input and output int switches, thus the domain-range scale information is only given for the floating point mode.

Warning: This definition is specifically for usage with *Recommendation ITU-R BT.2020* when adopting the constant luminance implementation.

References

[InternationalTUnion15a], [Wikipedia04e]

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])	Convert 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])	Convert 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])	Return the code value CV range for given bit-depth, range legality and representation.

colour.full_to_legal

colour.**full_to_legal**(*CV: ArrayLike*, *bit_depth: int = 10*, *in_int: bool = False*, *out_int: bool = False*) → NDArrayReal

Convert 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** (ArrayLike) – Full range code value CV or float equivalent of a code value at a given bit-depth.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_int** (bool) – Whether to treat the input value as int code value or float equivalent of a code value at a given bit-depth.
- **out_int** (bool) – Whether to return value as int 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 numpy.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: ArrayLike*, *bit_depth: int = 10*, *in_int: bool = False*, *out_int: bool = False*) → NDArrayReal

Convert 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** (ArrayLike) – Legal range code value CV or float equivalent of a code value at a given bit-depth.
- **bit_depth** (int) – Bit-depth used for conversion.
- **in_int** (bool) – Whether to treat the input value as int code value or float equivalent of a code value at a given bit-depth.
- **out_int** (bool) – Whether to return value as int 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 numpy.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: int = 10*, *is_legal: bool = False*, *is_int: bool = False*) → NDArrayReal

Return the code value CV range for given bit-depth, range legality and representation.

Parameters

- **bit_depth** (int) – Bit-depth of the code value CV range.
- **is_legal** (bool) – Whether the code value CV range is legal.
- **is_int** (bool) – Whether the code value CV range represents int code values.

Returns Code value CV range.

Return type numpy.ndarray

Examples

```
>>> CV_range(8, True, True)
array([ 16, 235])
>>> CV_range(8, True, False)
array([ 0.0627451...,  0.9215686...])
>>> CV_range(10, False, False)
array([ 0.,  1.])
```

YCoCg Colour Encoding

`colour`

RGB_to_YCoCg(RGB)	Convert an array of *R'G'B'* values to the corresponding *YCoCg* colour encoding values array.
YCoCg_to_RGB(YCoCg)	Convert an array of *YCoCg* colour encoding values to the corresponding *R'G'B'* values array.

colour.RGB_to_YCoCg

colour.**RGB_to_YCoCg**(*RGB: ArrayLike*) → NDArrayFloat

Convert an array of *R'G'B'* values to the corresponding *YCoCg* colour encoding values array.

Parameters **RGB** (ArrayLike) – Input *R'G'B'* array.

Returns *YCoCg* colour encoding array.

Return type numpy.ndarray

References

[MS03]

Examples

```
>>> RGB_to_YCoCg(np.array([1.0, 1.0, 1.0]))
array([ 1.,  0.,  0.])
>>> RGB_to_YCoCg(np.array([0.75, 0.5, 0.5]))
array([ 0.5625,  0.125 , -0.0625])
```

colour.YCoCg_to_RGB

colour.**YCoCg_to_RGB**(*YCoCg: ArrayLike*) → NDArrayFloat

Convert an array of *YCoCg* colour encoding values to the corresponding *R'G'B'* values array.

Parameters **YCoCg** (ArrayLike) – *YCoCg* colour encoding array.

Returns Output *R'G'B'* array.

Return type numpy.ndarray

References

[MS03]

Examples

```
>>> YCoCg_to_RGB(np.array([1.0, 0.0, 0.0]))
array([ 1.,  1.,  1.])
>>> YCoCg_to_RGB(np.array([0.5625, 0.125, -0.0625]))
array([ 0.75,  0.5 ,  0.5 ])
```

IC_TC_P Colour Encoding

colour

RGB_to_ICtCp(RGB[, method, L_p])	Convert from *ITU-R BT.2020* colourspace to IC_TC_P colour encoding.
ICtCp_to_RGB(ICtCp[, method, L_p])	Convert from IC_TC_P colour encoding to *ITU-R BT.2020* colourspace.
XYZ_to_ICtCp(XYZ[, illuminant, ...])	Convert from *CIE XYZ* tristimulus values to IC_TC_P colour encoding.
ICtCp_to_XYZ(ICtCp[, illuminant, ...])	Convert from IC_TC_P colour encoding to *CIE XYZ* tristimulus values.

colour.RGB_to_ICtCp

colour.**RGB_to_ICtCp**(*RGB: ArrayLike, method: Union[Literal['Dolby 2016', 'ITU-R BT.2100-1 HLG', 'ITU-R BT.2100-1 PQ', 'ITU-R BT.2100-2 HLG', 'ITU-R BT.2100-2 PQ'], str] = 'Dolby 2016', L_p: float = 10000*) → NDArrayFloat

Convert from *ITU-R BT.2020* colourspace to IC_TC_P colour encoding.

Parameters

- **RGB** (ArrayLike) – *ITU-R BT.2020* colourspace array.
- **method** (Union[Literal['Dolby 2016', 'ITU-R BT.2100-1 HLG', 'ITU-R BT.2100-1 PQ', 'ITU-R BT.2100-2 HLG', 'ITU-R BT.2100-2 PQ'], str]) – Computation method. *Recommendation ITU-R BT.2100* defines multiple variants of the IC_TC_P colour encoding:
  - *ITU-R BT.2100-1*
    - *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) and the IC_TC_P matrix from [Dolby16]: *Dolby 2016*, *ITU-R BT.2100-1 PQ*, *ITU-R BT.2100-2 PQ* methods.
    - *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF) and the IC_TC_P matrix from [Dolby16]: *ITU-R BT.2100-1 HLG* method.
  - *ITU-R BT.2100-2*
    - *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) and the IC_TC_P matrix from [Dolby16]: *Dolby 2016*, *ITU-R BT.2100-1 PQ*, *ITU-R BT.2100-2 PQ* methods.

    * *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF) and a custom IC_TC_P matrix from [InternationalTUnion18]: *ITU-R BT.2100-2 HLG* method.

  - **L_p** (`float`) – Display peak luminance cd/m^2 for *SMPTE ST 2084:2014* non-linear encoding. This parameter should stay at its default $10000cd/m^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.

Returns IC_TC_P colour encoding array.

Return type `numpy.ndarray`

Warning: The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The *ITU-R BT.2100-1 PQ* and *ITU-R BT.2100-2 PQ* methods are aliases for the *Dolby 2016* method.
- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations. The effective domain of *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) is [0.0001, 10000].

Domain	Scale - Reference	Scale - 1
RGB	UN	UN

Range	Scale - Reference	Scale - 1
ICtCp	I : [0, 1] CT : [-1, 1] CP : [-1, 1]	I : [0, 1] CT : [-1, 1] CP : [-1, 1]

References

[Dolby16], [LPY+16]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_ICtCp(RGB)
array([ 0.0735136...,  0.0047525...,  0.0935159...])
>>> RGB_to_ICtCp(RGB, method="ITU-R BT.2100-2 HLG")
array([ 0.6256789..., -0.0198449...,  0.3591125...])
```

colour.ICtCp_to_RGB

`colour.ICtCp_to_RGB(ICtCp: ArrayLike, method: Union[Literal['Dolby 2016', 'ITU-R BT.2100-1 HLG', 'ITU-R BT.2100-1 PQ', 'ITU-R BT.2100-2 HLG', 'ITU-R BT.2100-2 PQ'], str] = 'Dolby 2016', L_p: float = 10000)` → NDArrayFloat

Convert from IC_TC_P colour encoding to *ITU-R BT.2020* colourspace.

Parameters

- **ICtCp** (ArrayLike) – IC_TC_P colour encoding array.
- **method** (Union[Literal['Dolby 2016', 'ITU-R BT.2100-1 HLG', 'ITU-R BT.2100-1 PQ', 'ITU-R BT.2100-2 HLG', 'ITU-R BT.2100-2 PQ'], str]) – Computation method. *Recommendation ITU-R BT.2100* defines multiple variants of the IC_TC_P colour encoding:
  - *ITU-R BT.2100-1*
    * *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) and the IC_TC_P matrix from [Dolby16]: *Dolby 2016*, *ITU-R BT.2100-1 PQ*, *ITU-R BT.2100-2 PQ* methods.
    * *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF) and the IC_TC_P matrix from [Dolby16]: *ITU-R BT.2100-1 HLG* method.
  - *ITU-R BT.2100-2*
    * *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) and the IC_TC_P matrix from [Dolby16]: *Dolby 2016*, *ITU-R BT.2100-1 PQ*, *ITU-R BT.2100-2 PQ* methods.
    * *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF) and a custom IC_TC_P matrix from [InternationalTUnion18]: *ITU-R BT.2100-2 HLG* method.
- **L_p** (float) – Display peak luminance cd/m^2 for *SMPTE ST 2084:2014* non-linear encoding. This parameter should stay at its default $10000 cd/m^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.

Returns *ITU-R BT.2020* colourspace array.

Return type numpy.ndarray

> **Warning:** The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The *ITU-R BT.2100-1 PQ* and *ITU-R BT.2100-2 PQ* methods are aliases for the *Dolby 2016* method.
- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
ICtCp	I : [0, 1] CT : [-1, 1] CP : [-1, 1]	I : [0, 1] CT : [-1, 1] CP : [-1, 1]

Range	Scale - Reference	Scale - 1
RGB	UN	UN

References

[Dolby16], [LPY+16]

Examples

```
>>> ICtCp = np.array([0.07351364, 0.00475253, 0.09351596])
>>> ICtCp_to_RGB(ICtCp)
array([ 0.4562052...,  0.0308107...,  0.0409195...])
>>> ICtCp = np.array([0.62567899, -0.01984490, 0.35911259])
>>> ICtCp_to_RGB(ICtCp, method="ITU-R BT.2100-2 HLG")
array([ 0.4562052...,  0.0308107...,  0.0409195...])
```

colour.XYZ_to_ICtCp

colour.**XYZ_to_ICtCp**(*XYZ: ArrayLike, illuminant=CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65'], chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02', method: Union[Literal['Dolby 2016', 'ITU-R BT.2100-1 HLG', 'ITU-R BT.2100-1 PQ', 'ITU-R BT.2100-2 HLG', 'ITU-R BT.2100-2 PQ'], str] = 'Dolby 2016', L_p: float = 10000*) → NDArrayFloat

Convert from *CIE XYZ* tristimulus values to IC_TC_P colour encoding.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **illuminant** – Source illuminant chromaticity coordinates.
- **chromatic_adaptation_transform** (Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]) – *Chromatic adaptation* transform.
- **method** (Union[Literal['Dolby 2016', 'ITU-R BT.2100-1 HLG', 'ITU-R BT.2100-1 PQ', 'ITU-R BT.2100-2 HLG', 'ITU-R BT.2100-2 PQ'], str]) – Computation method. *Recommendation ITU-R BT.2100* defines multiple variants of the IC_TC_P colour encoding:
  - *ITU-R BT.2100-1*
    - *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) and the IC_TC_P matrix from [Dolby16]: *Dolby 2016*, *ITU-R BT.2100-1 PQ*, *ITU-R BT.2100-2 PQ* methods.
    - *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF) and the IC_TC_P matrix from [Dolby16]: *ITU-R BT.2100-1 HLG* method.
  - *ITU-R BT.2100-2*
    - *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) and the IC_TC_P matrix from [Dolby16]: *Dolby 2016*, *ITU-R BT.2100-1 PQ*, *ITU-R BT.2100-2 PQ* methods.

    * *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF) and a custom IC_TC_P matrix from [InternationalTUnion18]: *ITU-R BT.2100-2 HLG* method.

- **L_p** (float) – Display peak luminance cd/m^2 for *SMPTE ST 2084:2014* non-linear encoding. This parameter should stay at its default $10000cd/m^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.

Returns IC_TC_P colour encoding array.

Return type numpy.ndarray

Warning: The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference*
- The *ITU-R BT.2100-1 PQ* and *ITU-R BT.2100-2 PQ* methods are aliases for the *Dolby 2016* method. and *1* scales are only indicative that the data is not affected by scale transformations. The effective domain of *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) is [0.0001, 10000].

Domain	Scale - Reference	Scale - 1
XYZ	UN	UN

Range	Scale - Reference	Scale - 1
ICtCp	I : [0, 1] CT : [-1, 1] CP : [-1, 1]	I : [0, 1] CT : [-1, 1] CP : [-1, 1]

References

[Dolby16], [LPY+16]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_ICtCp(XYZ)
array([ 0.0685809..., -0.0028384...,  0.0602098...])
>>> XYZ_to_ICtCp(XYZ, method="ITU-R BT.2100-2 HLG")
array([ 0.5924279..., -0.0374073...,  0.2512267...])
```

colour.ICtCp_to_XYZ

colour.**ICtCp_to_XYZ**(*ICtCp: ArrayLike, illuminant=CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65'], chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02', method: Union[Literal['Dolby 2016', 'ITU-R BT.2100-1 HLG', 'ITU-R BT.2100-1 PQ', 'ITU-R BT.2100-2 HLG', 'ITU-R BT.2100-2 PQ'], str] = 'Dolby 2016', L_p: float = 10000*) → NDArrayFloat

Convert from IC_TC_P colour encoding to *CIE XYZ* tristimulus values.

Parameters

- **ICtCp** (ArrayLike) – IC_TC_P colour encoding array.
- **illuminant** – Source illuminant chromaticity coordinates.
- **chromatic_adaptation_transform** (Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]) – *Chromatic adaptation* transform.
- **method** (Union[Literal['Dolby 2016', 'ITU-R BT.2100-1 HLG', 'ITU-R BT.2100-1 PQ', 'ITU-R BT.2100-2 HLG', 'ITU-R BT.2100-2 PQ'], str]) – Computation method. *Recommendation ITU-R BT.2100* defines multiple variants of the IC_TC_P colour encoding:
  - *ITU-R BT.2100-1*
    - *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) and the IC_TC_P matrix from [Dolby16]: *Dolby 2016*, *ITU-R BT.2100-1 PQ*, *ITU-R BT.2100-2 PQ* methods.
    - *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF) and the IC_TC_P matrix from [Dolby16]: *ITU-R BT.2100-1 HLG* method.
  - *ITU-R BT.2100-2*
    - *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF) and the IC_TC_P matrix from [Dolby16]: *Dolby 2016*, *ITU-R BT.2100-1 PQ*, *ITU-R BT.2100-2 PQ* methods.
    - *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF) and a custom IC_TC_P matrix from [InternationalTUnion18]: *ITU-R BT.2100-2 HLG* method.
- **L_p** (float) – Display peak luminance cd/m^2 for *SMPTE ST 2084:2014* non-linear encoding. This parameter should stay at its default $10000 cd/m^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.

Returns *CIE XYZ* tristimulus values.

Return type numpy.ndarray

> **Warning:** The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The *ITU-R BT.2100-1 PQ* and *ITU-R BT.2100-2 PQ* methods are aliases for the *Dolby 2016* method.
- The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
`ICtCp`	`I` : [0, 1] `CT` : [-1, 1] `CP` : [-1, 1]	`I` : [0, 1] `CT` : [-1, 1] `CP` : [-1, 1]

Range	Scale - Reference	Scale - 1
`XYZ`	`UN`	`UN`

References

[Dolby16], [LPY+16]

Examples

```
>>> ICtCp = np.array([0.06858097, -0.00283842, 0.06020983])
>>> ICtCp_to_XYZ(ICtCp)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
>>> ICtCp = np.array([0.59242792, -0.03740730, 0.25122675])
>>> ICtCp_to_XYZ(ICtCp, method="ITU-R BT.2100-2 HLG")
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

RGB Representations

Prismatic Colourspace

`colour`

`RGB_to_Prismatic`(RGB)	Convert from *RGB* colourspace to *Prismatic* $L\rho\gamma\beta$ colourspace array.
`Prismatic_to_RGB`(Lrgb)	Convert from *Prismatic* $L\rho\gamma\beta$ colourspace array to *RGB* colourspace.

colour.RGB_to_Prismatic

`colour.`**`RGB_to_Prismatic`**(*RGB: ArrayLike*) → NDArrayFloat

Convert from *RGB* colourspace to *Prismatic* $L\rho\gamma\beta$ colourspace array.

Parameters **`RGB`** (`ArrayLike`) – *RGB* colourspace array.

Returns *Prismatic* $L\rho\gamma\beta$ colourspace array.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lrgb	[0, 1]	[0, 1]

References

[SH15]

Examples

```
>>> RGB = np.array([0.25, 0.50, 0.75])
>>> RGB_to_Prismatic(RGB)
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: ArrayLike*) → NDArrayFloat

Convert from *Prismatic* $L\rho\gamma\beta$ colourspace array to *RGB* colourspace.

Parameters **`Lrgb`** (`ArrayLike`) – *Prismatic* $L\rho\gamma\beta$ colourspace array.

Returns *RGB* colourspace array.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
Lrgb	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[SH15]

Examples

```
>>> Lrgb = np.array([0.75000000, 0.16666667, 0.33333333, 0.50000000])
>>> Prismatic_to_RGB(Lrgb)
array([ 0.25...    ,  0.4999999...,  0.75...   ])
```

HSV Colourspace

colour

RGB_to_HSV(RGB)	Convert from *RGB* colourspace to *HSV* colourspace.
HSV_to_RGB(HSV)	Convert from *HSV* colourspace to *RGB* colourspace.

colour.RGB_to_HSV

colour.**RGB_to_HSV**(*RGB: ArrayLike*) → NDArrayFloat

Convert from *RGB* colourspace to *HSV* colourspace.

Parameters **RGB** (ArrayLike) – *RGB* colourspace array.

Returns *HSV* array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
HSV	[0, 1]	[0, 1]

References

[EasyRGBh], [Smi78], [Wikipedia03g]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_HSV(RGB)
array([ 0.9960394...,  0.9324630...,  0.4562051...])
```

colour.HSV_to_RGB

colour.**HSV_to_RGB**(*HSV: ArrayLike*) → NDArrayFloat

Convert from *HSV* colourspace to *RGB* colourspace.

Parameters **HSV** (ArrayLike) – *HSV* colourspace array.

Returns *RGB* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
HSV	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[EasyRGBe], [Smi78], [Wikipedia03g]

Examples

```
>>> HSV = np.array([0.99603944, 0.93246304, 0.45620519])
>>> HSV_to_RGB(HSV)
array([ 0.4562051...,  0.0308107...,  0.0409195...])
```

HSL Colourspace

colour

RGB_to_HSL(RGB)	Convert from *RGB* colourspace to *HSL* colourspace.
HSL_to_RGB(HSL)	Convert from *HSL* colourspace to *RGB* colourspace.

colour.RGB_to_HSL

colour.**RGB_to_HSL**(*RGB: ArrayLike*) → NDArrayFloat

Convert from *RGB* colourspace to *HSL* colourspace.

Parameters **RGB** (ArrayLike) – *RGB* colourspace array.

Returns *HSL* array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
HSL	[0, 1]	[0, 1]

References

[EasyRGBg], [Smi78], [Wikipedia03g]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_HSL(RGB)
array([ 0.9960394...,  0.8734714...,  0.2435079...])
```

colour.HSL_to_RGB

colour.**HSL_to_RGB**(*HSL: ArrayLike*) → NDArrayFloat

Convert from *HSL* colourspace to *RGB* colourspace.

Parameters **HSL** (ArrayLike) – *HSL* colourspace array.

Returns *RGB* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
HSL	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[EasyRGBd], [Smi78], [Wikipedia03g]

Examples

```
>>> HSL = np.array([0.99603944, 0.87347144, 0.24350795])
>>> HSL_to_RGB(HSL)
array([ 0.4562051...,  0.0308107...,  0.0409195...])
```

HCL Colourspace

colour

RGB_to_HCL(RGB[, gamma, Y_0])	Convert from *RGB* colourspace to *HCL* colourspace according to *Sarifuddin and Missaoui (2005)* method.
HCL_to_RGB(HCL[, gamma, Y_0])	Convert from *HCL* colourspace to *RGB* colourspace according to *Sarifuddin and Missaoui (2005)* method.

colour.RGB_to_HCL

colour.**RGB_to_HCL**(*RGB: ArrayLike, gamma: float = 3, Y_0: float = 100*) → NDArrayFloat

Convert from *RGB* colourspace to *HCL* colourspace according to *Sarifuddin and Missaoui (2005)* method.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array.
- **gamma** (float) – Non-linear lightness exponent matching *Lightness* L^*.
- **Y_0** (float) – White reference luminance Y_0.

Returns *HCL* array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
HCL	[0, 1]	[0, 1]

- This implementation used the equations given in [SM05b] and the corrections from [Sar21].

References

[SM05a], [SM05b], [Wikipedia15]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_HCL(RGB)
array([-0.0316785...,  0.2841715...,  0.2285964...])
```

colour.HCL_to_RGB

`colour.HCL_to_RGB`(*HCL: ArrayLike*, *gamma: float = 3*, *Y_0: float = 100*) → NDArrayFloat

Convert from *HCL* colourspace to *RGB* colourspace according to *Sarifuddin and Missaoui (2005)* method.

Parameters

- **HCL** (ArrayLike) – *HCL* colourspace array.
- **gamma** (float) – Non-linear lightness exponent matching *Lightness* L^*.
- **Y_0** (float) – White reference luminance Y_0.

Returns *RGB* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
HCL	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

- This implementation used the equations given in [SM05b] and the corrections from [Sar21].

References

[SM05a], [SM05b], [Wikipedia15]

Examples

```
>>> HCL = np.array([-0.03167854, 0.28417150, 0.22859647])
>>> HCL_to_RGB(HCL)
array([ 0.4562033...,  0.0308104...,  0.0409192...])
```

CMY Colourspace

colour

RGB_to_CMY(RGB)	Convert from *RGB* colourspace to *CMY* colourspace.
CMY_to_RGB(CMY)	Convert from *CMY* colourspace to *CMY* colourspace.
CMY_to_CMYK(CMY)	Convert from *CMY* colourspace to *CMYK* colourspace.
CMYK_to_CMY(CMYK)	Convert from *CMYK* colourspace to *CMY* colourspace.

colour.RGB_to_CMY

colour.**RGB_to_CMY**(*RGB: ArrayLike*) → NDArrayFloat

Convert from *RGB* colourspace to *CMY* colourspace.

Parameters **RGB** (ArrayLike) – *RGB* colourspace array.

Returns *CMY* array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

References

[EasyRGBf]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_CMY(RGB)
array([ 0.5437948...,  0.9691892...,  0.9590804...])
```

colour.CMY_to_RGB

colour.**CMY_to_RGB**(*CMY: ArrayLike*) → NDArrayFloat

Convert from *CMY* colourspace to *CMY* colourspace.

Parameters **CMY** (ArrayLike) – *CMY* colourspace array.

Returns *RGB* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[EasyRGBc]

Examples

```
>>> CMY = np.array([0.54379481, 0.96918929, 0.95908048])
>>> CMY_to_RGB(CMY)
array([ 0.4562051...,  0.0308107...,  0.0409195...])
```

colour.CMY_to_CMYK

colour.**CMY_to_CMYK**(*CMY: ArrayLike*) → NDArrayFloat

Convert from *CMY* colourspace to *CMYK* colourspace.

Parameters **CMY** (ArrayLike) – *CMY* colourspace array.

Returns *CMYK* array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
CMYK	[0, 1]	[0, 1]

References

[EasyRGBb]

Examples

```
>>> CMY = np.array([0.54379481, 0.96918929, 0.95908048])
>>> CMY_to_CMYK(CMY)
array([ 0.        ,  0.9324630...,  0.9103045...,  0.5437948...])
```

colour.CMYK_to_CMY

colour.**CMYK_to_CMY**(*CMYK: ArrayLike*) → NDArrayFloat

Convert from *CMYK* colourspace to *CMY* colourspace.

Parameters **CMYK** (ArrayLike) – *CMYK* colourspace array.

Returns *CMY* array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
CMYK	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

References

[EasyRGBa]

Examples

```
>>> CMYK = np.array([0.50000000, 0.00000000, 0.74400000, 0.01960784])
>>> CMYK_to_CMY(CMYK)
array([ 0.5098039...,  0.0196078...,  0.7490196...])
```

IHLS - Hanbury (2003)

colour

RGB_to_IHLS(RGB)	Convert from *RGB* colourspace to *IHLS* (Improved HLS) colourspace.
IHLS_to_RGB(HYS)	Convert from *IHLS* (Improved HLS) colourspace to *RGB* colourspace.

colour.RGB_to_IHLS

`colour.RGB_to_IHLS(RGB: ArrayLike) → NDArrayFloat`

Convert from *RGB* colourspace to *IHLS* (Improved HLS) colourspace.

Parameters **RGB** (ArrayLike) – *RGB* colourspace array.

Returns *HYS* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
HYS	[0, 1]	[0, 1]

References

[Han03]

Examples

```
>>> RGB = np.array([0.45595571, 0.03039702, 0.04087245])
>>> RGB_to_IHLS(RGB)
array([ 6.2616051...,  0.1216271...,  0.4255586...])
```

colour.IHLS_to_RGB

`colour.IHLS_to_RGB(HYS: ArrayLike) → NDArrayFloat`

Convert from *IHLS* (Improved HLS) colourspace to *RGB* colourspace.

Parameters **HYS** (ArrayLike) – *IHLS* colourspace array.

Returns *RGB* colourspace array.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
HYS	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Han03]

Examples

```
>>> HYS = np.array([6.26160518, 0.12162712, 0.42555869])
>>> IHLS_to_RGB(HYS)
array([ 0.4559557...,  0.0303970...,  0.0408724...])
```

Recommendation ITU-T H.273 Code points for Video Signal Type Identification

colour

COLOUR_PRIMARIES_ITUTH273	*ColourPrimaries* indicates the chromaticity coordinates of the source colour primaries as specified in Table 3 of [InternationalOfStandardization21] and [InternationalTUnion21] in terms of the CIE 1931 definition of x and y, which shall be interpreted as specified by *ISO/ CIE 11664-1*.
TRANSFER_CHARACTERISTICS_ITUTH273	*TransferCharacteristics*, as specified in Table 3 of [InternationalOfStandardization21] and [InternationalTUnion21], either indicates the reference opto-electronic transfer characteristic function of the source picture as a function of a source input linear optical intensity input Lc with a nominal real-valued range of 0 to 1 or indicates the inverse of the reference electro-optical transfer characteristic function as a function of an output linear optical intensity Lo with a nominal real-valued range of 0 to 1.
MATRIX_COEFFICIENTS_ITUTH273	*MatrixCoefficients* describes the matrix coefficients used in deriving luma and chroma signals from the green, blue and red or X, Y and Z primaries, as specified in Table 4 and equations 11 to 77 of [InternationalOfStandardization21] and [InternationalTUnion21].

colour.COLOUR_PRIMARIES_ITUTH273

colour.**COLOUR_PRIMARIES_ITUTH273 = {0: array('Reserved', dtype='<U8'), 1: array([[0.64, 0.33], [0.3 , 0.6], [0.15, 0.06]]), 2: array('Unspecified', dtype='<U11'), 3: array('Reserved', dtype='<U8'), 4: array([[0.67, 0.33], [0.21, 0.71], [0.14, 0.08]]), 5: array([[0.64, 0.33], [0.29, 0.6], [0.15, 0.06]]), 6: array([[0.63 , 0.34], [0.31 , 0.595], [0.155, 0.07]]), 7: array([[0.63 , 0.34], [0.31 , 0.595], [0.155, 0.07]]), 8: array([[0.681, 0.319], [0.243, 0.692], [0.145, 0.049]]), 9: array([[0.708, 0.292], [0.17 , 0.797], [0.131, 0.046]]), 10: array([[1., 0.], [0., 1.], [0., 0.]]), 11: array([[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.06]]), 12: array([[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.06]]), 22: array([[0.63 , 0.34], [0.295, 0.605], [0.155, 0.077]]), 23: array('Reserved', dtype='<U8')}**

ColourPrimaries indicates the chromaticity coordinates of the source colour primaries as specified in Table 3 of [InternationalOfStandardization21] and [InternationalTUnion21] in terms of the CIE 1931 definition of x and y, which shall be interpreted as specified by *ISO/ CIE 11664-1*.

References

[InternationalOfStandardization13] [InternationalOfStandardization20] [InternationalOfStandardization21], [InternationalTUnion21]

colour.TRANSFER_CHARACTERISTICS_ITUTH273

```
colour.TRANSFER_CHARACTERISTICS_ITUTH273 = {0: <function _reserved>, 1: <function
oetf_BT709>, 2: <function _unspecified>, 3: <function _reserved>, 4: <function
_clipped_domain_function.<locals>.wrapped>, 5: <function
_clipped_domain_function.<locals>.wrapped>, 6: <function oetf_BT601>, 7: <function
oetf_SMPTE240M>, 8: <function linear_function>, 9: <function oetf_H273_Log>, 10: <function
oetf_H273_LogSqrt>, 11: <function oetf_H273_IEC61966_2>, 12: <function oetf_BT1361>, 13:
<function oetf_H273_IEC61966_2>, 14: <function _clipped_domain_function.<locals>.wrapped>,
15: <function _clipped_domain_function.<locals>.wrapped>, 16: <function
eotf_inverse_ST2084>, 17: <function eotf_inverse_H273_ST428_1>, 18: <function
oetf_BT2100_HLG>, 19: <function _reserved>}
```

TransferCharacteristics, as specified in Table 3 of [InternationalOfStandardization21] and [InternationalTUnion21], either indicates the reference opto-electronic transfer characteristic function of the source picture as a function of a source input linear optical intensity input Lc with a nominal real-valued range of 0 to 1 or indicates the inverse of the reference electro-optical transfer characteristic function as a function of an output linear optical intensity Lo with a nominal real-valued range of 0 to 1.

Notes

- For simplicity, no clipping is implemented for *TransferCharacteristics 13* as it is a function of whether the context is *sRGB* or *sYCC*.
- For TransferCharacteristics equal to 18, the equations given in Table 3 are normalized for a source input linear optical intensity Lc with a nominal real-valued range of 0 to 1. An alternative scaling that is mathematically equivalent is used in ARIB STD-B67 (2015) with the source input linear optical intensity having a nominal real-valued range of 0 to 12.

References

[InternationalOfStandardization13] [InternationalOfStandardization20] [InternationalOfStandardization21], [InternationalTUnion21]

colour.MATRIX_COEFFICIENTS_ITUTH273

```
colour.MATRIX_COEFFICIENTS_ITUTH273 = {0: array('Identity', dtype='<U8'), 1: array([
0.2126, 0.0722]), 2: array('Unspecified', dtype='<U11'), 3: array('Reserved',
dtype='<U8'), 4: array([ 0.3 , 0.11]), 5: array([ 0.299, 0.114]), 6: array([ 0.299,
0.114]), 7: array([ 0.212, 0.087]), 8: array('YCgCo', dtype='<U5'), 9: array([ 0.2627,
0.0593]), 10: array([ 0.2627, 0.0593]), 11: array("Y'D'ZD'X", dtype='<U8'), 12: array('See
equations 32 to 37', dtype='<U22'), 13: array('See equations 32 to 37', dtype='<U22'), 14:
array('ICTCP', dtype='<U5'), 15: array('Reserved', dtype='<U8')}
```

MatrixCoefficients describes the matrix coefficients used in deriving luma and chroma signals from the green, blue and red or X, Y and Z primaries, as specified in Table 4 and equations 11 to 77 of [InternationalOfStandardization21] and [InternationalTUnion21].

Notes

- See `colour.WEIGHTS_YCBCR` attribute and the `colour.matrix_YCbCr()`, `colour.offset_YCbCr()`, `colour.RGB_to_YCbCr()`, `colour.YCbCr_to_RGB()`, `colour.RGB_to_YcCbcCrc()`, `colour.YcCbcCrc_to_RGB()` definitions for an implementation.

References

[InternationalOfStandardization13] [InternationalOfStandardization20] [InternationalOfStandardization21], [InternationalTUnion21]

Ancillary Objects

`colour.models`

`describe_video_signal_colour_primaries`(...)	Describe given video signal colour primaries code point.
`describe_video_signal_transfer_characteristics`(...)	Describe given video signal transfer characteristics code point.
`describe_video_signal_matrix_coefficients`(...)	Describe given video signal matrix coefficients code point.

colour.models.describe_video_signal_colour_primaries

`colour.models.`**`describe_video_signal_colour_primaries`**(*code_point: int*, *print_description: bool = True*, ***kwargs*) → str

Describe given video signal colour primaries code point.

Parameters

- **`code_point`** (`int`) – Video signal colour primaries code point to describe from `colour.COLOUR_PRIMARIES_ITUTH273` attribute.
- **`print_description`** (`bool`) – Whether to print the description.
- **`padding`** – {`colour.utilities.message_box()`}, Padding on each side of the message.
- **`print_callable`** – {`colour.utilities.message_box()`}, Callable used to print the message box.
- **`width`** – {`colour.utilities.message_box()`}, Message box width.

Returns Video signal colour primaries code point description.

Return type str

References

[FFmpegDevelopers22a], [InternationalOfStandardization13], [InternationalOfStandardization21], [InternationalOfStandardization20], [InternationalTUnion21]

Examples

```
>>> description = describe_video_signal_colour_primaries(1, width=75)
===========================================================================
*                                                                         *
*   Colour Primaries: 1                                                   *
*   -------------------                                                   *
*                                                                         *
*   Primaries        : [[ 0.64  0.33]                                     *
*                        [ 0.3   0.6 ]                                    *
*                        [ 0.15  0.06]]                                   *
*   Whitepoint       : [ 0.3127  0.329 ]                                  *
*   Whitepoint Name  : D65                                                *
*   NPM              : [[ 0.4123908   0.35758434  0.18048079]             *
*                        [ 0.21263901  0.71516868  0.07219232]            *
*                        [ 0.01933082  0.11919478  0.95053215]]           *
*   NPM -1           : [[ 3.24096994 -1.53738318 -0.49861076]             *
*                        [-0.96924364  1.8759675   0.04155506]            *
*                        [ 0.05563008 -0.20397696  1.05697151]]           *
*   FFmpeg Constants : ['AVCOL_PRI_BT709', 'BT709']                       *
*                                                                         *
===========================================================================
>>> description = describe_video_signal_colour_primaries(2, width=75)
===========================================================================
*                                                                         *
*   Colour Primaries: 2                                                   *
*   -------------------                                                   *
*                                                                         *
*   Primaries        : Unspecified                                        *
*   Whitepoint       : Unspecified                                        *
*   Whitepoint Name  : Unspecified                                        *
*   NPM              : Unspecified                                        *
*   NPM -1           : Unspecified                                        *
*   FFmpeg Constants : ['AVCOL_PRI_UNSPECIFIED', 'UNSPECIFIED']           *
*                                                                         *
===========================================================================
>>> description = describe_video_signal_colour_primaries(
...     FFmpegConstantsColourPrimaries_ITUTH273.JEDEC_P22, width=75
... )
===========================================================================
*                                                                         *
*   Colour Primaries: 22                                                  *
*   --------------------                                                  *
*                                                                         *
*   Primaries        : [[ 0.63   0.34 ]                                   *
*                        [ 0.295  0.605]                                  *
*                        [ 0.155  0.077]]                                 *
*   Whitepoint       : [ 0.3127  0.329 ]                                  *
*   Whitepoint Name  : D65                                                *
*   NPM              : [[ 0.42942013  0.3277917   0.1932441 ]             *
*                        [ 0.23175055  0.67225077  0.09599868]            *
*                        [ 0.02044858  0.11111583  0.95749334]]           *
*   NPM -1           : [[ 3.13288278 -1.44707454 -0.48720324]             *
*                        [-1.08850877  2.01538781  0.01762239]            *
*                        [ 0.05941301 -0.20297883  1.05275352]]           *
*   FFmpeg Constants : ['AVCOL_PRI_EBU3213', 'AVCOL_PRI_JEDEC_P22',       *
*   'EBU3213', 'JEDEC_P22']                                               *
```

(continues on next page)

(continued from previous page)

```
*                                                                         *
===========================================================================
```

colour.models.describe_video_signal_transfer_characteristics

colour.models.**describe_video_signal_transfer_characteristics**(*code_point: int*, *print_description: bool = True*, ***kwargs*) → str

Describe given video signal transfer characteristics code point.

Parameters

- **code_point** (int) – Video signal transfer characteristics code point to describe from colour.TRANSFER_CHARACTERISTICS_ITUTH273 attribute.
- **print_description** (bool) – Whether to print the description.
- **padding** – {colour.utilities.message_box()}, Padding on each side of the message.
- **print_callable** – {colour.utilities.message_box()}, Callable used to print the message box.
- **width** – {colour.utilities.message_box()}, Message box width.

Returns Video signal colour primaries code point description.

Return type str

References

[FFmpegDevelopers22c], [InternationalOfStandardization13], [InternationalOfStandardization21], [InternationalOfStandardization20], [InternationalTUnion21]

Examples

```
>>> description = describe_video_signal_transfer_characteristics(
...     1, width=75
... )
...
===========================================================================
*                                                                         *
*   Transfer Characteristics: 1                                           *
*   ---------------------------                                           *
*                                                                         *
*   Function          : <function oetf_BT709 at 0x...>...*
*   FFmpeg Constants : ['AVCOL_TRC_BT709', 'BT709']                       *
*                                                                         *
===========================================================================
>>> description = describe_video_signal_transfer_characteristics(
...     2, width=75
... )
...
===========================================================================
*                                                                         *
*   Transfer Characteristics: 2                                           *
*   ---------------------------                                           *
```

(continues on next page)

(continued from previous page)

```
*                                                                           *
*   Function          : <function _unspecified at 0x...>...*
*   FFmpeg Constants : ['AVCOL_TRC_UNSPECIFIED', 'UNSPECIFIED']            *
*                                                                           *
===========================================================================
>>> description = describe_video_signal_transfer_characteristics(
...     FFmpegConstantsTransferCharacteristics_ITUTH273.SMPTE428, width=75
... )
...
===========================================================================
*                                                                           *
*   Transfer Characteristics: 17                                            *
*   ----------------------------                                            *
*                                                                           *
*   Function          : <function eotf_inverse_H273_ST428_1 at              *
*   0x...>...*
*   FFmpeg Constants : ['AVCOL_TRC_SMPTE428', 'AVCOL_TRC_SMPTEST428_1',    *
*   'SMPTE428', 'SMPTEST428_1']                                             *
*                                                                           *
===========================================================================
```

colour.models.describe_video_signal_matrix_coefficients

colour.models.**describe_video_signal_matrix_coefficients**(*code_point: int*, *print_description: bool = True*, ***kwargs*) → str

Describe given video signal matrix coefficients code point.

Parameters

- **code_point** (int) – Video signal matrix coefficients code point to describe from colour.MATRIX_COEFFICIENTS_ITUTH273 attribute.
- **print_description** (bool) – Whether to print the description.
- **padding** – {colour.utilities.message_box()}, Padding on each side of the message.
- **print_callable** – {colour.utilities.message_box()}, Callable used to print the message box.
- **width** – {colour.utilities.message_box()}, Message box width.

Returns Video signal colour primaries code point description.

Return type str

References

[FFmpegDevelopers22b], [InternationalOfStandardization13], [InternationalOfStandardization21], [InternationalOfStandardization20], [InternationalTUnion21]

Examples

```
>>> description = describe_video_signal_matrix_coefficients(1, width=75)
===========================================================================
*                                                                         *
*   Matrix Coefficients: 1                                                *
*   ----------------------                                                *
*                                                                         *
*   Matrix Coefficients : [ 0.2126  0.0722]                               *
*   FFmpeg Constants    : ['AVCOL_SPC_BT709', 'BT709']                    *
*                                                                         *
===========================================================================
>>> description = describe_video_signal_matrix_coefficients(2, width=75)
===========================================================================
*                                                                         *
*   Matrix Coefficients: 2                                                *
*   ----------------------                                                *
*                                                                         *
*   Matrix Coefficients : Unspecified                                     *
*   FFmpeg Constants    : ['AVCOL_SPC_UNSPECIFIED', 'UNSPECIFIED']        *
*                                                                         *
===========================================================================
>>> description = describe_video_signal_matrix_coefficients(
...     FFmpegConstantsMatrixCoefficients_ITUTH273.ICTCP, width=75
... )
...
===========================================================================
*                                                                         *
*   Matrix Coefficients: 14                                               *
*   -----------------------                                               *
*                                                                         *
*   Matrix Coefficients : ICTCP                                           *
*   FFmpeg Constants    : ['AVCOL_SPC_ICTCP', 'ICTCP']                    *
*                                                                         *
===========================================================================
```

Pointer's Gamut

`colour`

CCS_ILLUMINANT_POINTER_GAMUT	ndarray(shape, dtype=float, buffer=None, offset=0,
DATA_POINTER_GAMUT_VOLUME	ndarray(shape, dtype=float, buffer=None, offset=0,
CCS_POINTER_GAMUT_BOUNDARY	ndarray(shape, dtype=float, buffer=None, offset=0,

colour.models.CCS_ILLUMINANT_POINTER_GAMUT

colour.models.**CCS_ILLUMINANT_POINTER_GAMUT = array([0.31005673, 0.3161457])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** (`(for the __new__ method; see Notes)`) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.
- **offset** (`int, optional`) – Offset of array data in buffer.
- **strides** (`tuple of ints, optional`) – Strides of data in memory.
- **order** (`{'C', 'F'}, optional`) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.models.DATA_POINTER_GAMUT_VOLUME

`colour.models.DATA_POINTER_GAMUT_VOLUME = array([[ 15, 10, 0], [ 15, 15, 10], [ 15, 14, 20], ..., [ 90, 9, 330], [ 90, 4, 340], [ 90, 6, 350]])`

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** (`(for the __new__ method; see Notes)`) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.
- **offset** (`int, optional`) – Offset of array data in buffer.
- **strides** (`tuple of ints, optional`) – Strides of data in memory.
- **order** (`{'C', 'F'}, optional`) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.models.CCS_POINTER_GAMUT_BOUNDARY

`colour.models.`**`CCS_POINTER_GAMUT_BOUNDARY = array([[0.659, 0.316], [0.634, 0.351], [0.594, 0.391], [0.557, 0.427], [0.523, 0.462], [0.482, 0.491], [0.444, 0.515], [0.409, 0.546], [0.371, 0.558], [0.332, 0.573], [0.288, 0.584], [0.242, 0.576], [0.202, 0.53], [0.177, 0.454], [0.151, 0.389], [0.151, 0.33], [0.162, 0.295], [0.157, 0.266], [0.159, 0.245], [0.142, 0.214], [0.141, 0.195], [0.129, 0.168], [0.138, 0.141], [0.145, 0.129], [0.145, 0.106], [0.161, 0.094], [0.188, 0.084], [0.252, 0.104], [0.324, 0.127], [0.393, 0.165], [0.451, 0.199], [0.508, 0.226]])`**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(. . .)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** (`(for the __new__ method; see Notes)`) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.

- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` An ndarray alias generic w.r.t. its *dtype.type <numpy.dtype.type>*.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

Colour Notation Systems

Munsell Renotation System

`colour`

`munsell_colour_to_xyY`(munsell_colour)	Convert given *Munsell* colour to *CIE xyY* colourspace.
`xyY_to_munsell_colour`(xyY[, hue_decimals, ...])	Convert from *CIE xyY* colourspace to *Munsell* colour.

colour.munsell_colour_to_xyY

`colour.`**`munsell_colour_to_xyY`**(*munsell_colour: ArrayLike*) → NDArrayFloat

Convert given *Munsell* colour to *CIE xyY* colourspace.

Parameters **`munsell_colour`** (`ArrayLike`) – *Munsell* colour.

Returns *CIE xyY* colourspace array.

Return type `numpy.NDArrayFloat`

Notes

Range	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[Cen], [Cen12]

Examples

```
>>> munsell_colour_to_xyY("4.2YR 8.1/5.3")
array([ 0.3873694...,  0.3575165...,  0.59362   ])
>>> munsell_colour_to_xyY("N8.9")
array([ 0.31006   ,  0.31616   ,  0.7461345...])
```

colour.xyY_to_munsell_colour

colour.**xyY_to_munsell_colour**(*xyY: ArrayLike*, *hue_decimals: int = 1*, *value_decimals: int = 1*, *chroma_decimals: int = 1*) → str | NDArrayStr

Convert from *CIE xyY* colourspace to *Munsell* colour.

Parameters

- **xyY** (ArrayLike) – *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 str or numpy.NDArrayFloat

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[Cen], [Cen12]

Examples

```
>>> xyY = np.array([0.38736945, 0.35751656, 0.59362000])
>>> xyY_to_munsell_colour(xyY)
'4.2YR 8.1/5.3'
```

Dataset

colour

MUNSELL_COLOURS	Define the *Munsell Renotation System* datasets.

colour.MUNSELL_COLOURS

colour.**MUNSELL_COLOURS = CanonicalMapping({'Munsell Colours All': ..., 'Munsell Colours 1929': ..., 'Munsell Colours Real': ..., 'all': ..., '1929': ..., 'real': ...})**

Define the *Munsell Renotation System* datasets.

- `Munsell Colours All`: *all* published *Munsell* colours, including the extrapolated colors.
- `Munsell Colours 1929`: the colours appearing in the 1929 *Munsell Book of Color*. These data has been used in the scaling experiments leading to the 1943 renotation.
- `Munsell Colours Real`: *real*, within MacAdam limits *Munsell* colours only. They are the colours listed in the original 1943 renotation article *(Newhall, Nickerson, & Judd, 1943)*.

Notes

- The Munsell Renotation data commonly available within the *all.dat*, *experimental.dat* and *real.dat* files features *CIE xyY* colourspace values that are scaled by a $1/0.975 \simeq 1.02568$ factor. If you are performing conversions using *Munsell Colorlab* specification, e.g. *2.5R 9/2*, according to *ASTM D1535-08e1* method, you should not scale the output Y Luminance. However, if you use directly the *CIE xyY* colourspace values from the Munsell Renotation data, you should scale the Y Luminance before conversions by a 0.975 factor.

  ASTM D1535-08e1 states that:

  ```
  The coefficients of this equation are obtained from the 1943 equation
  by multiplying each coefficient by 0.975, the reflectance factor of
  magnesium oxide with respect to the perfect reflecting diffuser, and
  rounding to ve digits of precision.
  ```

- Chromaticities assume *CIE Illuminant C*, approximately 6700K, as neutral origin for both the hue and chroma loci.

References

- [MunsellCScienceb] : Munsell Color Science. (n.d.). Munsell Colours Data. Retrieved August 20, 2014, from http://www.cis.rit.edu/research/mcsl2/online/munsell.php

Aliases:

- 'all': 'Munsell Colours All'
- '1929': 'Munsell Colours 1929'
- 'real': 'Munsell Colours Real'

Munsell Value

colour

munsell_value(Y[, method])	Return the *Munsell* value V of given *luminance* Y using given method.
MUNSELL_VALUE_METHODS	Supported *Munsell* value computation methods.

colour.munsell_value

colour.**munsell_value**(*Y: ArrayLike, method: Union[Literal['ASTM D1535', 'Ladd 1955', 'McCamy 1987', 'Moon 1943', 'Munsell 1933', 'Priest 1920', 'Saunderson 1944'], str] = 'ASTM D1535'*) → NDArrayFloat

Return the *Munsell* value V of given *luminance* Y using given method.

Parameters

- **Y** (ArrayLike) – *luminance* Y.
- **method** (Union[Literal['ASTM D1535', 'Ladd 1955', 'McCamy 1987', 'Moon 1943', 'Munsell 1933', 'Priest 1920', 'Saunderson 1944'], str]) – Computation method.

Returns *Munsell* value V.

Return type np.float or numpy.NDArrayFloat

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[ASTMInternational89], [Wikipedia07a]

Examples

```
>>> munsell_value(12.23634268)
4.0824437...
>>> munsell_value(12.23634268, method="Priest 1920")
3.4980484...
>>> munsell_value(12.23634268, method="Munsell 1933")
4.1627702...
>>> munsell_value(12.23634268, method="Moon 1943")
4.0688120...
>>> munsell_value(12.23634268, method="Saunderson 1944")
...
4.0444736...
>>> munsell_value(12.23634268, method="Ladd 1955")
```

(continues on next page)

(continued from previous page)

```
4.0511633...
>>> munsell_value(12.23634268, method="McCamy 1987")
4.0814348...
```

colour.MUNSELL_VALUE_METHODS

colour.**MUNSELL_VALUE_METHODS = CanonicalMapping({'Priest 1920': ..., 'Munsell 1933': ..., 'Moon 1943': ..., 'Saunderson 1944': ..., 'Ladd 1955': ..., 'McCamy 1987': ..., 'ASTM D1535': ..., 'astm2008': ...})**

Supported *Munsell* value computation methods.

References

[ASTMInternational89], [Wikipedia07a]

Aliases:

- 'astm2008': 'ASTM D1535'

Priest, Gibson and MacNicholas (1920)

colour.notation

munsell_value_Priest1920(Y)	Return the *Munsell* value V of given *luminance* Y using *Priest et al. (1920)* method.

colour.notation.munsell_value_Priest1920

colour.notation.**munsell_value_Priest1920**(*Y: ArrayLike*) → NDArrayFloat

Return the *Munsell* value V of given *luminance* Y using *Priest et al. (1920)* method.

Parameters Y (ArrayLike) – *luminance* Y.

Returns *Munsell* value V.

Return type np.float or numpy.NDArrayFloat

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wikipedia07a]

Examples

```
>>> munsell_value_Priest1920(12.23634268)
3.4980484...
```

Munsell, Sloan and Godlove (1933)

`colour.notation`

`munsell_value_Munsell1933`(Y)	Return the *Munsell* value V of given *luminance* Y using *Munsell et al. (1933)* method.

colour.notation.munsell_value_Munsell1933

`colour.notation.`**`munsell_value_Munsell1933`**(*Y: ArrayLike*) → NDArrayFloat

Return the *Munsell* value V of given *luminance* Y using *Munsell et al. (1933)* method.

Parameters **`Y`** (`ArrayLike`) – *luminance* Y.

Returns *Munsell* value V.

Return type `np.float` or `numpy.NDArrayFloat`

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wikipedia07a]

Examples

```
>>> munsell_value_Munsell1933(12.23634268)
4.1627702...
```

Moon and Spencer (1943)

`colour.notation`

munsell_value_Moon1943(Y)	Return 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: ArrayLike*) → NDArrayFloat

Return the *Munsell* value V of given *luminance* Y using *Moon and Spencer (1943)* method.

Parameters **Y** (ArrayLike) – *luminance* Y.

Returns *Munsell* value V.

Return type np.float or numpy.NDArrayFloat

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wikipedia07a]

Examples

```
>>> munsell_value_Moon1943(12.23634268)
4.0688120...
```

Saunderson and Milner (1944)

`colour.notation`

munsell_value_Saunderson1944(Y)	Return 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: ArrayLike*) → NDArrayFloat

Return the *Munsell* value V of given *luminance* Y using *Saunderson and Milner (1944)* method.

Parameters **Y** (ArrayLike) – *luminance* Y.

Returns *Munsell* value V.

Return type np.float or numpy.NDArrayFloat

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wikipedia07a]

Examples

```
>>> munsell_value_Saunderson1944(12.23634268)
4.0444736...
```

Ladd and Pinney (1955)

colour.notation

munsell_value_Ladd1955(Y)	Return 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: ArrayLike*) → NDArrayFloat

Return the *Munsell* value V of given *luminance* Y using *Ladd and Pinney (1955)* method.

Parameters **Y** (ArrayLike) – *luminance* Y.

Returns *Munsell* value V.

Return type np.float or numpy.NDArrayFloat

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wikipedia07a]

Examples

```
>>> munsell_value_Ladd1955(12.23634268)
4.0511633...
```

McCamy (1987)

`colour.notation`

`munsell_value_McCamy1987`(Y)	Return the *Munsell* value V of given *luminance* Y using *McCamy (1987)* method.

colour.notation.munsell_value_McCamy1987

`colour.notation.`**`munsell_value_McCamy1987`**(*Y: ArrayLike*) → NDArrayFloat

Return the *Munsell* value V of given *luminance* Y using *McCamy (1987)* method.

Parameters **Y** (`ArrayLike`) – *luminance* Y.

Returns *Munsell* value V.

Return type `np.float` or `numpy.NDArrayFloat`

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[ASTMInternational89]

Examples

```
>>> munsell_value_McCamy1987(12.23634268)
4.0814348...
```

ASTM D1535-08e1

colour.notation

munsell_value_ASTMD1535(Y)	Return the *Munsell* value V of given *luminance* Y using an inverse lookup table from *ASTM D1535-08e1* method.

colour.notation.munsell_value_ASTMD1535

colour.notation.**munsell_value_ASTMD1535**(*Y: ArrayLike*) → NDArrayFloat

Return the *Munsell* value V of given *luminance* Y using an inverse lookup table from *ASTM D1535-08e1* method.

Parameters **Y** (ArrayLike) – *luminance* Y

Returns *Munsell* value V.

Return type np.float or numpy.NDArrayFloat

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

- The *Munsell* value* computation with *ASTM D1535-08e1* method is only defined for domain [0, 100].

References

[ASTMInternational89]

Examples

```
>>> munsell_value_ASTMD1535(12.23634268)
4.0824437...
```

Hexadecimal Representation

`colour.notation`

RGB_to_HEX(RGB)	Convert from *RGB* colourspace to hexadecimal representation.
HEX_to_RGB(HEX)	Convert from hexadecimal representation to *RGB* colourspace.

colour.notation.RGB_to_HEX

colour.notation.**RGB_to_HEX**(*RGB: ArrayLike*) → NDArrayStr

Convert from *RGB* colourspace to hexadecimal representation.

Parameters **RGB** (ArrayLike) – *RGB* colourspace array.

Returns Hexadecimal representation.

Return type str or numpy.array

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> RGB = np.array([0.66666667, 0.86666667, 1.00000000])
>>> RGB_to_HEX(RGB)
'#aaddff'
```

colour.notation.HEX_to_RGB

colour.notation.**HEX_to_RGB**(*HEX: ArrayLike*) → NDArrayFloat

Convert from hexadecimal representation to *RGB* colourspace.

Parameters **HEX** (ArrayLike) – Hexadecimal representation.

Returns *RGB* colourspace array.

Return type numpy.array

Notes

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> HEX = "#aaddff"
>>> HEX_to_RGB(HEX)
array([ 0.6666666...,  0.8666666...,  1.        ])
```

Web Colours

`colour.notation`

`CSS_COLOR_3_BASIC`	The list of basic colour keywords.
`CSS_COLOR_3_EXTENDED`	The list of the X11 colors supported by popular browsers with the addition of gray/grey variants from SVG 1.0.
`CSS_COLOR_3`	List of colour keywords as given by as given by *CSS Color Module Level 3 W3C Recommendation*.
`keyword_to_RGB_CSSColor3`(keyword)	Convert given colour keyword to *RGB* colourspace according to *CSS Color Module Level 3 W3C Recommendation*.

colour.notation.CSS_COLOR_3_BASIC

`colour.notation.`**`CSS_COLOR_3_BASIC = CanonicalMapping({'black': ..., 'silver': ..., 'gray': ..., 'white': ..., 'maroon': ..., 'red': ..., 'purple': ..., 'fuchsia': ..., 'green': ..., 'lime': ..., 'olive': ..., 'yellow': ..., 'navy': ..., 'blue': ..., 'teal': ..., 'aqua': ...})`**

The list of basic colour keywords. The colour names are ASCII case-insensitive.

References

[W3C22]

colour.notation.CSS_COLOR_3_EXTENDED

```
colour.notation.CSS_COLOR_3_EXTENDED = CanonicalMapping({'aliceblue': ..., 'antiquewhite':
..., 'aqua': ..., 'aquamarine': ..., 'azure': ..., 'beige': ..., 'bisque': ..., 'black':
..., 'blanchedalmond': ..., 'blue': ..., 'blueviolet': ..., 'brown': ..., 'burlywood':
..., 'cadetblue': ..., 'chartreuse': ..., 'chocolate': ..., 'coral': ...,
'cornflowerblue': ..., 'cornsilk': ..., 'crimson': ..., 'cyan': ..., 'darkblue': ...,
'darkcyan': ..., 'darkgoldenrod': ..., 'darkgray': ..., 'darkgreen': ..., 'darkgrey': ...,
'darkkhaki': ..., 'darkmagenta': ..., 'darkolivegreen': ..., 'darkorange': ...,
'darkorchid': ..., 'darkred': ..., 'darksalmon': ..., 'darkseagreen': ...,
'darkslateblue': ..., 'darkslategray': ..., 'darkslategrey': ..., 'darkturquoise': ...,
'darkviolet': ..., 'deeppink': ..., 'deepskyblue': ..., 'dimgray': ..., 'dimgrey': ...,
'dodgerblue': ..., 'firebrick': ..., 'floralwhite': ..., 'forestgreen': ..., 'fuchsia':
..., 'gainsboro': ..., 'ghostwhite': ..., 'gold': ..., 'goldenrod': ..., 'gray': ...,
'green': ..., 'greenyellow': ..., 'grey': ..., 'honeydew': ..., 'hotpink': ...,
'indianred': ..., 'indigo': ..., 'ivory': ..., 'khaki': ..., 'lavender': ...,
'lavenderblush': ..., 'lawngreen': ..., 'lemonchiffon': ..., 'lightblue': ...,
'lightcoral': ..., 'lightcyan': ..., 'lightgoldenrodyellow': ..., 'lightgray': ...,
'lightgreen': ..., 'lightgrey': ..., 'lightpink': ..., 'lightsalmon': ...,
'lightseagreen': ..., 'lightskyblue': ..., 'lightslategray': ..., 'lightslategrey': ...,
'lightsteelblue': ..., 'lightyellow': ..., 'lime': ..., 'limegreen': ..., 'linen': ...,
'magenta': ..., 'maroon': ..., 'mediumaquamarine': ..., 'mediumblue': ..., 'mediumorchid':
..., 'mediumpurple': ..., 'mediumseagreen': ..., 'mediumslateblue': ...,
'mediumspringgreen': ..., 'mediumturquoise': ..., 'mediumvioletred': ..., 'midnightblue':
..., 'mintcream': ..., 'mistyrose': ..., 'moccasin': ..., 'navajowhite': ..., 'navy': ...,
'oldlace': ..., 'olive': ..., 'olivedrab': ..., 'orange': ..., 'orangered': ..., 'orchid':
..., 'palegoldenrod': ..., 'palegreen': ..., 'paleturquoise': ..., 'palevioletred': ...,
'papayawhip': ..., 'peachpuff': ..., 'peru': ..., 'pink': ..., 'plum': ..., 'powderblue':
..., 'purple': ..., 'red': ..., 'rosybrown': ..., 'royalblue': ..., 'saddlebrown': ...,
'salmon': ..., 'sandybrown': ..., 'seagreen': ..., 'seashell': ..., 'sienna': ...,
'silver': ..., 'skyblue': ..., 'slateblue': ..., 'slategray': ..., 'slategrey': ...,
'snow': ..., 'springgreen': ..., 'steelblue': ..., 'tan': ..., 'teal': ..., 'thistle':
..., 'tomato': ..., 'turquoise': ..., 'violet': ..., 'wheat': ..., 'white': ...,
'whitesmoke': ..., 'yellow': ..., 'yellowgreen': ...})
```

The list of the X11 colors supported by popular browsers with the addition of gray/grey variants from SVG 1.0. The resulting list is precisely the same as the SVG 1.0 color keyword names. This specification extends their definition beyond SVG.

References

[W3C22]

colour.notation.CSS_COLOR_3

colour.notation.**CSS_COLOR_3 = CanonicalMapping({'black': ..., 'silver': ..., 'gray': ..., 'white': ..., 'maroon': ..., 'red': ..., 'purple': ..., 'fuchsia': ..., 'green': ..., 'lime': ..., 'olive': ..., 'yellow': ..., 'navy': ..., 'blue': ..., 'teal': ..., 'aqua': ..., 'aliceblue': ..., 'antiquewhite': ..., 'aquamarine': ..., 'azure': ..., 'beige': ..., 'bisque': ..., 'blanchedalmond': ..., 'blueviolet': ..., 'brown': ..., 'burlywood': ..., 'cadetblue': ..., 'chartreuse': ..., 'chocolate': ..., 'coral': ..., 'cornflowerblue': ..., 'cornsilk': ..., 'crimson': ..., 'cyan': ..., 'darkblue': ..., 'darkcyan': ..., 'darkgoldenrod': ..., 'darkgray': ..., 'darkgreen': ..., 'darkgrey': ..., 'darkkhaki': ..., 'darkmagenta': ..., 'darkolivegreen': ..., 'darkorange': ..., 'darkorchid': ..., 'darkred': ..., 'darksalmon': ..., 'darkseagreen': ..., 'darkslateblue': ..., 'darkslategray': ..., 'darkslategrey': ..., 'darkturquoise': ..., 'darkviolet': ..., 'deeppink': ..., 'deepskyblue': ..., 'dimgray': ..., 'dimgrey': ..., 'dodgerblue': ..., 'firebrick': ..., 'floralwhite': ..., 'forestgreen': ..., 'gainsboro': ..., 'ghostwhite': ..., 'gold': ..., 'goldenrod': ..., 'greenyellow': ..., 'grey': ..., 'honeydew': ..., 'hotpink': ..., 'indianred': ..., 'indigo': ..., 'ivory': ..., 'khaki': ..., 'lavender': ..., 'lavenderblush': ..., 'lawngreen': ..., 'lemonchiffon': ..., 'lightblue': ..., 'lightcoral': ..., 'lightcyan': ..., 'lightgoldenrodyellow': ..., 'lightgray': ..., 'lightgreen': ..., 'lightgrey': ..., 'lightpink': ..., 'lightsalmon': ..., 'lightseagreen': ..., 'lightskyblue': ..., 'lightslategray': ..., 'lightslategrey': ..., 'lightsteelblue': ..., 'lightyellow': ..., 'limegreen': ..., 'linen': ..., 'magenta': ..., 'mediumaquamarine': ..., 'mediumblue': ..., 'mediumorchid': ..., 'mediumpurple': ..., 'mediumseagreen': ..., 'mediumslateblue': ..., 'mediumspringgreen': ..., 'mediumturquoise': ..., 'mediumvioletred': ..., 'midnightblue': ..., 'mintcream': ..., 'mistyrose': ..., 'moccasin': ..., 'navajowhite': ..., 'oldlace': ..., 'olivedrab': ..., 'orange': ..., 'orangered': ..., 'orchid': ..., 'palegoldenrod': ..., 'palegreen': ..., 'paleturquoise': ..., 'palevioletred': ..., 'papayawhip': ..., 'peachpuff': ..., 'peru': ..., 'pink': ..., 'plum': ..., 'powderblue': ..., 'rosybrown': ..., 'royalblue': ..., 'saddlebrown': ..., 'salmon': ..., 'sandybrown': ..., 'seagreen': ..., 'seashell': ..., 'sienna': ..., 'skyblue': ..., 'slateblue': ..., 'slategray': ..., 'slategrey': ..., 'snow': ..., 'springgreen': ..., 'steelblue': ..., 'tan': ..., 'thistle': ..., 'tomato': ..., 'turquoise': ..., 'violet': ..., 'wheat': ..., 'whitesmoke': ..., 'yellowgreen': ...})**

List of colour keywords as given by as given by *CSS Color Module Level 3 W3C Recommendation*.

References

[W3C22]

colour.notation.keyword_to_RGB_CSSColor3

colour.notation.**keyword_to_RGB_CSSColor3**(*keyword: str*) → NDArrayFloat

Convert given colour keyword to *RGB* colourspace according to *CSS Color Module Level 3 W3C Recommendation*.

Parameters **keyword** (str) – Colour keyword.

Returns *RGB* colourspace array.

Return type numpy.array

Notes

- All the RGB colors are specified in the *IEC 61966-2-1:1999 sRGB* colourspace.

Examples

```
>>> keyword_to_RGB_CSSColor3("black")
array([ 0.,  0.,  0.])
>>> keyword_to_RGB_CSSColor3("white")
array([ 1.,  1.,  1.])
>>> keyword_to_RGB_CSSColor3("aliceblue")
array([ 0.9411764...,  0.9725490...,  1.        ])
```

Optical Phenomena

Rayleigh Scattering

`colour`

`rayleigh_scattering`(wavelength[, ...])	Return the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).
`sd_rayleigh_scattering`([shape, ...])	Return the *Rayleigh* spectral distribution for given spectral shape.
`scattering_cross_section`(wavelength[, ...])	Return 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: ArrayLike, CO2_concentration: ArrayLike = CONSTANT_STANDARD_CO2_CONCENTRATION, temperature: ArrayLike = CONSTANT_STANDARD_AIR_TEMPERATURE, pressure: ArrayLike = CONSTANT_AVERAGE_PRESSURE_MEAN_SEA_LEVEL, latitude: ArrayLike = CONSTANT_DEFAULT_LATITUDE, altitude: ArrayLike = CONSTANT_DEFAULT_ALTITUDE, avogadro_constant: ArrayLike = CONSTANT_AVOGADRO, n_s_function: Callable = air_refraction_index_Bodhaine1999, F_air_function: Callable = F_air_Bodhaine1999*) → NDArrayFloat

Return the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).

Parameters

- **`wavelength`** (`ArrayLike`) – Wavelength λ in centimeters (cm).
- **`CO2_concentration`** (`ArrayLike`) – CO_2 concentration in parts per million (ppm).
- **`temperature`** (`ArrayLike`) – Air temperature $T[K]$ in kelvin degrees.
- **`pressure`** (`ArrayLike`) – Surface pressure P of the measurement site.
- **`latitude`** (`ArrayLike`) – Latitude of the site in degrees.
- **`altitude`** (`ArrayLike`) – Altitude of the site in meters.
- **`avogadro_constant`** (`ArrayLike`) – *Avogadro*'s number (molecules mol^{-1}).

- **n_s_function** (`Callable`) – Air refraction index n_s computation method.
- **F_air_function** (`Callable`) – $(6 + 3_p)/(6 - 7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns *Rayleigh* optical depth $T_r(\lambda)$.

Return type `numpy.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], [Wikipedia01d]

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8)
0.0936290...
```

colour.sd_rayleigh_scattering

`colour.sd_rayleigh_scattering`(*shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_DEFAULT, CO2_concentration: ArrayLike = CONSTANT_STANDARD_CO2_CONCENTRATION, temperature: ArrayLike = CONSTANT_STANDARD_AIR_TEMPERATURE, pressure: ArrayLike = CONSTANT_AVERAGE_PRESSURE_MEAN_SEA_LEVEL, latitude: ArrayLike = CONSTANT_DEFAULT_LATITUDE, altitude: ArrayLike = CONSTANT_DEFAULT_ALTITUDE, avogadro_constant: ArrayLike = CONSTANT_AVOGADRO, n_s_function: Callable = air_refraction_index_Bodhaine1999, F_air_function: Callable = F_air_Bodhaine1999*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return the *Rayleigh* spectral distribution for given spectral shape.

Parameters

- **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape used to create the *Rayleigh* scattering spectral distribution.
- **CO2_concentration** (`ArrayLike`) – CO_2 concentration in parts per million (ppm).
- **temperature** (`ArrayLike`) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (`ArrayLike`) – Surface pressure P of the measurement site.
- **latitude** (`ArrayLike`) – Latitude of the site in degrees.
- **altitude** (`ArrayLike`) – Altitude of the site in meters.
- **avogadro_constant** (`ArrayLike`) – *Avogadro*'s number (molecules mol^{-1}).
- **n_s_function** (`Callable`) – Air refraction index n_s computation method.
- **F_air_function** (`Callable`) – $(6 + 3_p)/(6 - 7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns *Rayleigh* optical depth spectral distribution.

Return type `colour.SpectralDistribution`

References

[BWDS99], [Wikipedia01d]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_rayleigh_scattering()
...
SpectralDistribution([[ 360.        ,    0.5602465...],
                      [ 361.        ,    0.5537481...],
                      [ 362.        ,    0.5473446...],
                      [ 363.        ,    0.5410345...],
                      [ 364.        ,    0.5348161...],
                      [ 365.        ,    0.5286877...],
                      [ 366.        ,    0.5226477...],
                      [ 367.        ,    0.5166948...],
                      [ 368.        ,    0.5108272...],
                      [ 369.        ,    0.5050436...],
                      [ 370.        ,    0.4993425...],
                      [ 371.        ,    0.4937224...],
                      [ 372.        ,    0.4881820...],
                      [ 373.        ,    0.4827199...],
                      [ 374.        ,    0.4773348...],
                      [ 375.        ,    0.4720253...],
                      [ 376.        ,    0.4667902...],
                      [ 377.        ,    0.4616282...],
                      [ 378.        ,    0.4565380...],
                      [ 379.        ,    0.4515186...],
                      [ 380.        ,    0.4465686...],
                      [ 381.        ,    0.4416869...],
                      [ 382.        ,    0.4368724...],
                      [ 383.        ,    0.4321240...],
                      [ 384.        ,    0.4274405...],
                      [ 385.        ,    0.4228209...],
                      [ 386.        ,    0.4182641...],
                      [ 387.        ,    0.4137692...],
                      [ 388.        ,    0.4093350...],
                      [ 389.        ,    0.4049607...],
                      [ 390.        ,    0.4006451...],
                      [ 391.        ,    0.3963874...],
                      [ 392.        ,    0.3921867...],
                      [ 393.        ,    0.3880419...],
                      [ 394.        ,    0.3839523...],
                      [ 395.        ,    0.3799169...],
                      [ 396.        ,    0.3759348...],
                      [ 397.        ,    0.3720053...],
                      [ 398.        ,    0.3681274...],
                      [ 399.        ,    0.3643003...],
                      [ 400.        ,    0.3605233...],
                      [ 401.        ,    0.3567956...],
                      [ 402.        ,    0.3531163...],
                      [ 403.        ,    0.3494847...],
                      [ 404.        ,    0.3459001...],
                      [ 405.        ,    0.3423617...],
```

(continues on next page)

(continued from previous page)

```
                     [ 406.        ,   0.3388689...],
                     [ 407.        ,   0.3354208...],
                     [ 408.        ,   0.3320169...],
                     [ 409.        ,   0.3286563...],
                     [ 410.        ,   0.3253386...],
                     [ 411.        ,   0.3220629...],
                     [ 412.        ,   0.3188287...],
                     [ 413.        ,   0.3156354...],
                     [ 414.        ,   0.3124822...],
                     [ 415.        ,   0.3093687...],
                     [ 416.        ,   0.3062941...],
                     [ 417.        ,   0.3032579...],
                     [ 418.        ,   0.3002596...],
                     [ 419.        ,   0.2972985...],
                     [ 420.        ,   0.2943741...],
                     [ 421.        ,   0.2914858...],
                     [ 422.        ,   0.2886332...],
                     [ 423.        ,   0.2858157...],
                     [ 424.        ,   0.2830327...],
                     [ 425.        ,   0.2802837...],
                     [ 426.        ,   0.2775683...],
                     [ 427.        ,   0.2748860...],
                     [ 428.        ,   0.2722362...],
                     [ 429.        ,   0.2696185...],
                     [ 430.        ,   0.2670324...],
                     [ 431.        ,   0.2644775...],
                     [ 432.        ,   0.2619533...],
                     [ 433.        ,   0.2594594...],
                     [ 434.        ,   0.2569952...],
                     [ 435.        ,   0.2545605...],
                     [ 436.        ,   0.2521548...],
                     [ 437.        ,   0.2497776...],
                     [ 438.        ,   0.2474285...],
                     [ 439.        ,   0.2451072...],
                     [ 440.        ,   0.2428133...],
                     [ 441.        ,   0.2405463...],
                     [ 442.        ,   0.2383059...],
                     [ 443.        ,   0.2360916...],
                     [ 444.        ,   0.2339033...],
                     [ 445.        ,   0.2317404...],
                     [ 446.        ,   0.2296026...],
                     [ 447.        ,   0.2274895...],
                     [ 448.        ,   0.2254009...],
                     [ 449.        ,   0.2233364...],
                     [ 450.        ,   0.2212956...],
                     [ 451.        ,   0.2192782...],
                     [ 452.        ,   0.2172839...],
                     [ 453.        ,   0.2153124...],
                     [ 454.        ,   0.2133633...],
                     [ 455.        ,   0.2114364...],
                     [ 456.        ,   0.2095313...],
                     [ 457.        ,   0.2076478...],
                     [ 458.        ,   0.2057855...],
                     [ 459.        ,   0.2039442...],
                     [ 460.        ,   0.2021235...],
                     [ 461.        ,   0.2003233...],
```

(continues on next page)

(continued from previous page)

```
                  [ 462.        ,    0.1985432...],
                  [ 463.        ,    0.1967829...],
                  [ 464.        ,    0.1950423...],
                  [ 465.        ,    0.1933209...],
                  [ 466.        ,    0.1916186...],
                  [ 467.        ,    0.1899351...],
                  [ 468.        ,    0.1882702...],
                  [ 469.        ,    0.1866236...],
                  [ 470.        ,    0.1849951...],
                  [ 471.        ,    0.1833844...],
                  [ 472.        ,    0.1817913...],
                  [ 473.        ,    0.1802156...],
                  [ 474.        ,    0.1786570...],
                  [ 475.        ,    0.1771153...],
                  [ 476.        ,    0.1755903...],
                  [ 477.        ,    0.1740818...],
                  [ 478.        ,    0.1725895...],
                  [ 479.        ,    0.1711133...],
                  [ 480.        ,    0.1696529...],
                  [ 481.        ,    0.1682082...],
                  [ 482.        ,    0.1667789...],
                  [ 483.        ,    0.1653648...],
                  [ 484.        ,    0.1639658...],
                  [ 485.        ,    0.1625816...],
                  [ 486.        ,    0.1612121...],
                  [ 487.        ,    0.1598570...],
                  [ 488.        ,    0.1585163...],
                  [ 489.        ,    0.1571896...],
                  [ 490.        ,    0.1558769...],
                  [ 491.        ,    0.1545779...],
                  [ 492.        ,    0.1532925...],
                  [ 493.        ,    0.1520205...],
                  [ 494.        ,    0.1507617...],
                  [ 495.        ,    0.1495160...],
                  [ 496.        ,    0.1482832...],
                  [ 497.        ,    0.1470632...],
                  [ 498.        ,    0.1458558...],
                  [ 499.        ,    0.1446607...],
                  [ 500.        ,    0.1434780...],
                  [ 501.        ,    0.1423074...],
                  [ 502.        ,    0.1411488...],
                  [ 503.        ,    0.140002 ...],
                  [ 504.        ,    0.1388668...],
                  [ 505.        ,    0.1377433...],
                  [ 506.        ,    0.1366311...],
                  [ 507.        ,    0.1355301...],
                  [ 508.        ,    0.1344403...],
                  [ 509.        ,    0.1333615...],
                  [ 510.        ,    0.1322936...],
                  [ 511.        ,    0.1312363...],
                  [ 512.        ,    0.1301897...],
                  [ 513.        ,    0.1291535...],
                  [ 514.        ,    0.1281277...],
                  [ 515.        ,    0.1271121...],
                  [ 516.        ,    0.1261065...],
                  [ 517.        ,    0.1251110...],
```

(continues on next page)

(continued from previous page)

```
                    [ 518.        ,    0.1241253...],
                    [ 519.        ,    0.1231493...],
                    [ 520.        ,    0.1221829...],
                    [ 521.        ,    0.1212261...],
                    [ 522.        ,    0.1202786...],
                    [ 523.        ,    0.1193405...],
                    [ 524.        ,    0.1184115...],
                    [ 525.        ,    0.1174915...],
                    [ 526.        ,    0.1165806...],
                    [ 527.        ,    0.1156784...],
                    [ 528.        ,    0.1147851...],
                    [ 529.        ,    0.1139004...],
                    [ 530.        ,    0.1130242...],
                    [ 531.        ,    0.1121564...],
                    [ 532.        ,    0.1112971...],
                    [ 533.        ,    0.1104459...],
                    [ 534.        ,    0.1096030...],
                    [ 535.        ,    0.1087681...],
                    [ 536.        ,    0.1079411...],
                    [ 537.        ,    0.1071221...],
                    [ 538.        ,    0.1063108...],
                    [ 539.        ,    0.1055072...],
                    [ 540.        ,    0.1047113...],
                    [ 541.        ,    0.1039229...],
                    [ 542.        ,    0.1031419...],
                    [ 543.        ,    0.1023682...],
                    [ 544.        ,    0.1016019...],
                    [ 545.        ,    0.1008427...],
                    [ 546.        ,    0.1000906...],
                    [ 547.        ,    0.0993456...],
                    [ 548.        ,    0.0986075...],
                    [ 549.        ,    0.0978763...],
                    [ 550.        ,    0.0971519...],
                    [ 551.        ,    0.0964342...],
                    [ 552.        ,    0.0957231...],
                    [ 553.        ,    0.0950186...],
                    [ 554.        ,    0.0943206...],
                    [ 555.        ,    0.0936290...],
                    [ 556.        ,    0.0929438...],
                    [ 557.        ,    0.0922649...],
                    [ 558.        ,    0.0915922...],
                    [ 559.        ,    0.0909256...],
                    [ 560.        ,    0.0902651...],
                    [ 561.        ,    0.0896106...],
                    [ 562.        ,    0.0889620...],
                    [ 563.        ,    0.0883194...],
                    [ 564.        ,    0.0876825...],
                    [ 565.        ,    0.0870514...],
                    [ 566.        ,    0.0864260...],
                    [ 567.        ,    0.0858063...],
                    [ 568.        ,    0.0851921...],
                    [ 569.        ,    0.0845834...],
                    [ 570.        ,    0.0839801...],
                    [ 571.        ,    0.0833822...],
                    [ 572.        ,    0.0827897...],
                    [ 573.        ,    0.0822025...],
```

(continues on next page)

(continued from previous page)

```
                                [ 574.        ,    0.0816204...],
                                [ 575.        ,    0.0810436...],
                                [ 576.        ,    0.0804718...],
                                [ 577.        ,    0.0799051...],
                                [ 578.        ,    0.0793434...],
                                [ 579.        ,    0.0787866...],
                                [ 580.        ,    0.0782347...],
                                [ 581.        ,    0.0776877...],
                                [ 582.        ,    0.0771454...],
                                [ 583.        ,    0.0766079...],
                                [ 584.        ,    0.0760751...],
                                [ 585.        ,    0.0755469...],
                                [ 586.        ,    0.0750234...],
                                [ 587.        ,    0.0745043...],
                                [ 588.        ,    0.0739898...],
                                [ 589.        ,    0.0734797...],
                                [ 590.        ,    0.0729740...],
                                [ 591.        ,    0.0724727...],
                                [ 592.        ,    0.0719757...],
                                [ 593.        ,    0.0714830...],
                                [ 594.        ,    0.0709944...],
                                [ 595.        ,    0.0705101...],
                                [ 596.        ,    0.0700299...],
                                [ 597.        ,    0.0695538...],
                                [ 598.        ,    0.0690818...],
                                [ 599.        ,    0.0686137...],
                                [ 600.        ,    0.0681497...],
                                [ 601.        ,    0.0676895...],
                                [ 602.        ,    0.0672333...],
                                [ 603.        ,    0.0667809...],
                                [ 604.        ,    0.0663323...],
                                [ 605.        ,    0.0658875...],
                                [ 606.        ,    0.0654464...],
                                [ 607.        ,    0.0650091...],
                                [ 608.        ,    0.0645753...],
                                [ 609.        ,    0.0641453...],
                                [ 610.        ,    0.0637187...],
                                [ 611.        ,    0.0632958...],
                                [ 612.        ,    0.0628764...],
                                [ 613.        ,    0.0624604...],
                                [ 614.        ,    0.0620479...],
                                [ 615.        ,    0.0616388...],
                                [ 616.        ,    0.0612331...],
                                [ 617.        ,    0.0608307...],
                                [ 618.        ,    0.0604316...],
                                [ 619.        ,    0.0600358...],
                                [ 620.        ,    0.0596433...],
                                [ 621.        ,    0.0592539...],
                                [ 622.        ,    0.0588678...],
                                [ 623.        ,    0.0584848...],
                                [ 624.        ,    0.0581049...],
                                [ 625.        ,    0.0577281...],
                                [ 626.        ,    0.0573544...],
                                [ 627.        ,    0.0569837...],
                                [ 628.        ,    0.0566160...],
                                [ 629.        ,    0.0562513...],
```

(continues on next page)

(continued from previous page)

```
                     [ 630.        ,    0.0558895...],
                     [ 631.        ,    0.0555306...],
                     [ 632.        ,    0.0551746...],
                     [ 633.        ,    0.0548215...],
                     [ 634.        ,    0.0544712...],
                     [ 635.        ,    0.0541237...],
                     [ 636.        ,    0.0537789...],
                     [ 637.        ,    0.0534369...],
                     [ 638.        ,    0.0530977...],
                     [ 639.        ,    0.0527611...],
                     [ 640.        ,    0.0524272...],
                     [ 641.        ,    0.0520960...],
                     [ 642.        ,    0.0517674...],
                     [ 643.        ,    0.0514413...],
                     [ 644.        ,    0.0511179...],
                     [ 645.        ,    0.0507970...],
                     [ 646.        ,    0.0504786...],
                     [ 647.        ,    0.0501627...],
                     [ 648.        ,    0.0498493...],
                     [ 649.        ,    0.0495383...],
                     [ 650.        ,    0.0492298...],
                     [ 651.        ,    0.0489236...],
                     [ 652.        ,    0.0486199...],
                     [ 653.        ,    0.0483185...],
                     [ 654.        ,    0.0480194...],
                     [ 655.        ,    0.0477227...],
                     [ 656.        ,    0.0474283...],
                     [ 657.        ,    0.0471361...],
                     [ 658.        ,    0.0468462...],
                     [ 659.        ,    0.0465585...],
                     [ 660.        ,    0.0462730...],
                     [ 661.        ,    0.0459898...],
                     [ 662.        ,    0.0457087...],
                     [ 663.        ,    0.0454297...],
                     [ 664.        ,    0.0451529...],
                     [ 665.        ,    0.0448782...],
                     [ 666.        ,    0.0446055...],
                     [ 667.        ,    0.0443350...],
                     [ 668.        ,    0.0440665...],
                     [ 669.        ,    0.0438000...],
                     [ 670.        ,    0.0435356...],
                     [ 671.        ,    0.0432731...],
                     [ 672.        ,    0.0430127...],
                     [ 673.        ,    0.0427542...],
                     [ 674.        ,    0.0424976...],
                     [ 675.        ,    0.0422430...],
                     [ 676.        ,    0.0419902...],
                     [ 677.        ,    0.0417394...],
                     [ 678.        ,    0.0414905...],
                     [ 679.        ,    0.0412434...],
                     [ 680.        ,    0.0409981...],
                     [ 681.        ,    0.0407547...],
                     [ 682.        ,    0.0405131...],
                     [ 683.        ,    0.0402732...],
                     [ 684.        ,    0.0400352...],
                     [ 685.        ,    0.0397989...],
```

(continues on next page)

(continued from previous page)

```
                [ 686.        ,   0.0395643...],
                [ 687.        ,   0.0393315...],
                [ 688.        ,   0.0391004...],
                [ 689.        ,   0.0388710...],
                [ 690.        ,   0.0386433...],
                [ 691.        ,   0.0384173...],
                [ 692.        ,   0.0381929...],
                [ 693.        ,   0.0379701...],
                [ 694.        ,   0.0377490...],
                [ 695.        ,   0.0375295...],
                [ 696.        ,   0.0373115...],
                [ 697.        ,   0.0370952...],
                [ 698.        ,   0.0368804...],
                [ 699.        ,   0.0366672...],
                [ 700.        ,   0.0364556...],
                [ 701.        ,   0.0362454...],
                [ 702.        ,   0.0360368...],
                [ 703.        ,   0.0358297...],
                [ 704.        ,   0.0356241...],
                [ 705.        ,   0.0354199...],
                [ 706.        ,   0.0352172...],
                [ 707.        ,   0.0350160...],
                [ 708.        ,   0.0348162...],
                [ 709.        ,   0.0346178...],
                [ 710.        ,   0.0344208...],
                [ 711.        ,   0.0342253...],
                [ 712.        ,   0.0340311...],
                [ 713.        ,   0.0338383...],
                [ 714.        ,   0.0336469...],
                [ 715.        ,   0.0334569...],
                [ 716.        ,   0.0332681...],
                [ 717.        ,   0.0330807...],
                [ 718.        ,   0.0328947...],
                [ 719.        ,   0.0327099...],
                [ 720.        ,   0.0325264...],
                [ 721.        ,   0.0323443...],
                [ 722.        ,   0.0321634...],
                [ 723.        ,   0.0319837...],
                [ 724.        ,   0.0318054...],
                [ 725.        ,   0.0316282...],
                [ 726.        ,   0.0314523...],
                [ 727.        ,   0.0312777...],
                [ 728.        ,   0.0311042...],
                [ 729.        ,   0.0309319...],
                [ 730.        ,   0.0307609...],
                [ 731.        ,   0.0305910...],
                [ 732.        ,   0.0304223...],
                [ 733.        ,   0.0302548...],
                [ 734.        ,   0.0300884...],
                [ 735.        ,   0.0299231...],
                [ 736.        ,   0.0297590...],
                [ 737.        ,   0.0295960...],
                [ 738.        ,   0.0294342...],
                [ 739.        ,   0.0292734...],
                [ 740.        ,   0.0291138...],
                [ 741.        ,   0.0289552...],
```

(continues on next page)

(continued from previous page)

```
           [ 742.        ,    0.0287977...],
           [ 743.        ,    0.0286413...],
           [ 744.        ,    0.0284859...],
           [ 745.        ,    0.0283316...],
           [ 746.        ,    0.0281784...],
           [ 747.        ,    0.0280262...],
           [ 748.        ,    0.0278750...],
           [ 749.        ,    0.0277248...],
           [ 750.        ,    0.0275757...],
           [ 751.        ,    0.0274275...],
           [ 752.        ,    0.0272804...],
           [ 753.        ,    0.0271342...],
           [ 754.        ,    0.0269890...],
           [ 755.        ,    0.0268448...],
           [ 756.        ,    0.0267015...],
           [ 757.        ,    0.0265592...],
           [ 758.        ,    0.0264179...],
           [ 759.        ,    0.0262775...],
           [ 760.        ,    0.0261380...],
           [ 761.        ,    0.0259995...],
           [ 762.        ,    0.0258618...],
           [ 763.        ,    0.0257251...],
           [ 764.        ,    0.0255893...],
           [ 765.        ,    0.0254544...],
           [ 766.        ,    0.0253204...],
           [ 767.        ,    0.0251872...],
           [ 768.        ,    0.0250550...],
           [ 769.        ,    0.0249236...],
           [ 770.        ,    0.0247930...],
           [ 771.        ,    0.0246633...],
           [ 772.        ,    0.0245345...],
           [ 773.        ,    0.0244065...],
           [ 774.        ,    0.0242794...],
           [ 775.        ,    0.0241530...],
           [ 776.        ,    0.0240275...],
           [ 777.        ,    0.0239029...],
           [ 778.        ,    0.0237790...],
           [ 779.        ,    0.0236559...],
           [ 780.        ,    0.0235336...]],
          SpragueInterpolator,
          {},
          Extrapolator,
          {'method': 'Constant', 'left': None, 'right': None})
```

colour.scattering_cross_section

colour.**scattering_cross_section**(*wavelength: ArrayLike, CO2_concentration: ArrayLike = CONSTANT_STANDARD_CO2_CONCENTRATION, temperature: ArrayLike = CONSTANT_STANDARD_AIR_TEMPERATURE, avogadro_constant: ArrayLike = CONSTANT_AVOGADRO, n_s_function: Callable = air_refraction_index_Bodhaine1999, F_air_function: Callable = F_air_Bodhaine1999*) → NDArrayFloat

Return 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** (ArrayLike) – Wavelength λ in centimeters (cm).
- **CO2_concentration** (ArrayLike) – CO_2 concentration in parts per million (ppm).
- **temperature** (ArrayLike) – Air temperature $T[K]$ in kelvin degrees.
- **avogadro_constant** (ArrayLike) – *Avogadro*'s number (molecules mol^{-1}).
- **n_s_function** (Callable) – Air refraction index n_s computation method.
- **F_air_function** (Callable) – $(6 + 3_p)/(6 - 7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns Scattering cross-section per molecule σ of dry air.

Return type numpy.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], [Wikipedia01d]

Examples

```
>>> scattering_cross_section(555 * 10e-8)
4.3466692...e-27
```

colour.phenomena

rayleigh_optical_depth(wavelength[, ...])	Return 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: ArrayLike, CO2_concentration: ArrayLike = CONSTANT_STANDARD_CO2_CONCENTRATION, temperature: ArrayLike = CONSTANT_STANDARD_AIR_TEMPERATURE, pressure: ArrayLike = CONSTANT_AVERAGE_PRESSURE_MEAN_SEA_LEVEL, latitude: ArrayLike = CONSTANT_DEFAULT_LATITUDE, altitude: ArrayLike = CONSTANT_DEFAULT_ALTITUDE, avogadro_constant: ArrayLike = CONSTANT_AVOGADRO, n_s_function: Callable = air_refraction_index_Bodhaine1999, F_air_function: Callable = F_air_Bodhaine1999*) → NDArrayFloat

Return the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).

Parameters

- **wavelength** (ArrayLike) – Wavelength λ in centimeters (cm).
- **CO2_concentration** (ArrayLike) – CO_2 concentration in parts per million (ppm).
- **temperature** (ArrayLike) – Air temperature $T[K]$ in kelvin degrees.

- **pressure** (ArrayLike) – Surface pressure P of the measurement site.
- **latitude** (ArrayLike) – Latitude of the site in degrees.
- **altitude** (ArrayLike) – Altitude of the site in meters.
- **avogadro_constant** (ArrayLike) – *Avogadro*'s number (molecules mol^{-1}).
- **n_s_function** (Callable) – Air refraction index n_s computation method.
- **F_air_function** (Callable) – $(6+3_p)/(6-7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns *Rayleigh* optical depth $T_r(\lambda)$.

Return type numpy.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], [Wikipedia01d]

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8)
0.0936290...
```

Plotting

Common

colour.plotting

ColourSwatch(RGB, name)	Define a data structure for a colour swatch.

colour.plotting.ColourSwatch

class colour.plotting.**ColourSwatch**(*RGB: ArrayLike, name: str | None = <factory>*)

Bases: object

Define a data structure for a colour swatch.

Parameters

- **RGB** (ArrayLike) – RGB Colour.
- **name** (str | None) – Colour name.

Return type None

__eq__(*other*)

Return self==value.

__hash__ = None

__init__(*RGB: ArrayLike*, *name: str | None = <factory>*) → None

Parameters

- **RGB** (ArrayLike) –
- **name** (str | None) –

Return type None

__repr__()

Return repr(self).

__weakref__

list of weak references to the object (if defined)

z colour_style colour_cycle artist camera render label_rectangles uniform_axes3d plot_single_colour_swatch plot_multi_colour_swatches plot_single_function plot_multi_functions plot_image

Ancillary Objects

`colour.plotting.common`

KwargsArtist	Define the keyword argument types for the `colour.plotting.artist()` definition.
KwargsCamera	Define the keyword argument types for the `colour.plotting.camera()` definition.
KwargsRender	Define the keyword argument types for the `colour.plotting.render()` definition.
filter_passthrough(mapping, filterers[, ...])	Return mapping objects matching given filterers while passing through class instances whose type is one of the mapping element types.
filter_RGB_colourspaces(filterers[, ...])	Return the *RGB* colourspaces matching given filterers.
filter_cmfs(filterers[, allow_non_siblings])	Return the colour matching functions matching given filterers.
filter_illuminants(filterers[, ...])	Return the illuminants matching given filterers.
filter_colour_checkers(filterers[, ...])	Return the colour checkers matching given filterers.

colour.plotting.common.KwargsArtist

class `colour.plotting.common.`**KwargsArtist**

Define the keyword argument types for the `colour.plotting.artist()` definition.

Parameters

- **axes** (matplotlib.axes._axes.Axes) – Axes that will be passed through without creating a new figure.
- **uniform** (bool) – Whether to create the figure with an equal aspect ratio.

__init__(**args*, ***kwargs*)

Methods

`__init__`(*args, **kwargs)	
`clear`()	
`copy`()	
`fromkeys`([value])	Create a new dictionary with keys from iterable and values set to value.
`get`(key[, default])	Return the value for key if key is in the dictionary, else default.
`items`()	
`keys`()	
`pop`(k[,d])	If the key is not found, return the default if given; otherwise, raise a KeyError.
`popitem`()	Remove and return a (key, value) pair as a 2-tuple.
`setdefault`(key[, default])	Insert key with a value of default if key is not in the dictionary.
`update`([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
`values`()	

Attributes

`axes`
`uniform`

colour.plotting.common.KwargsCamera

class `colour.plotting.common.`**`KwargsCamera`**

Define the keyword argument types for the `colour.plotting.camera()` definition.

Parameters

- **figure** (`matplotlib.figure.Figure`) – Figure to apply the render elements onto.
- **axes** (`matplotlib.axes._axes.Axes`) – Axes to apply the render elements onto.
- **azimuth** (`float | None`) – Camera azimuth.
- **elevation** (`float | None`) – Camera elevation.
- **camera_aspect** (`Union[Literal['equal'], str]`) – Matplotlib axes aspect. Default is *equal*.

`__init__`(**args*, ***kwargs*)

Methods

`__init__`(*args, **kwargs)	
`clear`()	
`copy`()	
`fromkeys`([value])	Create a new dictionary with keys from iterable and values set to value.
`get`(key[, default])	Return the value for key if key is in the dictionary, else default.
`items`()	
`keys`()	
`pop`(k[,d])	If the key is not found, return the default if given; otherwise, raise a KeyError.
`popitem`()	Remove and return a (key, value) pair as a 2-tuple.
`setdefault`(key[, default])	Insert key with a value of default if key is not in the dictionary.
`update`([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
`values`()	

Attributes

`figure`
`axes`
`azimuth`
`elevation`
`camera_aspect`

colour.plotting.common.KwargsRender

class `colour.plotting.common.`**`KwargsRender`**

Define the keyword argument types for the `colour.plotting.render()` definition.

Parameters

- **`figure`** (`matplotlib.figure.Figure`) – Figure to apply the render elements onto.
- **`axes`** (`matplotlib.axes._axes.Axes`) – Axes to apply the render elements onto.
- **`filename`** (`str`) – Figure will be saved using given `filename` argument.

- **show** (bool) – Whether to show the figure and call matplotlib.pyplot.show() definition.
- **aspect** (Union[Literal['auto', 'equal'], float]) – Matplotlib axes aspect.
- **axes_visible** (bool) – Whether the axes are visible. Default is *True*.
- **bounding_box** (ArrayLike) – Array defining current axes limits such *bounding_box = (x min, x max, y min, y max)*.
- **tight_layout** (bool) – Whether to invoke the matplotlib.pyplot.tight_layout() definition.
- **legend** (bool) – Whether to display the legend. Default is *False*.
- **legend_columns** (int) – Number of columns in the legend. Default is *1*.
- **transparent_background** (bool) – Whether to turn off the background patch. Default is *True*.
- **title** (str) – Figure title.
- **wrap_title** (bool) – Whether to wrap the figure title. Default is *True*.
- **x_label** (str) – *X* axis label.
- **y_label** (str) – *Y* axis label.
- **x_ticker** (bool) – Whether to display the *X* axis ticker. Default is *True*.
- **y_ticker** (bool) – Whether to display the *Y* axis ticker. Default is *True*.

__init__(**args, **kwargs*)

Methods

__init__(*args, **kwargs)	
clear()	
copy()	
fromkeys([value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If the key is not found, return the default if given; otherwise, raise a KeyError.
popitem()	Remove and return a (key, value) pair as a 2-tuple.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

Attributes

figure
axes
filename
show
aspect
axes_visible
bounding_box
tight_layout
legend
legend_columns
transparent_background
title
wrap_title
x_label
y_label
x_ticker
y_ticker

colour.plotting.common.filter_passthrough

colour.plotting.common.**filter_passthrough**(*mapping: collections.abc.Mapping, filterers: Any | str | collections.abc.Sequence[Any | str], allow_non_siblings: bool = True*) → dict

Return mapping objects matching given filterers while passing through class instances whose type is one of the mapping element types.

This definition allows passing custom but compatible objects to the various plotting definitions that by default expect the key from a dataset element.

For example, a typical call to `colour.plotting.plot_multi_illuminant_sds()` definition is as follows:

```
>>> import colour
>>> colour.plotting.plot_multi_illuminant_sds(["A"])
...
```

With the previous example, it is also possible to pass a custom spectral distribution as follows:

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> colour.plotting.plot_multi_illuminant_sds(
...     ["A", colour.SpectralDistribution(data)]
... )
...
```

Similarly, a typical call to `colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1931()` definition is as follows:

```
>>> colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1931(
...     ["A"]
... )
...
```

But it is also possible to pass a custom whitepoint as follows:

```
>>> colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1931(
...     ["A", {"Custom": np.array([1 / 3 + 0.05, 1 / 3 + 0.05])}]
... )
...
```

Parameters

- **mapping** (`collections.abc.Mapping`) – Mapping to filter.
- **filterers** (`Any | str | collections.abc.Sequence[Any | str]`) – Filterer or object class instance (which is passed through directly if its type is one of the mapping element types) or list of filterers.
- **allow_non_siblings** (`bool`) – Whether to allow non-siblings to be also passed through.

Returns Filtered mapping.

Return type `dict`

Notes

- If the mapping passed is a `colour.utilities.CanonicalMapping` class instance, then the lower, slugified and canonical keys are also used for matching.

colour.plotting.common.filter_RGB_colourspaces

colour.plotting.common.**filter_RGB_colourspaces**(*filterers: colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colou | str], allow_non_siblings: bool = True*) → Dict[str, *colour.models.rgb.rgb_colourspace.RGB_Colourspace*]

Return the *RGB* colourspaces matching given filterers.

Parameters

- **filterers** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]`) – Filterer or `colour.RGB_Colourspace` class instance (which is passed through directly if its type is one of the mapping element types) or list of filterers. `filterers` elements can also be of any form supported by the `colour.plotting.common.filter_passthrough()` definition.
- **allow_non_siblings** (`bool`) – Whether to allow non-siblings to be also passed through.

Returns Filtered *RGB* colourspaces.

Return type `dict`

colour.plotting.common.filter_cmfs

colour.plotting.common.**filter_cmfs**(*filterers: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str], allow_non_siblings: bool = True*) → Dict[str, *colour.colorimetry.spectrum.MultiSpectralDistributions*]

Return the colour matching functions matching given filterers.

Parameters

- **filterers** (`colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]`) – Filterer or `colour.LMS_ConeFundamentals`, `colour.RGB_ColourMatchingFunctions` or `colour.XYZ_ColourMatchingFunctions` class instance (which is passed through directly if its type is one of the mapping element types) or list of filterers. `filterers` elements can also be of any form supported by the `colour.plotting.common.filter_passthrough()` definition.
- **allow_non_siblings** (`bool`) – Whether to allow non-siblings to be also passed through.

Returns Filtered colour matching functions.

Return type `dict`

colour.plotting.common.filter_illuminants

colour.plotting.common.**filter_illuminants**(*filterers: colour.colorimetry.spectrum.SpectralDistribution | str | collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | str], allow_non_siblings: bool = True*) → Dict[str, *colour.colorimetry.spectrum.SpectralDistribution*]

Return the illuminants matching given filterers.

Parameters

- **filterers** (colour.colorimetry.spectrum.SpectralDistribution | str | collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | str]) – Filterer or colour.SpectralDistribution class instance (which is passed through directly if its type is one of the mapping element types) or list of filterers. filterers elements can also be of any form supported by the colour.plotting.common.filter_passthrough() definition.
- **allow_non_siblings** (bool) – Whether to allow non-siblings to be also passed through.

Returns Filtered illuminants.

Return type dict

colour.plotting.common.filter_colour_checkers

colour.plotting.common.**filter_colour_checkers**(*filterers: colour.characterisation.datasets.colour_checkers.chromaticity_coord | str | collections.abc.Sequence[colour.characterisation.datasets.colour_checkers | str], allow_non_siblings: bool = True*) → Dict[str, *colour.characterisation.datasets.colour_checkers.chromaticity_coordin*]

Return the colour checkers matching given filterers.

Parameters

- **filterers** (colour.characterisation.datasets.colour_checkers.chromaticity_coordinates.ColourChecker | str | collections.abc.Sequence[colour.characterisation.datasets.colour_checkers.chromaticity_coordinates.ColourChecker | str]) – Filterer or colour.characterisation.ColourChecker class instance (which is passed through directly if its type is one of the mapping element types) or list of filterers. filterers elements can also be of any form supported by the colour.plotting.common.filter_passthrough() definition.
- **allow_non_siblings** (bool) – Whether to allow non-siblings to be also passed through.

Returns Filtered colour checkers.

Return type dict

Colorimetry

colour.plotting

plot_single_sd(sd[, cmfs, ...])	Plot given spectral distribution.
plot_multi_sds(sds[, plot_kwargs])	Plot given spectral distributions.
plot_single_cmfs([cmfs])	Plot given colour matching functions.
plot_multi_cmfs(cmfs, **kwargs)	Plot given colour matching functions.
plot_single_illuminant_sd(illuminant[, cmfs])	Plot given single illuminant spectral distribution.
plot_multi_illuminant_sds(illuminants, **kwargs)	Plot given illuminants spectral distributions.
plot_visible_spectrum([cmfs, ...])	Plot the visible colours spectrum using given standard observer *CIE XYZ* colour matching functions.
plot_single_lightness_function(function, ...)	Plot given *Lightness* function.
plot_multi_lightness_functions(functions, ...)	Plot given *Lightness* functions.
plot_single_luminance_function(function, ...)	Plot given *Luminance* function.
plot_multi_luminance_functions(functions, ...)	Plot given *Luminance* functions.
plot_blackbody_spectral_radiance([...])	Plot given blackbody spectral radiance.
plot_blackbody_colours([shape, cmfs])	Plot blackbody colours.

colour.plotting.plot_single_sd

colour.plotting.**plot_single_sd**(*sd:* colour.colorimetry.spectrum.SpectralDistribution, *cmfs:* colour.colorimetry.spectrum.MultiSpectralDistributions | *str* | collections.abc.Sequence*[*colour.colorimetry.spectrum.MultiSpectralDistributions | *str] = 'CIE 1931 2 Degree Standard Observer'*, *out_of_gamut_clipping:* bool *= True*, *modulate_colours_with_sd_amplitude:* bool *= False*, *equalize_sd_amplitude:* bool *= False*, ***kwargs:* Any) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given spectral distribution.

Parameters

- **sd** (colour.colorimetry.spectrum.SpectralDistribution) – Spectral distribution to plot.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **out_of_gamut_clipping** (bool) – Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.
- **modulate_colours_with_sd_amplitude** (bool) – Whether to modulate the colours with the spectral distribution amplitude.
- **equalize_sd_amplitude** (bool) – Whether to equalize the spectral distribution amplitude. Equalization occurs after the colours modulation thus setting both arguments to *True* will generate a spectrum strip where each wavelength colour is modulated by the spectral distribution amplitude. The usual 5% margin above the spectral distribution is also omitted.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

References

[Spi15]

Examples

```
>>> from colour import SpectralDistribution
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
... }
>>> sd = SpectralDistribution(data, name="Custom")
>>> plot_single_sd(sd)
(<Figure size ... with 1 Axes>, <...Axes...>)
```

colour.plotting.plot_multi_sds

colour.plotting.**plot_multi_sds**(*sds: collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions*, *plot_kwargs: Optional[Union[dict, List[dict]]] = None*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given spectral distributions.

Parameters

- **sds** (collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a List of colour.SpectralDistribution class instances.
- **plot_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for the matplotlib.pyplot.plot() definition, used to control the style of the plotted spectral distributions. *plot_kwargs* ` can be either a single dictionary applied to all the plotted spectral distributions with the same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - illuminant : The illuminant used to compute the spectral distributions colours. The default is the illuminant associated with the whitepoint of the default plotting colourspace. illuminant can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
  - cmfs : The standard observer colour matching functions used for computing the spectral distributions colours. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
  - normalise_sd_colours : Whether to normalise the computed spectral distributions colours. The default is *True*.
  - use_sd_colours : Whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the matplotlib.pyplot.plot() definition color argument with pre-computed values. The default is *True*.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

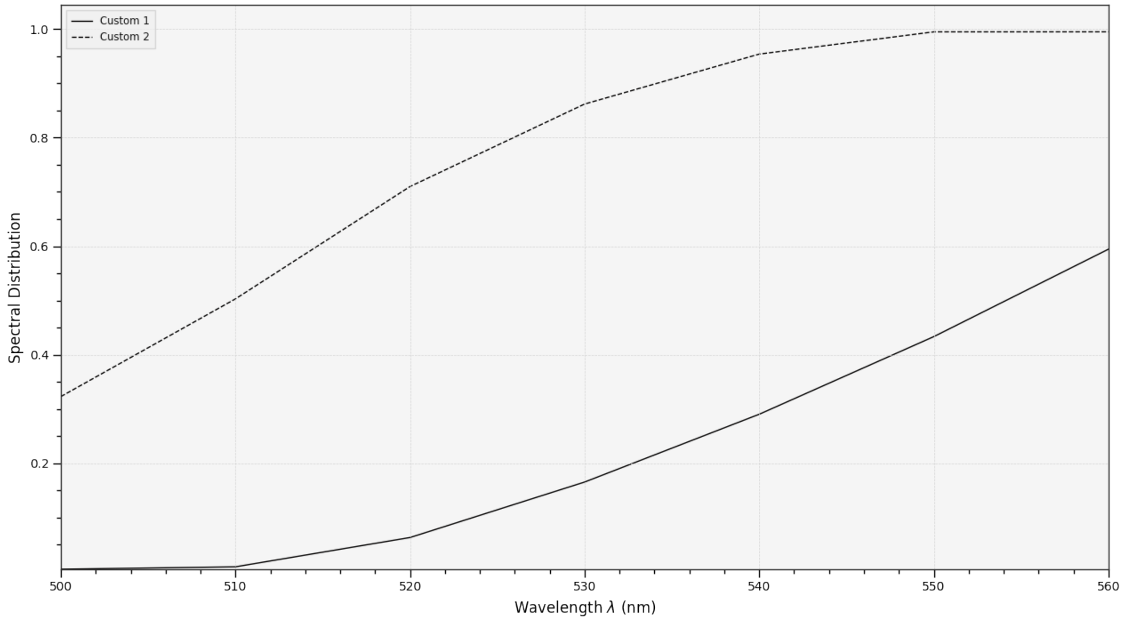
Examples

```
>>> from colour import SpectralDistribution
>>> data_1 = {
...     500: 0.004900,
...     510: 0.009300,
...     520: 0.063270,
...     530: 0.165500,
...     540: 0.290400,
...     550: 0.433450,
...     560: 0.594500,
... }
>>> data_2 = {
...     500: 0.323000,
...     510: 0.503000,
...     520: 0.710000,
...     530: 0.862000,
...     540: 0.954000,
...     550: 0.994950,
...     560: 0.995000,
```

(continues on next page)

(continued from previous page)

```
... }
>>> sd_1 = SpectralDistribution(data_1, name="Custom 1")
>>> sd_2 = SpectralDistribution(data_2, name="Custom 2")
>>> plot_kwargs = [
...     {"use_sd_colours": True},
...     {"use_sd_colours": True, "linestyle": "dashed"},
... ]
>>> plot_multi_sds([sd_1, sd_2], plot_kwargs=plot_kwargs)
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_single_cmfs

colour.plotting.**plot_single_cmfs**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer', **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given colour matching functions.

Parameters

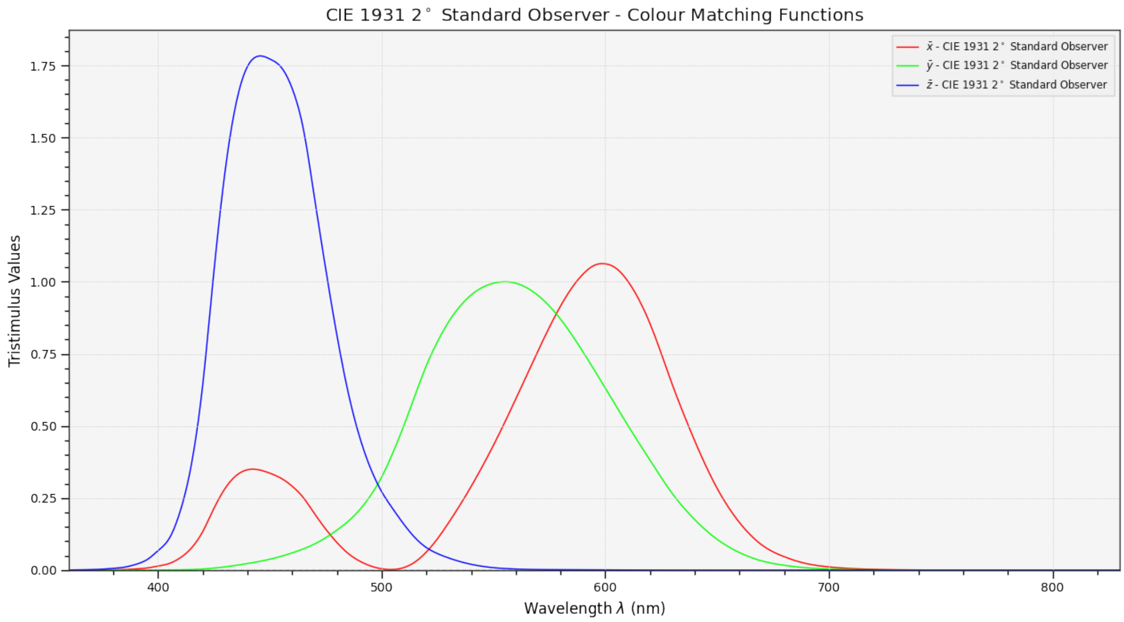
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions` | `str` | `collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions` | `str]`) – Colour matching functions to plot. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **kwargs** (`Any`) – {colour.plotting.artist(), `colour.plotting.plot_multi_cmfs()`, colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_cmfs("CIE 1931 2 Degree Standard Observer")
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_multi_cmfs

colour.plotting.**plot_multi_cmfs**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given colour matching functions.

Parameters

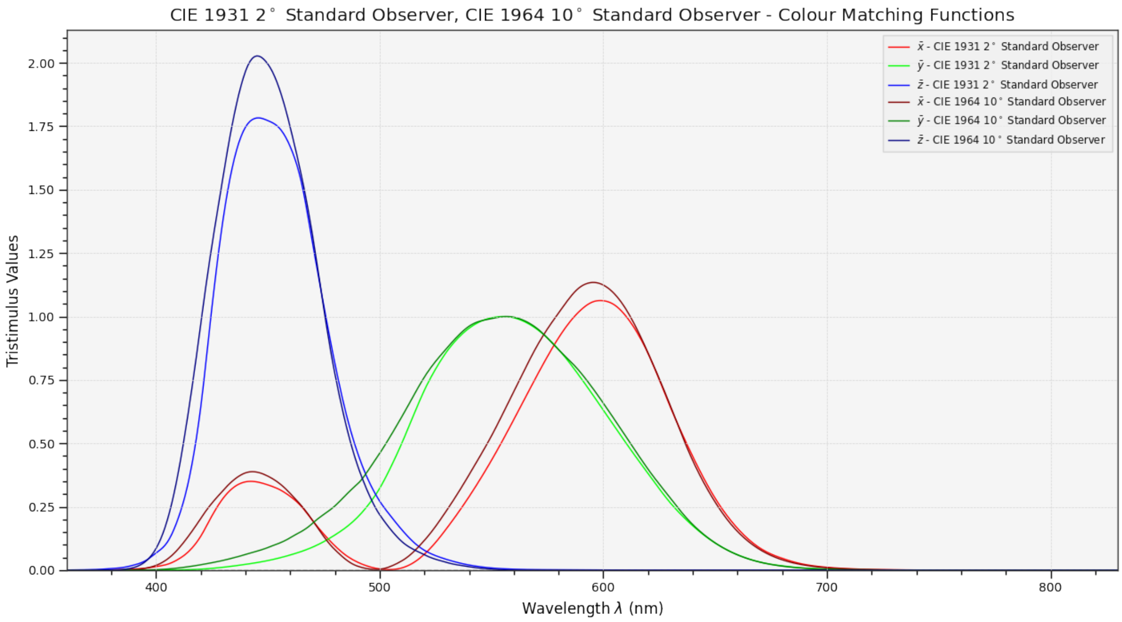
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]`) – Colour matching functions to plot. `cmfs` elements can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> cmfs = [
...     "CIE 1931 2 Degree Standard Observer",
...     "CIE 1964 10 Degree Standard Observer",
... ]
>>> plot_multi_cmfs(cmfs)
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_single_illuminant_sd

colour.plotting.**plot_single_illuminant_sd**(*illuminant:* colour.colorimetry.spectrum.SpectralDistribution | *str*, *cmfs:* colour.colorimetry.spectrum.MultiSpectralDistributions | *str* | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | *str*] = *'CIE 1931 2 Degree Standard Observer'*, ***kwargs:* Any) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given single illuminant spectral distribution.

Parameters

- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution | str) – Illuminant to plot. illuminant can be of any type or form supported by the colour.plotting.common.filter_illuminants() definition.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.render()}, See the documentation of

the previously listed definitions.

Returns Current figure and axes.

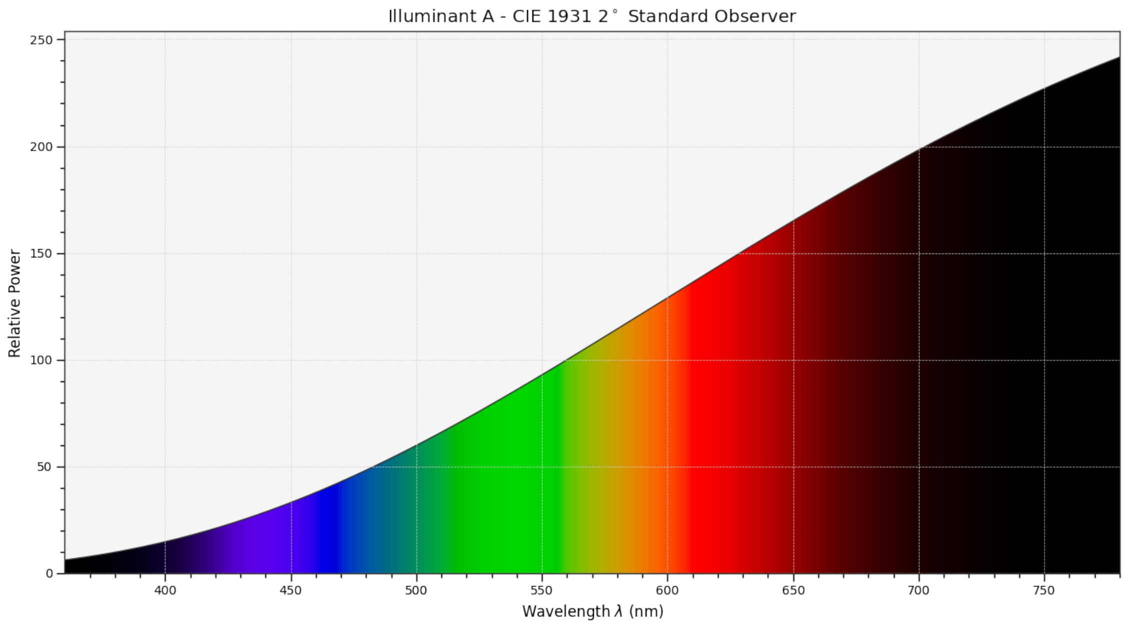
Return type tuple

References

[Spi15]

Examples

```
>>> plot_single_illuminant_sd("A")
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_multi_illuminant_sds

colour.plotting.**plot_multi_illuminant_sds**(*illuminants: colour.colorimetry.spectrum.SpectralDistribution | str | collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | str]*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given illuminants spectral distributions.

Parameters

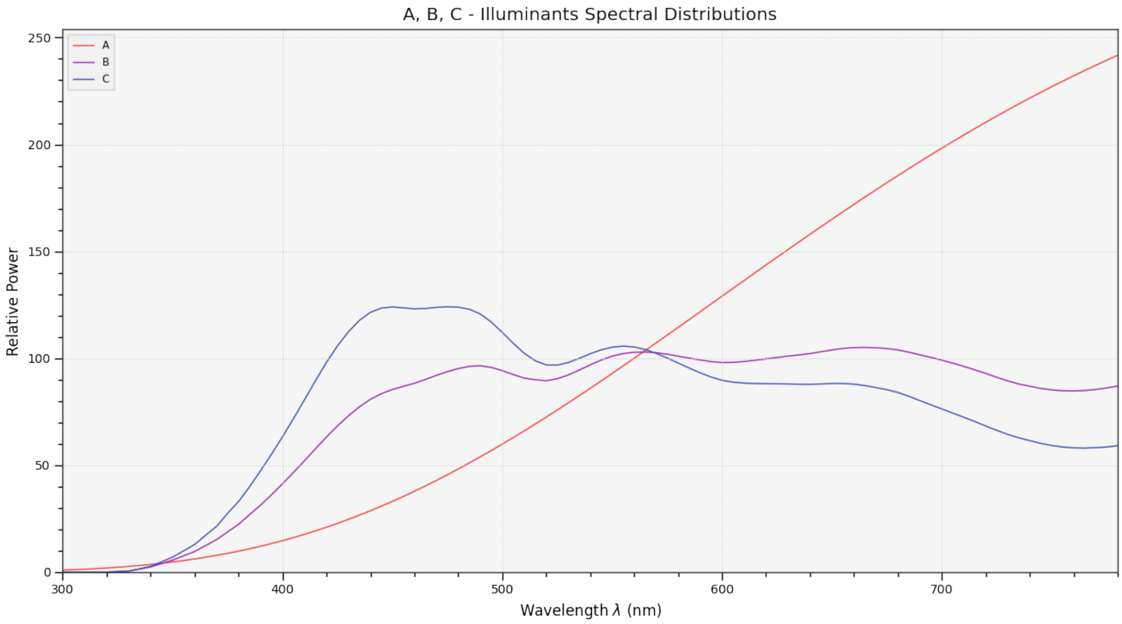
- **illuminants** (colour.colorimetry.spectrum.SpectralDistribution | str | collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | str]) – Illuminants to plot. illuminants elements can be of any type or form supported by the colour.plotting.common.filter_illuminants() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_multi_sds(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_illuminant_sds(["A", "B", "C"])
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_visible_spectrum

colour.plotting.**plot_visible_spectrum**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer', out_of_gamut_clipping: bool = True, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the visible colours spectrum using given standard observer *CIE XYZ* colour matching functions.

Parameters

- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **out_of_gamut_clipping** (bool) – Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

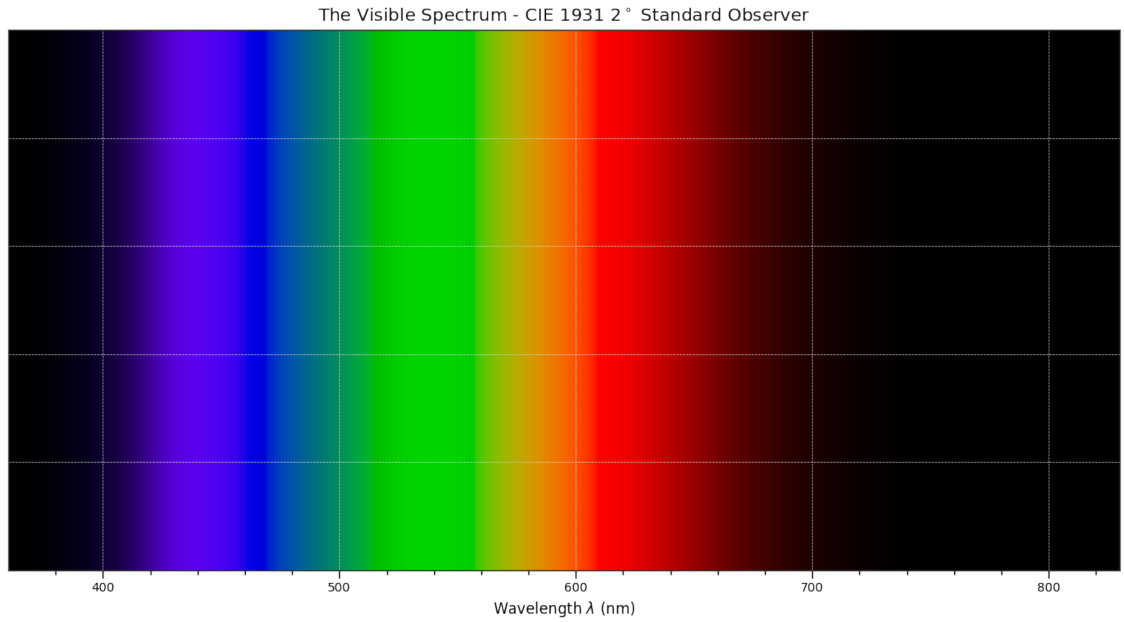
Return type tuple

References

[Spi15]

Examples

```
>>> plot_visible_spectrum()
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_single_lightness_function

colour.plotting.**plot_single_lightness_function**(*function: Union[Callable, str], **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *Lightness* function.

Parameters

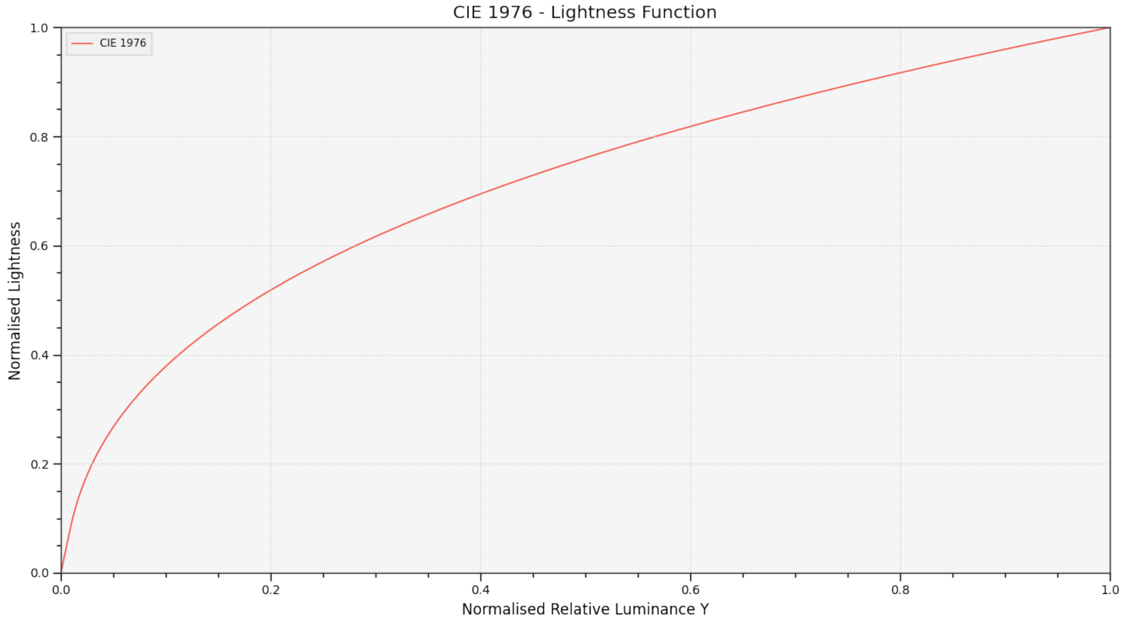
- **function** (Union[Callable, str]) – *Lightness* function to plot. function can be of any type or form supported by the colour.plotting.common.filter_passthrough() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_lightness_function("CIE 1976")
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_multi_lightness_functions

colour.plotting.**plot_multi_lightness_functions**(*functions: Union[Callable, str, collections.abc.Sequence[Union[Callable, str]]], **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *Lightness* functions.

Parameters

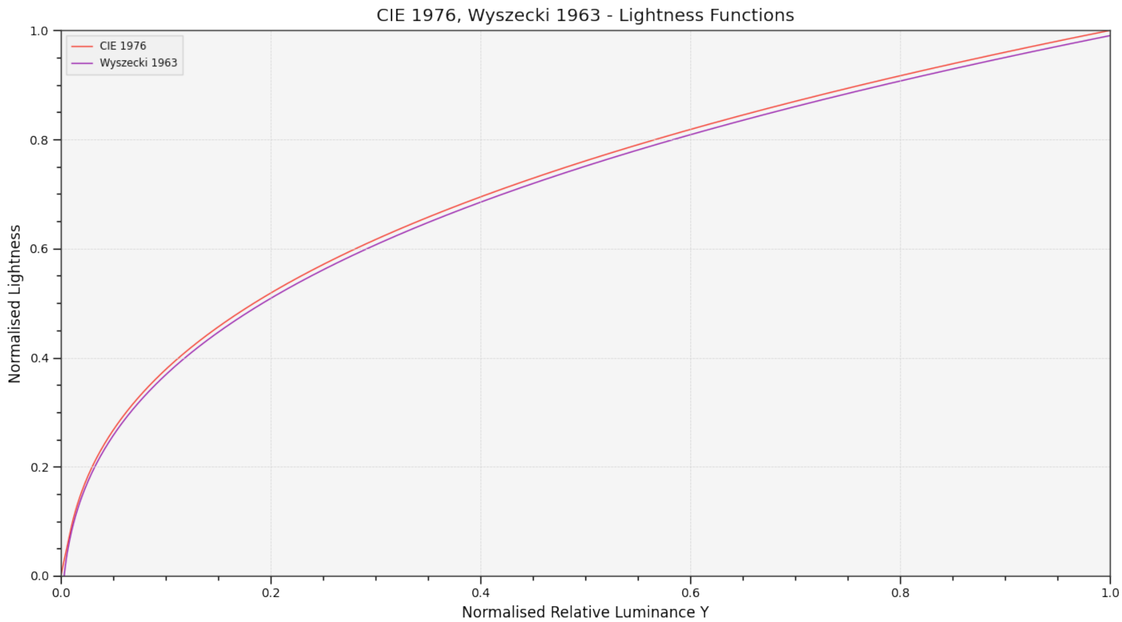
- **functions** (Union[Callable, str, collections.abc.Sequence[Union[Callable, str]]]) – *Lightness* functions to plot. functions elements can be of any type or form supported by the colour.plotting.common.filter_passthrough() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_lightness_functions(["CIE 1976", "Wyszecki 1963"])
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_single_luminance_function

`colour.plotting.`**`plot_single_luminance_function`**(*function: Union[Callable, str]*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *Luminance* function.

Parameters

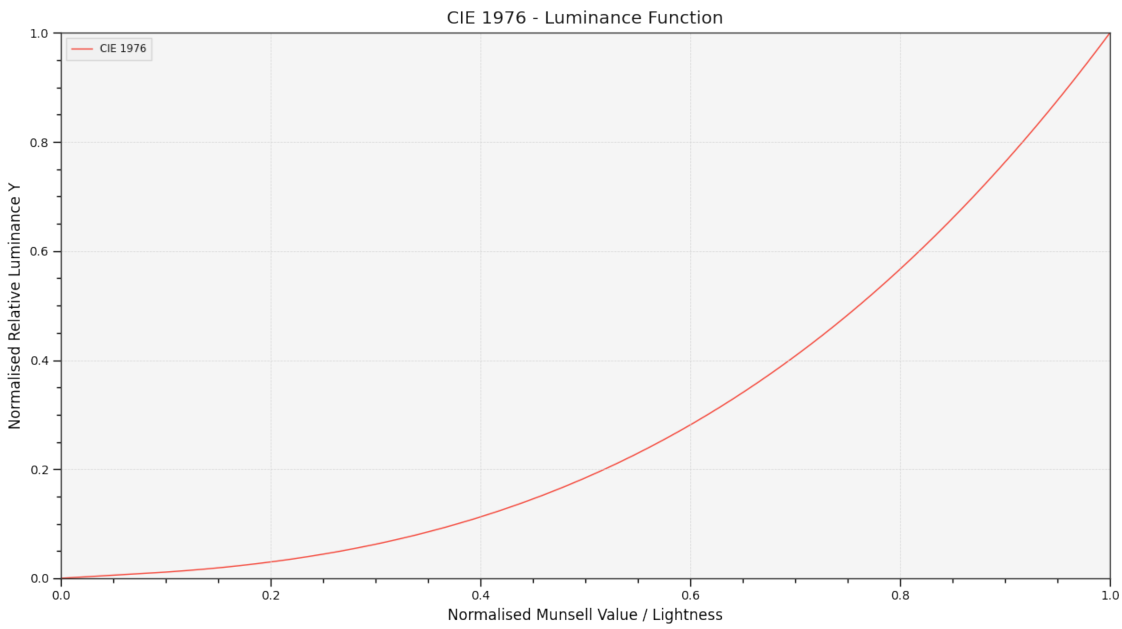
- **function** (`Union[Callable, str]`) – *Luminance* function to plot.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_luminance_function("CIE 1976")
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_multi_luminance_functions

colour.plotting.**plot_multi_luminance_functions**(*functions: Union[Callable, str, collections.abc.Sequence[Union[Callable, str]]], **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *Luminance* functions.

Parameters

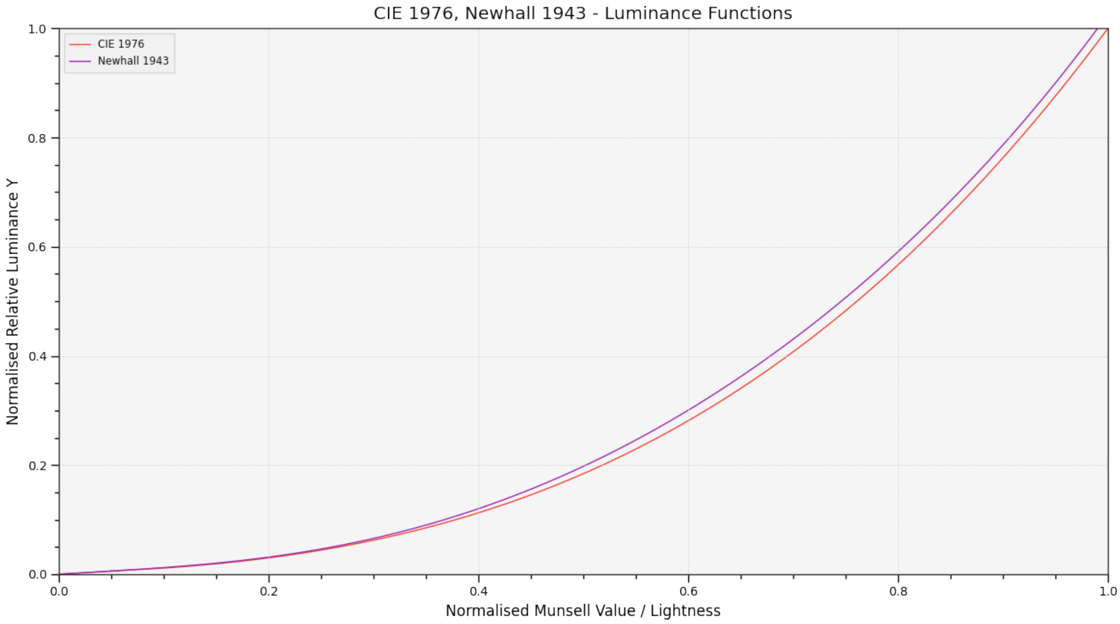
- **functions** (Union[Callable, str, collections.abc.Sequence[Union[Callable, str]]]) – *Luminance* functions to plot. functions elements can be of any type or form supported by the colour.plotting.common.filter_passthrough() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_luminance_functions(["CIE 1976", "Newhall 1943"])
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_blackbody_spectral_radiance

colour.plotting.**plot_blackbody_spectral_radiance**(*temperature: float = 3500*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer'*, *blackbody: str = 'VY Canis Major'*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given blackbody spectral radiance.

Parameters

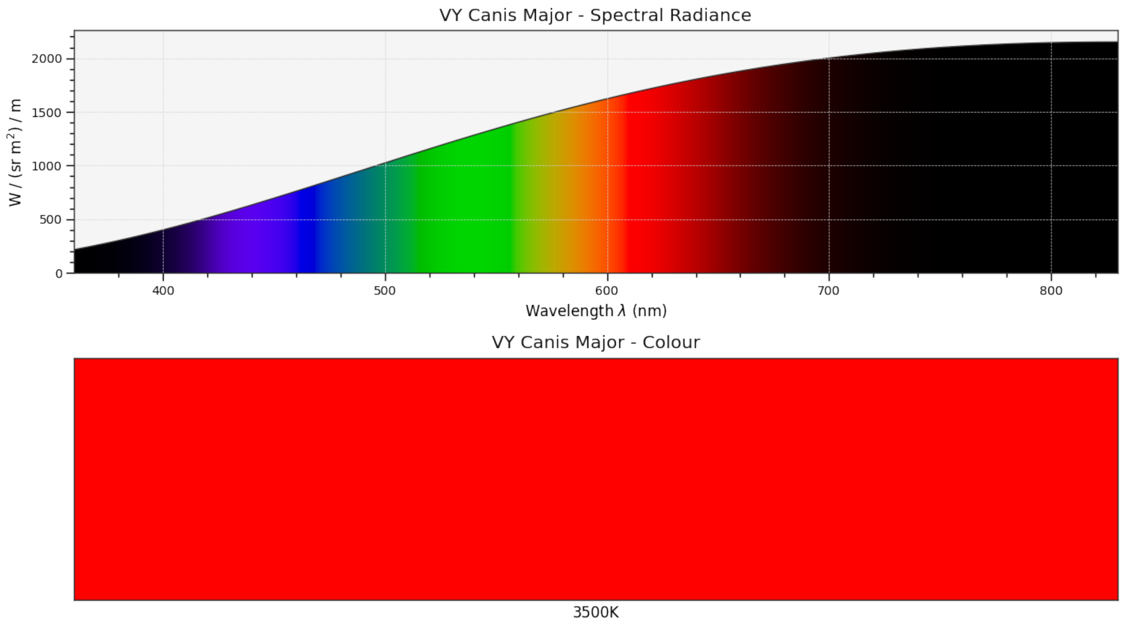
- **temperature** (float) – Blackbody temperature.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **blackbody** (str) – Blackbody name.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_blackbody_spectral_radiance(3500, blackbody="VY Canis Major")
...
(<Figure size ... with 2 Axes>, <...Axes...>)
```



colour.plotting.plot_blackbody_colours

colour.plotting.**plot_blackbody_colours**(*shape: colour.colorimetry.spectrum.SpectralShape = SpectralShape(150, 12500, 50)*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer'*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot blackbody colours.

Parameters

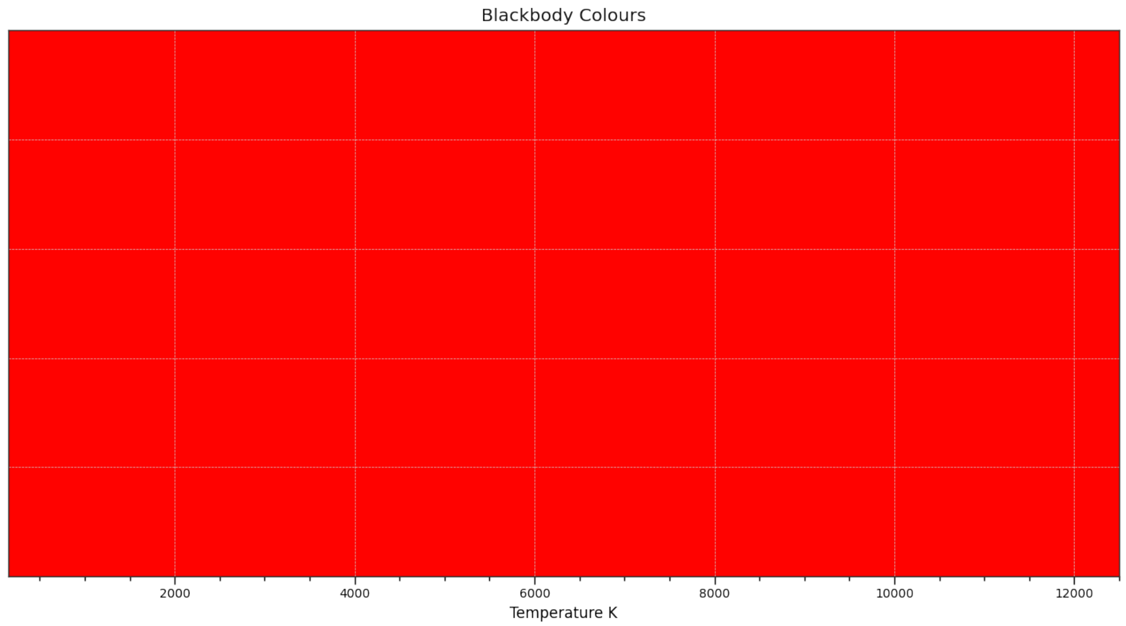
- **shape** (`colour.colorimetry.spectrum.SpectralShape`) – Spectral shape to use as plot boundaries.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions` | `str` | `collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]`) – Standard observer colour matching functions used for computing the blackbody colours. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_blackbody_colours(SpectralShape(150, 12500, 50))
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Colour Vision Deficiency

`colour.plotting`

`plot_cvd_simulation_Machado2009`(RGB[, ...])	Perform colour vision deficiency simulation on given *RGB* colourspace array using *Machado et al. (2009)* model.

colour.plotting.plot_cvd_simulation_Machado2009

colour.plotting.**plot_cvd_simulation_Machado2009**(*RGB: ArrayLike, deficiency: Literal['Deuteranomaly', 'Protanomaly', 'Tritanomaly'] | str = 'Protanomaly', severity: float = 0.5, M_a: ArrayLike | None = None, **kwargs: Any*) → Tuple[plt.Figure, plt.Axes]

Perform colour vision deficiency simulation on given *RGB* colourspace array using *Machado et al. (2009)* model.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array.
- **deficiency** (Literal['Deuteranomaly', 'Protanomaly', 'Tritanomaly'] | str) – Colour blindness / vision deficiency type.
- **severity** (float) – Severity of the colour vision deficiency in domain [0, 1].
- **M_a** (ArrayLike | None) – Anomalous trichromacy matrix to use instead of Machado (2010) pre-computed matrix.

- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_image(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Return type Tuple[plt.Figure, plt.Axes]

Notes

- Input *RGB* array is expected to be linearly encoded.

Returns Current figure and axes.

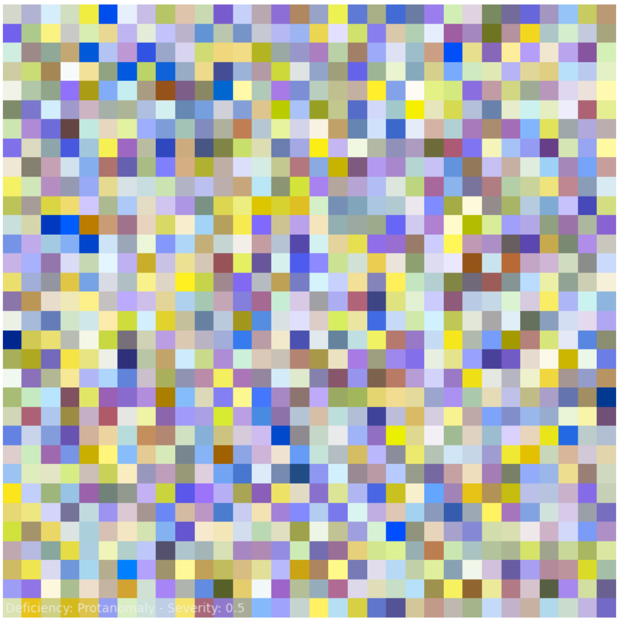
Return type tuple

Parameters

- **RGB** (ArrayLike) –
- **deficiency** (Literal['Deuteranomaly', 'Protanomaly', 'Tritanomaly'] | str) –
- **severity** (float) –
- **M_a** (ArrayLike | None) –
- **kwargs** (Any) –

Examples

```
>>> import numpy as np
>>> RGB = np.random.rand(32, 32, 3)
>>> plot_cvd_simulation_Machado2009(RGB)
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Colour Characterisation

colour.plotting

plot_single_colour_checker([colour_checker])	Plot given colour checker.
plot_multi_colour_checkers(colour_checkers, ...)	Plot and compares given colour checkers.

colour.plotting.plot_single_colour_checker

colour.plotting.**plot_single_colour_checker**(*colour_checker: colour.characterisation.datasets.colour_checkers.chromaticity_coordina | str = 'ColorChecker24 - After November 2014'*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given colour checker.

Parameters

- **colour_checker** (colour.characterisation.datasets.colour_checkers.chromaticity_coordinates.ColourChecker | str) – Color checker to plot. colour_checker can be of any type or form supported by the colour.plotting.common.filter_colour_checkers() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_multi_colour_swatches(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_colour_checker("ColorChecker 2005")
(<Figure size ... with 1 Axes>, <...Axes...>)
```

colour.plotting.plot_multi_colour_checkers

colour.plotting.**plot_multi_colour_checkers**(*colour_checkers:* colour.characterisation.datasets.colour_checkers.chromaticity_coordina | *str* | collections.abc.Sequence*[*colour.characterisation.datasets.colour_checkers.ch | *str]*, ***kwargs:* Any) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot and compares given colour checkers.

Parameters

- **colour_checkers** (`colour.characterisation.datasets.colour_checkers.chromaticity_coordinates.ColourChecker` | `str` | `collections.abc.Sequence[colour.characterisation.datasets.colour_checkers.chromaticity_coordinates.ColourChecker` | `str]`) – Color checker to plot, count must be less than or equal to 2. `colour_checkers` elements can be of any type or form supported by the `colour.plotting.common.filter_colour_checkers()` definition.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_colour_swatches()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_multi_colour_checkers(["ColorChecker 1976", "ColorChecker 2005"])
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```

Corresponding Chromaticities

`colour.plotting`

`plot_corresponding_chromaticities_prediction([...])`	Plot given chromatic adaptation model corresponding chromaticities prediction.

colour.plotting.plot_corresponding_chromaticities_prediction

colour.plotting.**plot_corresponding_chromaticities_prediction**(*experiment: Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.CorrespondingColourDataset] = 1, model: Union[Literal['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Zhai 2018', 'Von Kries'], str] = 'Von Kries', corresponding_chromaticities_prediction_kwargs: dict | None = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given chromatic adaptation model corresponding chromaticities prediction.

Parameters

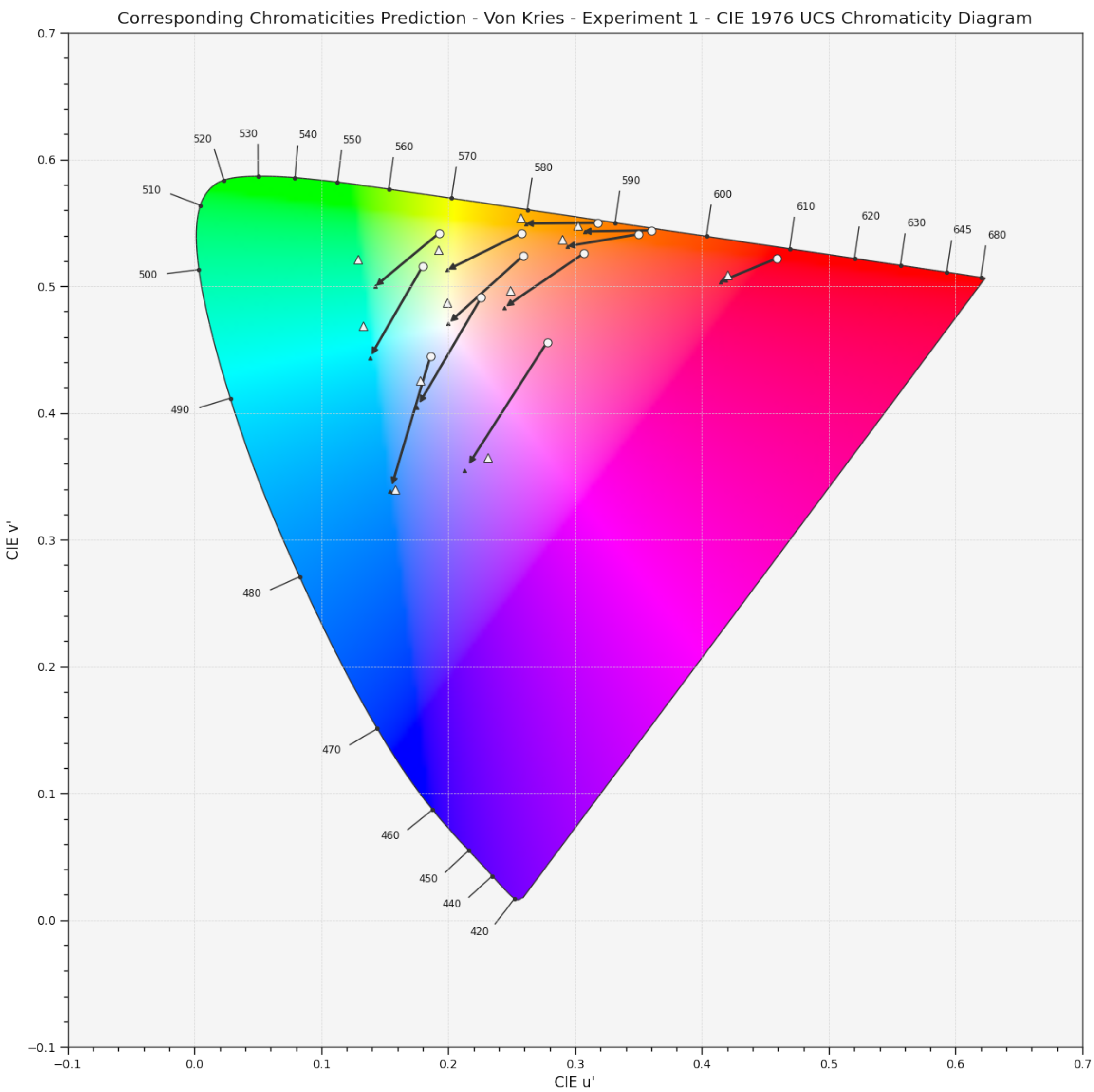
- **experiment** (`Union[Literal[1, 2, 3, 4, 6, 8, 9, 11, 12], colour.corresponding.prediction.CorrespondingColourDataset]`) – *Breneman (1987)* experiment number or `colour.CorrespondingColourDataset` class instance.
- **model** (`Union[Literal['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Zhai 2018', 'Von Kries'], str]`) – Corresponding chromaticities prediction model name.
- **corresponding_chromaticities_prediction_kwargs** (`dict | None`) – Keyword arguments for the `colour.corresponding_chromaticities_prediction()` definition.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_corresponding_chromaticities_prediction(1, "Von Kries")
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



CIE Chromaticity Diagrams

colour.plotting

plot_chromaticity_diagram_CIE1931([cmfs, ...])	Plot the *CIE 1931 Chromaticity Diagram*.
plot_chromaticity_diagram_CIE1960UCS([cmfs, ...])	Plot the *CIE 1960 UCS Chromaticity Diagram*.
plot_chromaticity_diagram_CIE1976UCS([cmfs, ...])	Plot the *CIE 1976 UCS Chromaticity Diagram*.
plot_sds_in_chromaticity_diagram_CIE1931(sds)	Plot given spectral distribution chromaticity coordinates into the *CIE 1931 Chromaticity Diagram*.
plot_sds_in_chromaticity_diagram_CIE1960UCS(sds)	Plot given spectral distribution chromaticity coordinates into the *CIE 1960 UCS Chromaticity Diagram*.
plot_sds_in_chromaticity_diagram_CIE1976UCS(sds)	Plot given spectral distribution chromaticity coordinates into the *CIE 1976 UCS Chromaticity Diagram*.

colour.plotting.plot_chromaticity_diagram_CIE1931

colour.plotting.**plot_chromaticity_diagram_CIE1931**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralD | str] = 'CIE 1931 2 Degree Standard Observer', show_diagram_colours: bool = True, show_spectral_locus: bool = True, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *CIE 1931 Chromaticity Diagram*.

Parameters

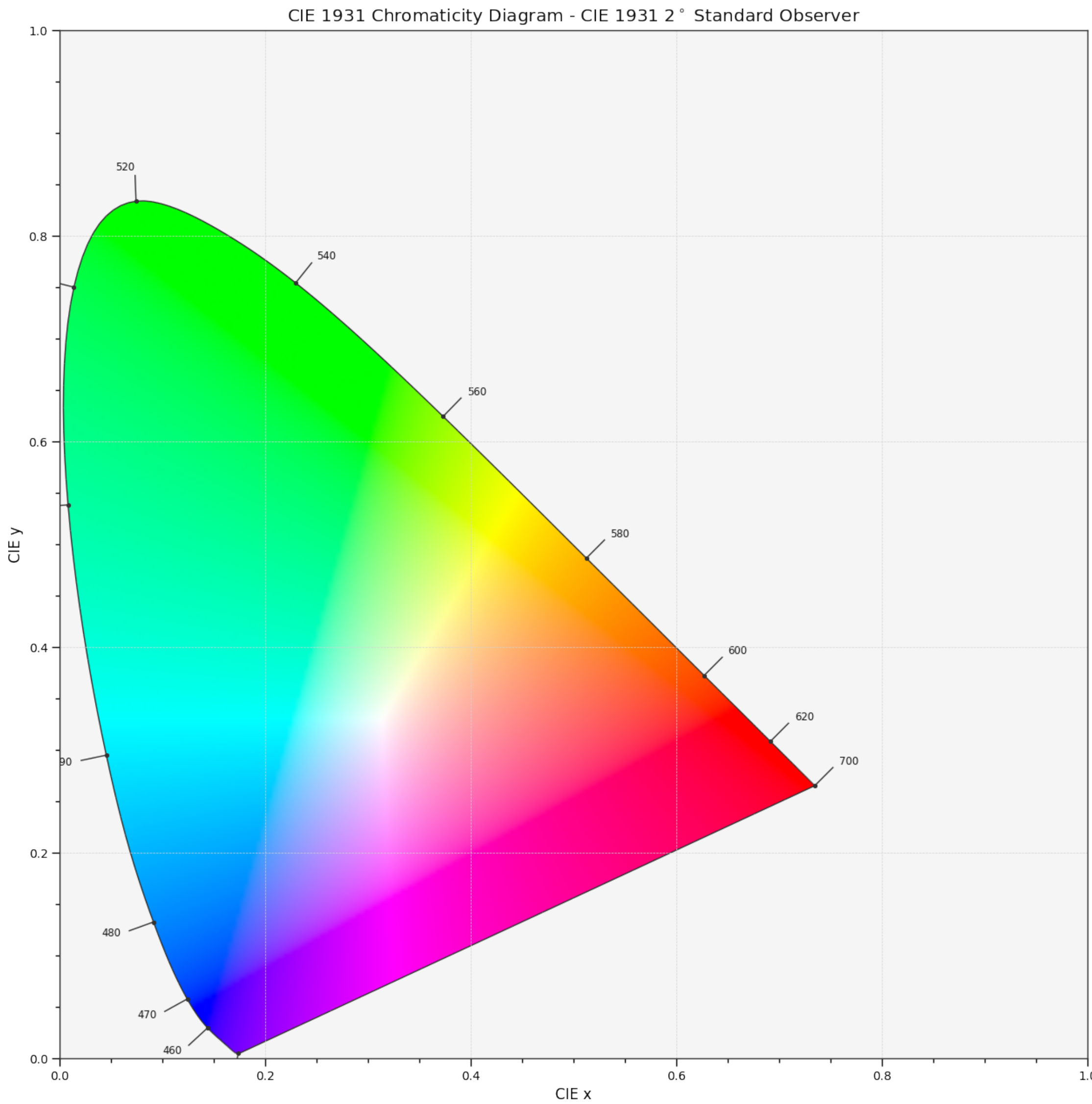
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **show_diagram_colours** (bool) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool) – Whether to display the *Spectral Locus*.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_CIE1931()
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_chromaticity_diagram_CIE1960UCS**(*cmfs:* colour.colorimetry.spectrum.MultiSpectralDistributions | *str* | collections.abc.Sequence*[*colour.colorimetry.spectrum.MultiSpectr *| str] = 'CIE 1931 2 Degree Standard Observer'*, *show_diagram_colours: bool = True*, *show_spectral_locus: bool = True*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]`) – Standard observer colour matching functions used for computing the spectral locus boundaries. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()`

definition.

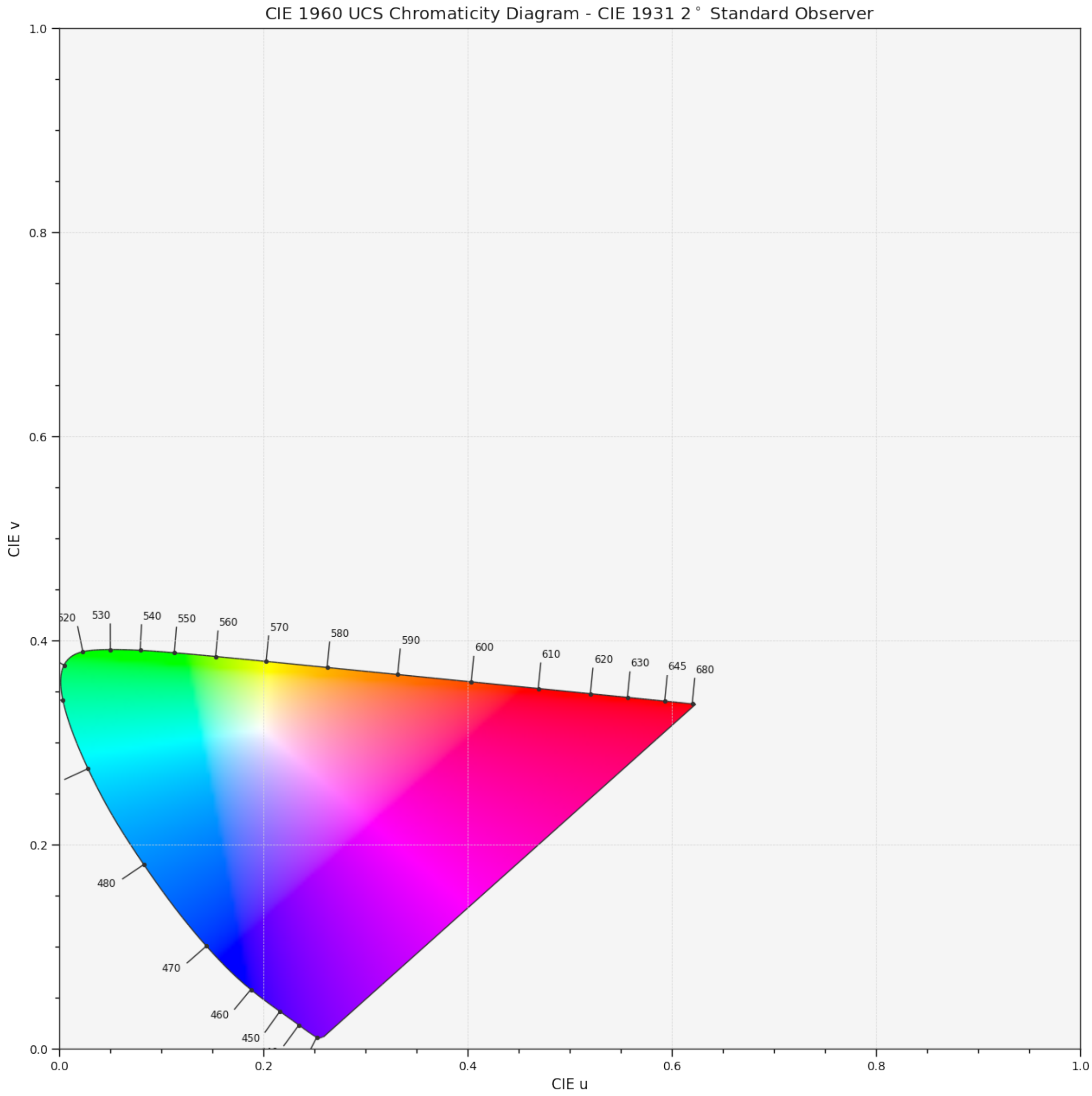
- **show_diagram_colours** (bool) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool) – Whether to display the *Spectral Locus*.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_CIE1960UCS()
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_chromaticity_diagram_CIE1976UCS**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer', show_diagram_colours: bool = True, show_spectral_locus: bool = True, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

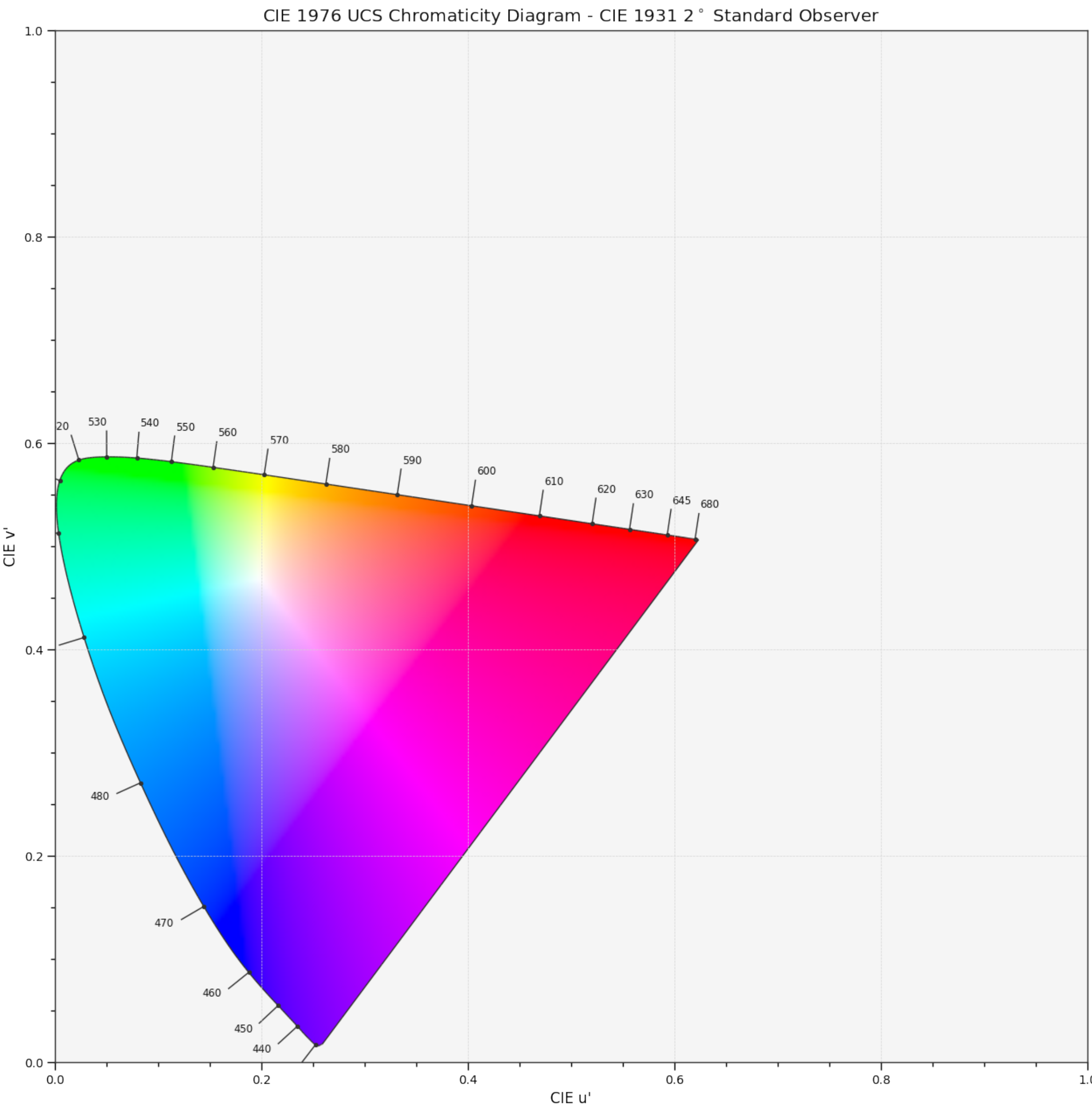
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **show_diagram_colours** (bool) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool) – Whether to display the *Spectral Locus*.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_CIE1976UCS()
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_sds_in_chromaticity_diagram_CIE1931**(*sds: collections.abc.Sequence[colour.colorimetry.spectrum.Spectra | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.MultiSpectralDistributions, cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiS | str] = 'CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable_CIE1931: Callable = plot_chromaticity_diagram_CIE1931, annotate_kwargs: Optional[Union[dict, List[dict]]] = None, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given spectral distribution chromaticity coordinates into the *CIE 1931 Chromaticity Diagram*.

Parameters

- **sds** (`collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.MultiSpectralDistributions`) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single `colour.MultiSpectralDistributions` class instance, a list of `colour.MultiSpectralDistributions` class instances or a list of `colour.SpectralDistribution` class instances.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]`) – Standard observer colour matching functions used for computing the spectral locus boundaries. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **chromaticity_diagram_callable_CIE1931** (`Callable`) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **annotate_kwargs** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. `annotate_kwargs` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution. The following special keyword arguments can also be used:
  - `annotate` : Whether to annotate the spectral distributions.
- **plot_kwargs** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.plot()` definition, used to control the style of the plotted spectral distributions. *plot_kwargs* ` can be either a single dictionary applied to all the plotted spectral distributions with the same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - `illuminant` : The illuminant used to compute the spectral distributions colours. The default is the illuminant associated with the whitepoint of the

default plotting colourspace. illuminant can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.

  – cmfs : The standard observer colour matching functions used for computing the spectral distributions colours. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.

  – normalise_sd_colours : Whether to normalise the computed spectral distributions colours. The default is *True*.

  – use_sd_colours : Whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the matplotlib.pyplot.plot() definition color argument with pre-computed values. The default is *True*.

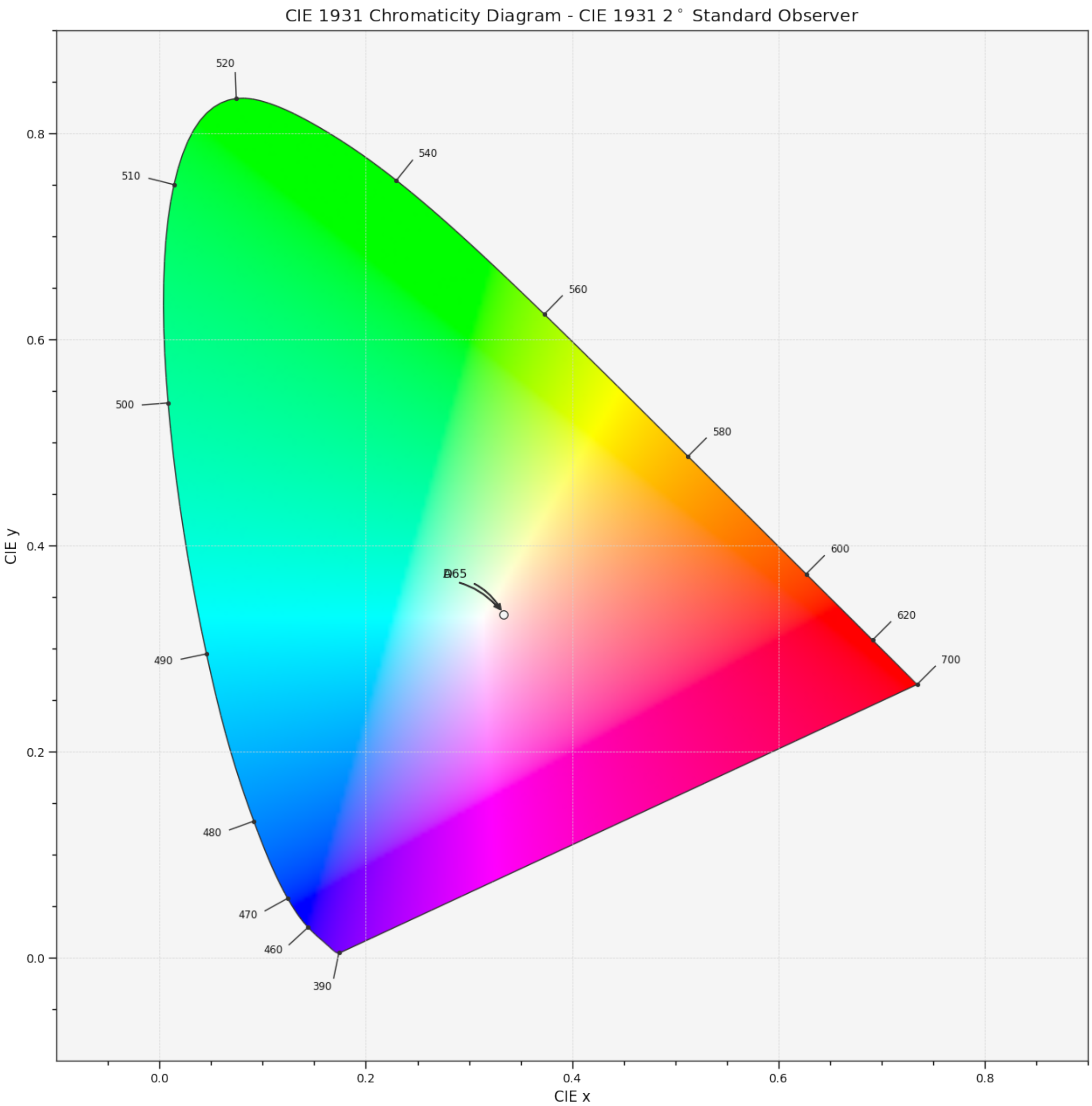
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> A = SDS_ILLUMINANTS["A"]
>>> D65 = SDS_ILLUMINANTS["D65"]
>>> plot_sds_in_chromaticity_diagram_CIE1931([A, D65])
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_sds_in_chromaticity_diagram_CIE1960UCS**(*sds: collections.abc.Sequence[colour.colorimetry.spectrum.Spe | colour.colorimetry.spectrum.MultiSpectralDistributio | colour.colorimetry.spectrum.MultiSpectralDistributio cmfs: colour.colorimetry.spectrum.MultiSpectralDistributio | str | collections.abc.Sequence[colour.colorimetry.spectrum.Mu | str] = 'CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable_CIE1960UCS: Callable = plot_chromaticity_diagram_CIE1960UCS, annotate_kwargs: Optional[Union[dict, List[dict]]] = None, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given spectral distribution chromaticity coordinates into the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **sds** (collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **chromaticity_diagram_callable_CIE1960UCS** (Callable) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **annotate_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for the matplotlib.pyplot.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. annotate_kwargs can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution. The following special keyword arguments can also be used:
  - annotate : Whether to annotate the spectral distributions.
- **plot_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for the matplotlib.pyplot.plot() definition, used to control the style of the plotted spectral distributions. *plot_kwargs*` can be either a single dictionary applied to all the plotted spectral distributions with the same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - illuminant : The illuminant used to compute the spectral distributions

colours. The default is the illuminant associated with the whitepoint of the default plotting colourspace. `illuminant` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.

  – `cmfs` : The standard observer colour matching functions used for computing the spectral distributions colours. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.

  – `normalise_sd_colours` : Whether to normalise the computed spectral distributions colours. The default is *True*.

  – `use_sd_colours` : Whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the `matplotlib.pyplot.plot()` definition `color` argument with pre-computed values. The default is *True*.

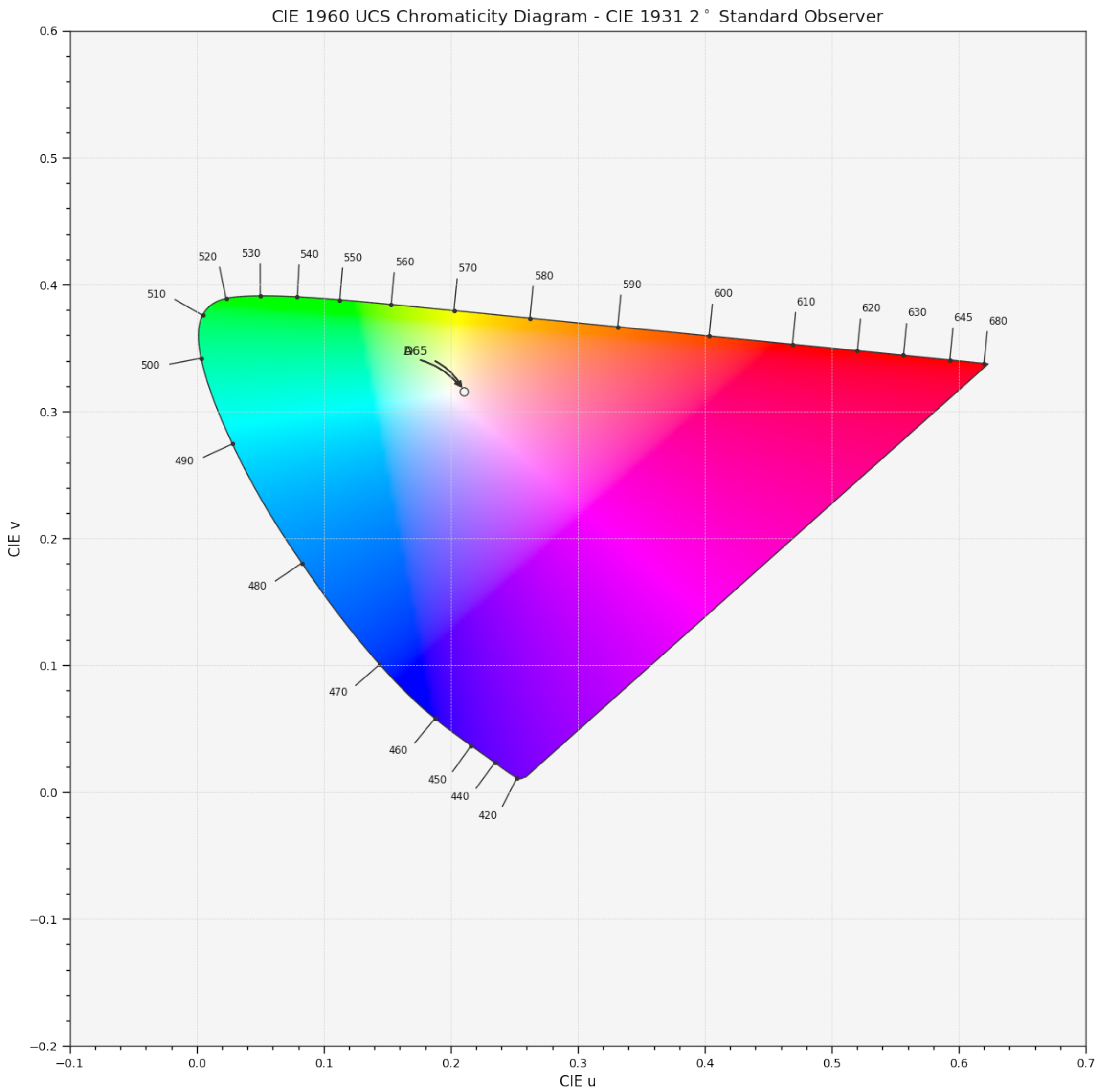
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> A = SDS_ILLUMINANTS["A"]
>>> D65 = SDS_ILLUMINANTS["D65"]
>>> plot_sds_in_chromaticity_diagram_CIE1960UCS([A, D65])
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_sds_in_chromaticity_diagram_CIE1976UCS**(*sds: collections.abc.Sequence[colour.colorimetry.spectrum.Spe | colour.colorimetry.spectrum.MultiSpectralDistributio | colour.colorimetry.spectrum.MultiSpectralDistributio cmfs: colour.colorimetry.spectrum.MultiSpectralDistributio | str | collections.abc.Sequence[colour.colorimetry.spectrum.Mu | str] = 'CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable_CIE1976UCS: Callable = plot_chromaticity_diagram_CIE1976UCS, annotate_kwargs: Optional[Union[dict, List[dict]]] = None, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given spectral distribution chromaticity coordinates into the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **sds** (`collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.MultiSpectralDistributions`) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single `colour.MultiSpectralDistributions` class instance, a list of `colour.MultiSpectralDistributions` class instances or a list of `colour.SpectralDistribution` class instances.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]`) – Standard observer colour matching functions used for computing the spectral locus boundaries. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **chromaticity_diagram_callable_CIE1976UCS** (`Callable`) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **annotate_kwargs** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. `annotate_kwargs` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution. The following special keyword arguments can also be used:
  - `annotate` : Whether to annotate the spectral distributions.
- **plot_kwargs** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.plot()` definition, used to control the style of the plotted spectral distributions. *plot_kwargs*` can be either a single dictionary applied to all the plotted spectral distributions with the same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - `illuminant` : The illuminant used to compute the spectral distributions

colours. The default is the illuminant associated with the whitepoint of the default plotting colourspace. `illuminant` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.

  - `cmfs` : The standard observer colour matching functions used for computing the spectral distributions colours. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
  - `normalise_sd_colours` : Whether to normalise the computed spectral distributions colours. The default is *True*.
  - `use_sd_colours` : Whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the `matplotlib.pyplot.plot()` definition `color` argument with pre-computed values. The default is *True*.

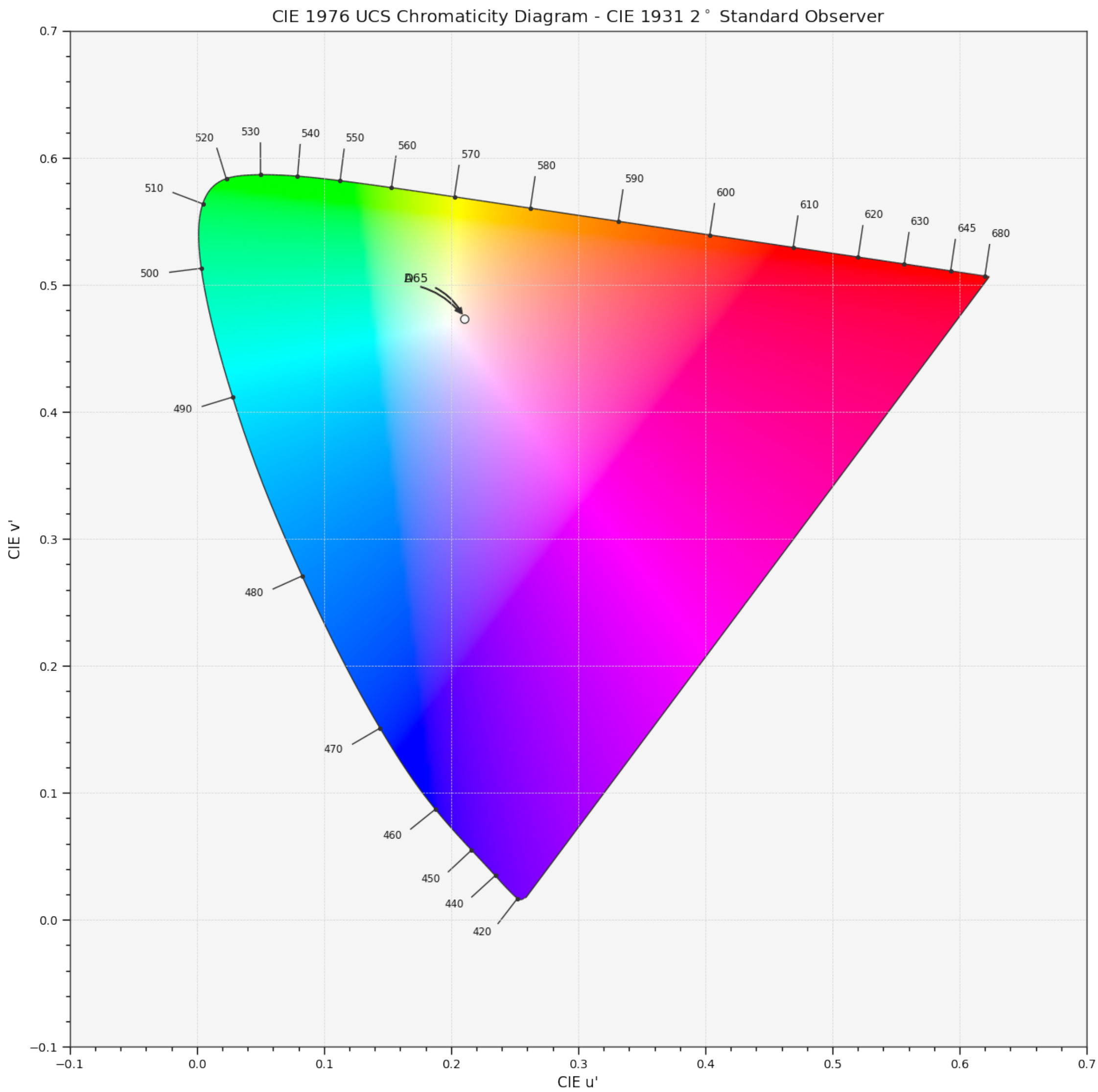
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> A = SDS_ILLUMINANTS["A"]
>>> D65 = SDS_ILLUMINANTS["D65"]
>>> plot_sds_in_chromaticity_diagram_CIE1976UCS([A, D65])
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Ancillary Objects

`colour.plotting.diagrams`

plot_spectral_locus([cmfs, ...])	Plot the *Spectral Locus* according to given method.
plot_chromaticity_diagram_colours([samples, ...])	Plot the *Chromaticity Diagram* colours according to given method.
plot_chromaticity_diagram([cmfs, ...])	Plot the *Chromaticity Diagram* according to given method.
plot_sds_in_chromaticity_diagram(sds[, ...])	Plot given spectral distribution chromaticity co-ordinates into the *Chromaticity Diagram* using given method.

colour.plotting.diagrams.plot_spectral_locus

colour.plotting.diagrams.**plot_spectral_locus**(*cmfs: MultiSpectralDistributions | str | Sequence[MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer', spectral_locus_colours: ArrayLike | str | None = None, spectral_locus_opacity: float = 1, spectral_locus_labels: Sequence | None = None, method: Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str = 'CIE 1931', **kwargs: Any*) → Tuple[plt.Figure, plt.Axes]

Plot the *Spectral Locus* according to given method.

Parameters

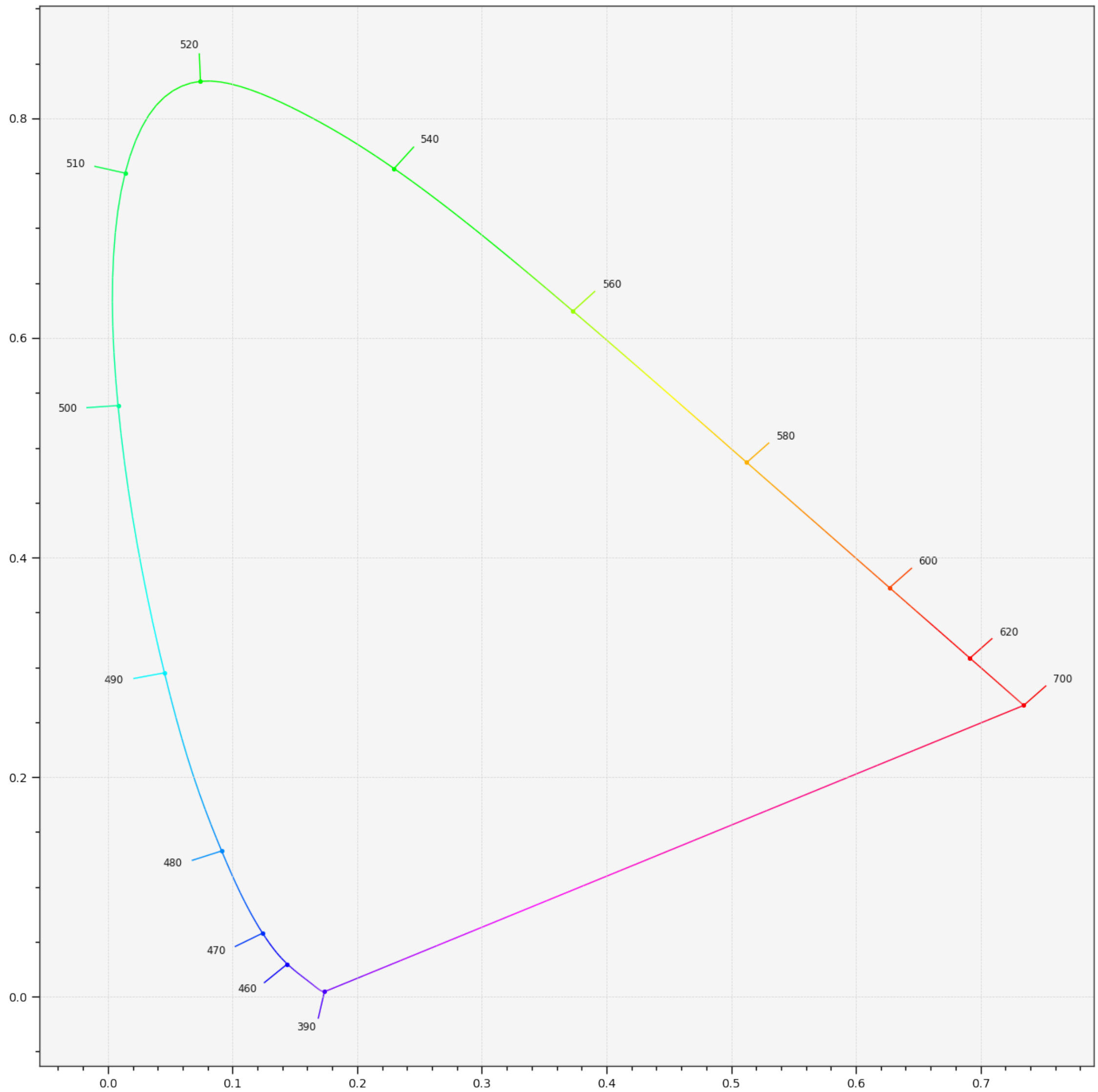
- **cmfs** (MultiSpectralDistributions | str | Sequence[MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **spectral_locus_colours** (ArrayLike | str | None) – Colours of the *Spectral Locus*, if spectral_locus_colours is set to *RGB*, the colours will be computed according to the corresponding chromaticity coordinates.
- **spectral_locus_opacity** (float) – Opacity of the *Spectral Locus*.
- **spectral_locus_labels** (Sequence | None) – Array of wavelength labels used to customise which labels will be drawn around the spectral locus. Passing an empty array will result in no wavelength labels being drawn.
- **method** (Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str) – *Chromaticity Diagram* method.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_spectral_locus(spectral_locus_colours="RGB")
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.diagrams.plot_chromaticity_diagram_colours

colour.plotting.diagrams.**plot_chromaticity_diagram_colours**(*samples: int = 256, diagram_colours: ArrayLike | str | None = None, diagram_opacity: float = 1, diagram_clipping_path: ArrayLike | None = None, cmfs: MultiSpectralDistributions | str | Sequence[MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer', method: Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str = 'CIE 1931', **kwargs: Any*) → Tuple[plt.Figure, plt.Axes]

Plot the *Chromaticity Diagram* colours according to given method.

Parameters

- **samples** (int) – Samples count on one axis when computing the *Chromaticity Diagram* colours.

- **diagram_colours** (ArrayLike | str | None) – Colours of the *Chromaticity Diagram*, if diagram_colours is set to *RGB*, the colours will be computed according to the corresponding coordinates.
- **diagram_opacity** (float) – Opacity of the *Chromaticity Diagram*.
- **diagram_clipping_path** (ArrayLike | None) – Path of points used to clip the *Chromaticity Diagram* colours.
- **cmfs** (MultiSpectralDistributions | str | Sequence[MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **method** (Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str) – *Chromaticity Diagram* method.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_colours(diagram_colours="RGB")
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```

colour.plotting.diagrams.plot_chromaticity_diagram

colour.plotting.diagrams.**plot_chromaticity_diagram**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectral | str] = 'CIE 1931 2 Degree Standard Observer', show_diagram_colours: bool = True, show_spectral_locus: bool = True, method: Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str] = 'CIE 1931', **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *Chromaticity Diagram* according to given method.

Parameters

- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.

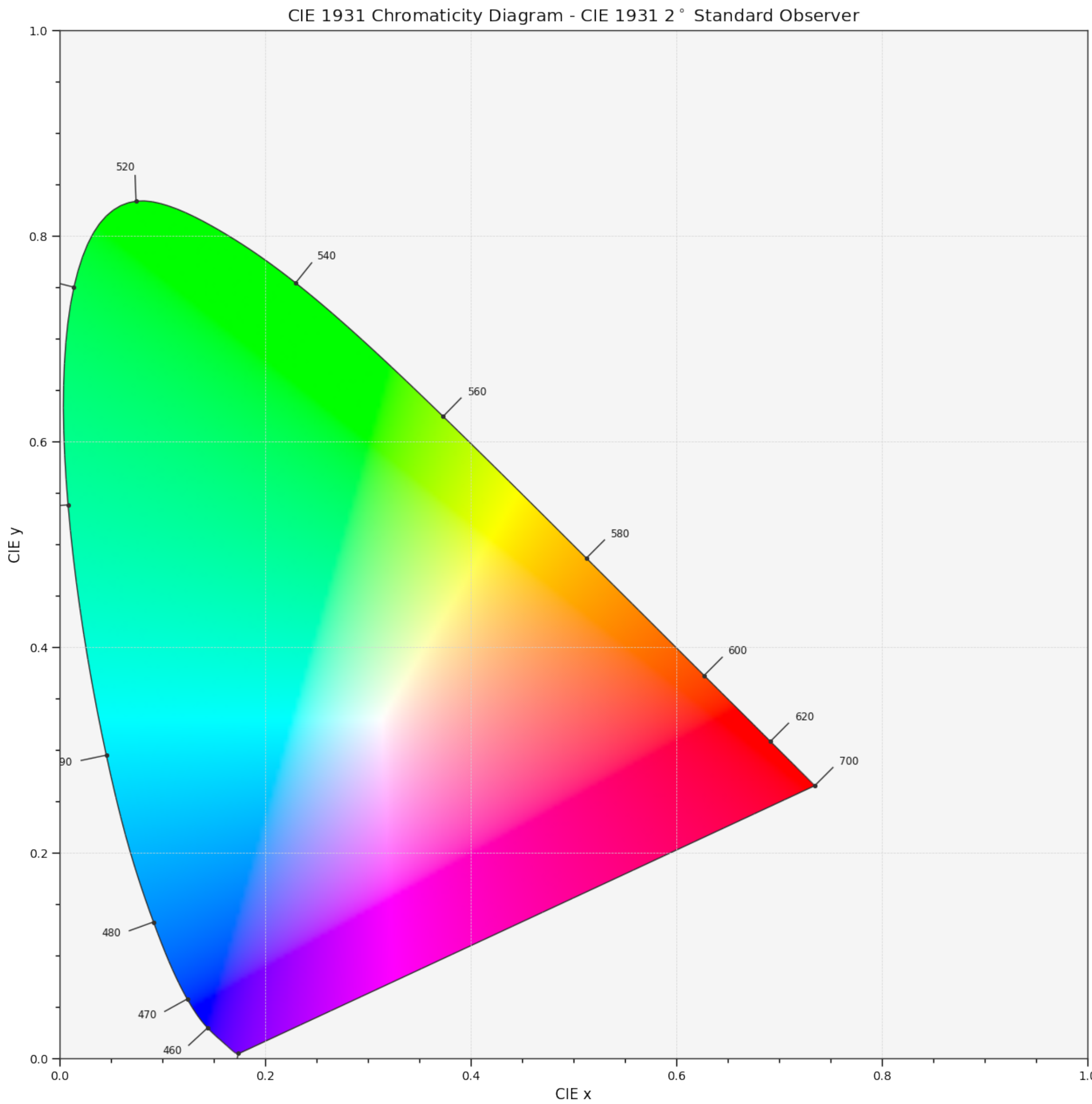
- **show_diagram_colours** (`bool`) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (`bool`) – Whether to display the *Spectral Locus*.
- **method** (`Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str]`) – *Chromaticity Diagram* method.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_spectral_locus()`, `colour.plotting.diagrams.plot_chromaticity_diagram_colours()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_chromaticity_diagram()
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.diagrams.plot_sds_in_chromaticity_diagram

colour.plotting.diagrams.**plot_sds_in_chromaticity_diagram**(*sds: collections.abc.Sequence[colour.colorimetry.spectrum.Spect | colour.colorimetry.spectrum.MultiSpectralDistribution | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistribution cmfs: colour.colorimetry.spectrum.MultiSpectralDistribution | str | collections.abc.Sequence[colour.colorimetry.spectrum.Multi | str] = 'CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable: Callable = plot_chromaticity_diagram, method: Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str] = 'CIE 1931', annotate_kwargs: Optional[Union[dict, List[dict]]] = None, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given spectral distribution chromaticity coordinates into the *Chromaticity Diagram* using given method.

Parameters

- **sds** (collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a List of colour.SpectralDistribution class instances.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **chromaticity_diagram_callable** (Callable) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str]) – *Chromaticity Diagram* method.
- **annotate_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for the matplotlib.pyplot.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. annotate_kwargs can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution. The following special keyword arguments can also be used:
  - annotate : Whether to annotate the spectral distributions.

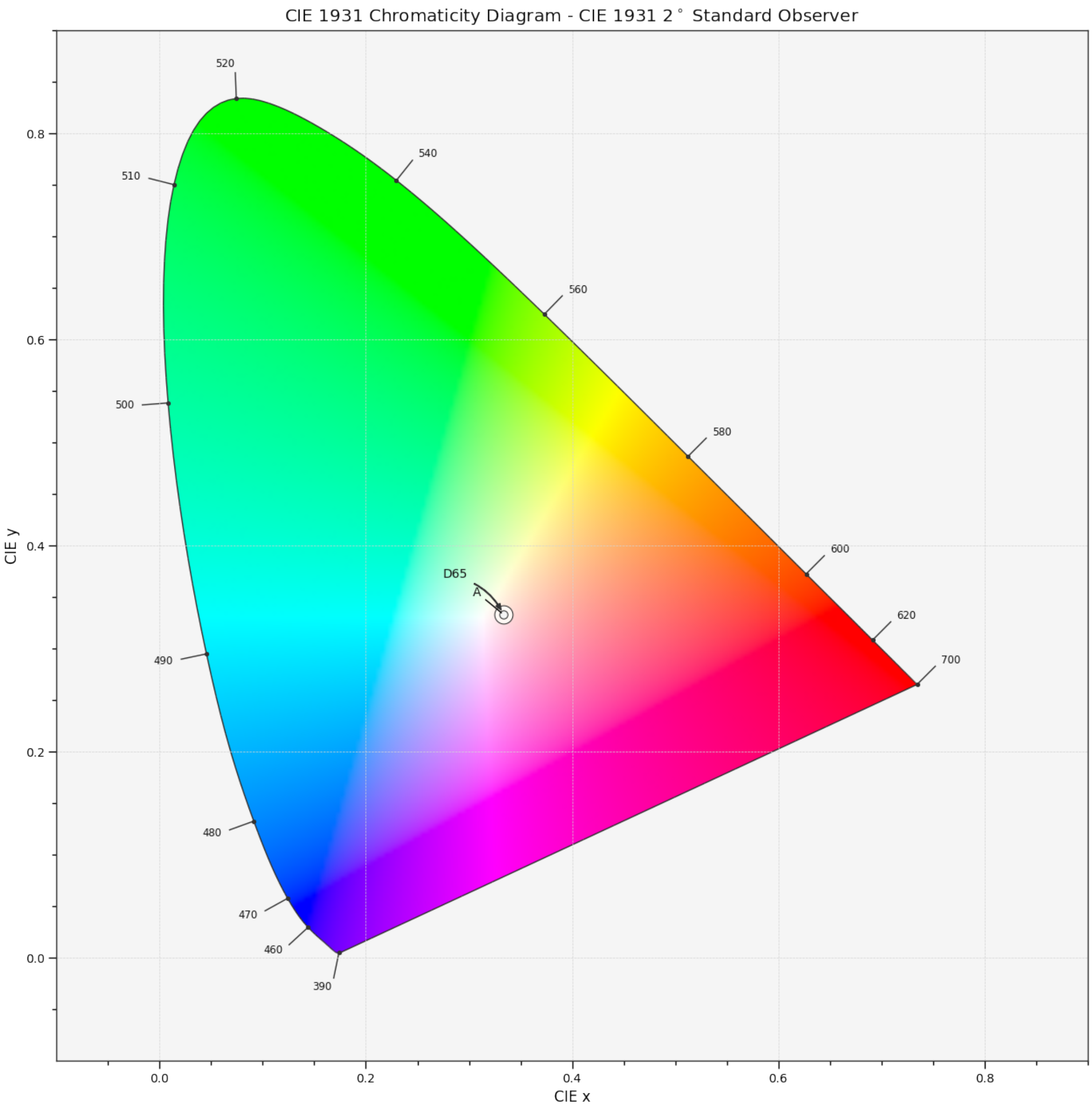
- **plot_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for the matplotlib.pyplot.plot() definition, used to control the style of the plotted spectral distributions. *plot_kwargs* ` can be either a single dictionary applied to all the plotted spectral distributions with the same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - illuminant : The illuminant used to compute the spectral distributions colours. The default is the illuminant associated with the whitepoint of the default plotting colourspace. illuminant can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
  - cmfs : The standard observer colour matching functions used for computing the spectral distributions colours. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
  - normalise_sd_colours : Whether to normalise the computed spectral distributions colours. The default is *True*.
  - use_sd_colours : Whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the matplotlib.pyplot.plot() definition color argument with pre-computed values. The default is *True*.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> A = SDS_ILLUMINANTS["A"]
>>> D65 = SDS_ILLUMINANTS["D65"]
>>> annotate_kwargs = [
...     {"xytext": (-25, 15), "arrowprops": {"arrowstyle": "-"}},
...     {},
... ]
>>> plot_kwargs = [
...     {
...         "illuminant": SDS_ILLUMINANTS["E"],
...         "markersize": 15,
...         "normalise_sd_colours": True,
...         "use_sd_colours": True,
...     },
...     {"illuminant": SDS_ILLUMINANTS["E"]},
... ]
>>> plot_sds_in_chromaticity_diagram(
...     [A, D65], annotate_kwargs=annotate_kwargs, plot_kwargs=plot_kwargs
... )
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Colour Models

`colour.plotting`

plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931(...)	Plot given *RGB* colourspaces in the *CIE 1931 Chromaticity Diagram*.
plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS(...)	Plot given *RGB* colourspaces in the *CIE 1960 UCS Chromaticity Diagram*.
plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS(...)	Plot given *RGB* colourspaces in the *CIE 1976 UCS Chromaticity Diagram*.
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(RGB)	Plot given *RGB* colourspace array in the *CIE 1931 Chromaticity Diagram*.
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS(RGB)	Plot given *RGB* colourspace array in the *CIE 1960 UCS Chromaticity Diagram*.
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS(RGB)	Plot given *RGB* colourspace array in the *CIE 1976 UCS Chromaticity Diagram*.
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931([...])	Plot *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1931 Chromaticity Diagram*.
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS([...])	Plot *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1960 UCS Chromaticity Diagram*.
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS([...])	Plot *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1976 UCS Chromaticity Diagram*.
plot_single_cctf(cctf[, cctf_decoding])	Plot given colourspace colour component transfer function.
plot_multi_cctfs(cctfs[, cctf_decoding])	Plot given colour component transfer functions.
plot_constant_hue_loci(data[, model, ...])	Plot given constant hue loci colour matches data such as that from [HB95] or [EF98] that are easily loaded with Colour - Datasets.

colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931**(*colourspaces: colour.models.rgb.rgb_colourspace.RGB_ | str | collections.abc.Sequence[colour.models.rgb.rg | str], cmfs: colour.colorimetry.spectrum.MultiSpectr | str | collections.abc.Sequence[colour.colorimetry.sp | str] = 'CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable_CIE1931: Callable = plot_chromaticity_diagram_CIE1931, show_whitepoints: bool = True, show_pointer_gamut: bool = False, chromatically_adapt: bool = False, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *RGB* colourspaces in the *CIE 1931 Chromaticity Diagram*.

Parameters

- **colourspaces** (colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]) – *RGB* colourspaces to plot. colourspaces elements can be of any type or form supported by the colour.plotting.common.filter_RGB_colourspaces() definition.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **chromaticity_diagram_callable_CIE1931** (Callable) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **show_whitepoints** (bool) – Whether to display the *RGB* colourspaces whitepoints.
- **show_pointer_gamut** (bool) – Whether to display the *Pointer's Gamut*.
- **chromatically_adapt** (bool) – Whether to chromatically adapt the *RGB* colourspaces given in colourspaces to the whitepoint of the default plotting colourspace.
- **plot_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for the matplotlib.pyplot.plot() definition, used to control the style of the plotted *RGB* colourspaces. plot_kwargs can be either a single dictionary applied to all the plotted *RGB* colourspaces with the same settings or a sequence of dictionaries with different settings for each plotted *RGB* colourspace.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_pointer_gamut(), colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931(
...     ["ITU-R BT.709", "ACEScg", "S-Gamut"]
... )
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```

colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS**(*colourspaces: colour.models.rgb.rgb_colourspace.R | str | collections.abc.Sequence[colour.models.rg | str], cmfs: colour.colorimetry.spectrum.MultiSpe | str | collections.abc.Sequence[colour.colorimetr | str] = 'CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable_CIE1960UCS: Callable = plot_chromaticity_diagram_CIE1960U show_whitepoints: bool = True, show_pointer_gamut: bool = False, chromatically_adapt: bool = False, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *RGB* colourspaces in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]) – *RGB* colourspaces to plot. colourspaces elements can be of any type or form supported by the colour.plotting.common.filter_RGB_colourspaces() definition.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **chromaticity_diagram_callable_CIE1960UCS** (Callable) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **show_whitepoints** (bool) – Whether to display the *RGB* colourspaces whitepoints.
- **show_pointer_gamut** (bool) – Whether to display the *Pointer's Gamut*.
- **chromatically_adapt** (bool) – Whether to chromatically adapt the *RGB* colourspaces given in colourspaces to the whitepoint of the default plotting colourspace.
- **plot_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for

the `matplotlib.pyplot.plot()` definition, used to control the style of the plotted *RGB* colourspaces. `plot_kwargs` can be either a single dictionary applied to all the plotted *RGB* colourspaces with the same settings or a sequence of dictionaries with different settings for each plotted *RGB* colourspace.

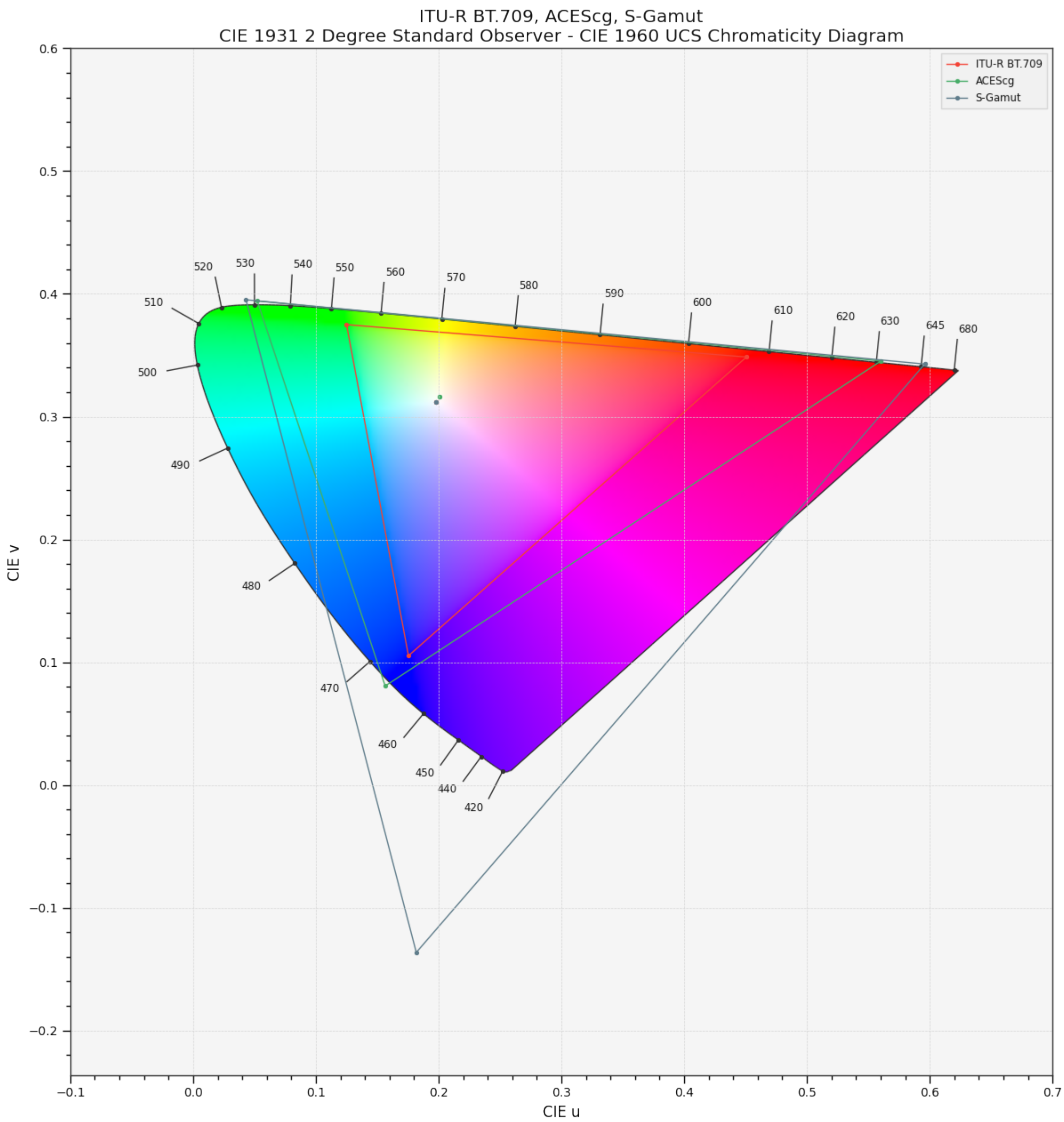
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.models.plot_pointer_gamut()`, `colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS(
...     ["ITU-R BT.709", "ACEScg", "S-Gamut"]
... )
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS**(*colourspaces: colour.models.rgb.rgb_colourspace.R | str | collections.abc.Sequence[colour.models.rg | str], cmfs: colour.colorimetry.spectrum.MultiSpe | str | collections.abc.Sequence[colour.colorimetr | str] = 'CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable_CIE1976UCS: Callable = plot_chromaticity_diagram_CIE1976U show_whitepoints: bool = True, show_pointer_gamut: bool = False, chromatically_adapt: bool = False, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *RGB* colourspaces in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace` | `str` | `collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]`) – *RGB* colourspaces to plot. `colourspaces` elements can be of any type or form supported by the `colour.plotting.common.filter_RGB_colourspaces()` definition.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions` | `str` | `collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]`) – Standard observer colour matching functions used for computing the spectral locus boundaries. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **chromaticity_diagram_callable_CIE1976UCS** (`Callable`) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **show_whitepoints** (`bool`) – Whether to display the *RGB* colourspaces whitepoints.
- **show_pointer_gamut** (`bool`) – Whether to display the *Pointer's Gamut*.
- **chromatically_adapt** (`bool`) – Whether to chromatically adapt the *RGB* colourspaces given in `colourspaces` to the whitepoint of the default plotting colourspace.
- **plot_kwargs** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for

the matplotlib.pyplot.plot() definition, used to control the style of the plotted *RGB* colourspaces. plot_kwargs can be either a single dictionary applied to all the plotted *RGB* colourspaces with the same settings or a sequence of dictionaries with different settings for each plotted *RGB* colourspace.

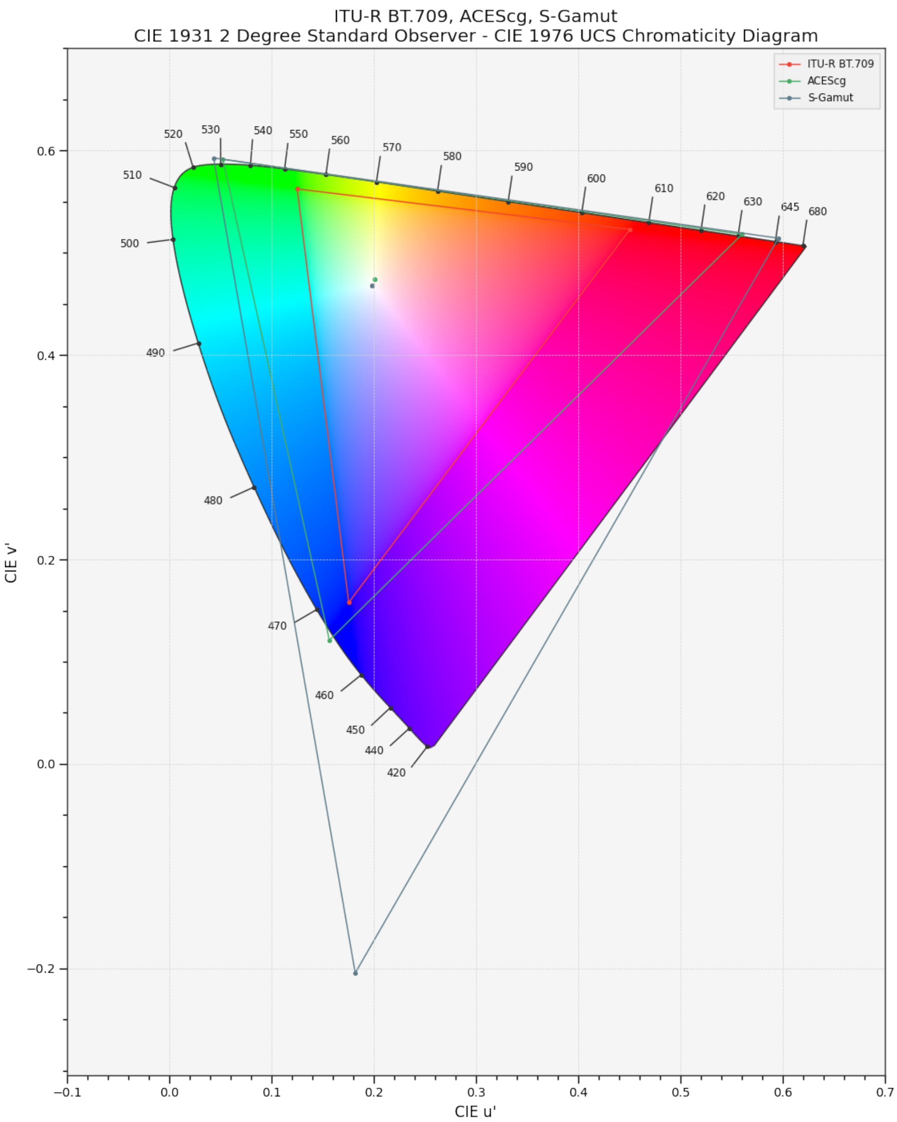
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_pointer_gamut(), colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS(
...     ["ITU-R BT.709", "ACEScg", "S-Gamut"]
... )
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931**(*RGB: ArrayLike*, *colourspace: colour.models.rgb.rgb_colourspace.RG | str | collections.abc.Sequence[colour.models.rgb | str] = 'sRGB'*, *chromaticity_diagram_callable_CIE1931: Callable = plot_RGB_colourspaces_in_chromaticity*, *scatter_kwargs: dict | None = None*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *RGB* colourspace array in the *CIE 1931 Chromaticity Diagram*.

Parameters

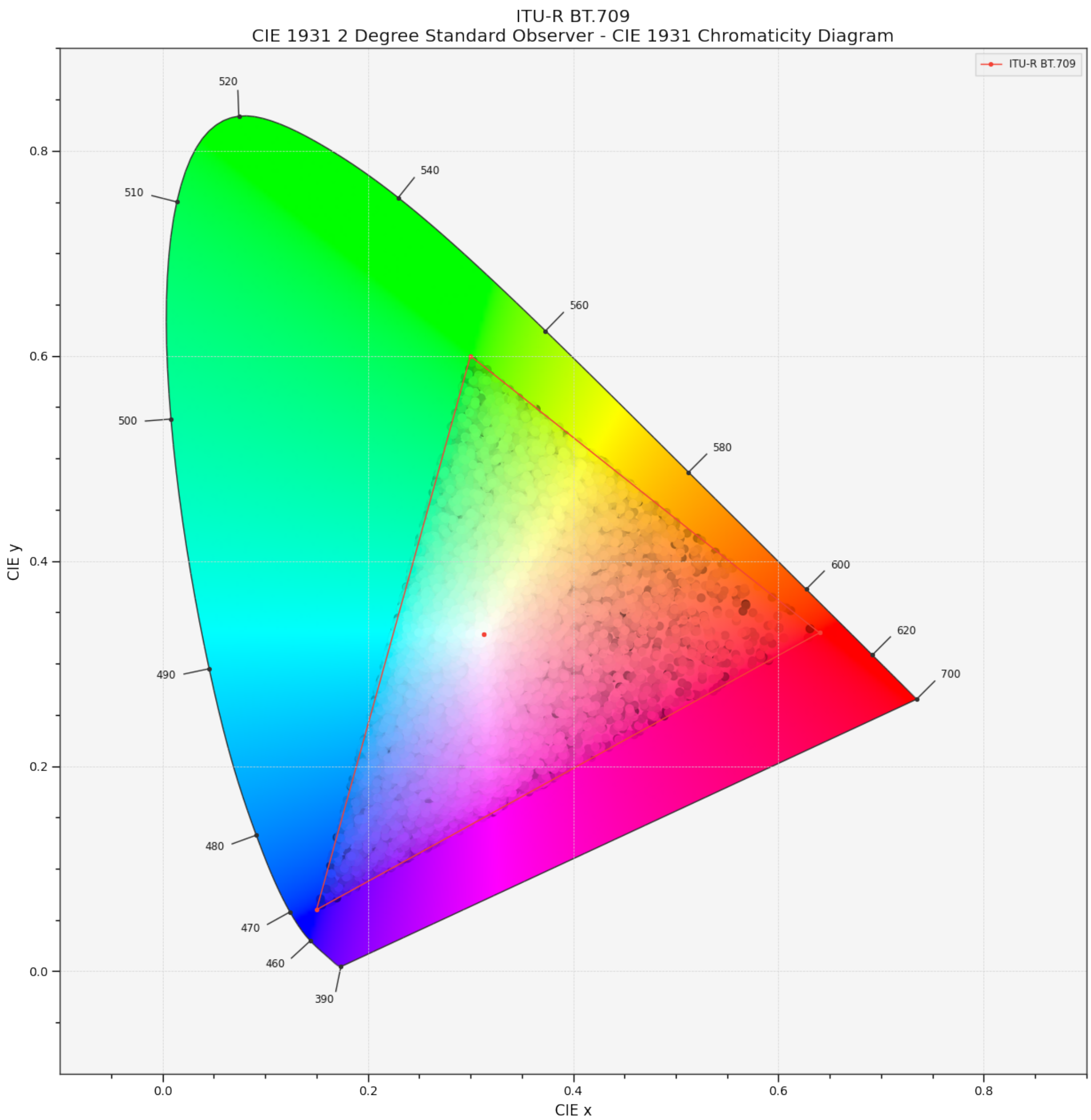
- **RGB** (ArrayLike) – *RGB* colourspace array.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.common.filter_RGB_colourspaces() definition.
- **chromaticity_diagram_callable_CIE1931** (Callable) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **scatter_kwargs** (dict | None) – Keyword arguments for the matplotlib.pyplot.scatter() definition. The following special keyword arguments can also be used:
  - c : If c is set to *RGB*, the scatter will use the colours as given by the RGB argument.
  - apply_cctf_encoding : If apply_cctf_encoding is set to *False*, the encoding colour component transfer function / opto-electronic transfer function is not applied when encoding the samples to the plotting space.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(
...     RGB, "ITU-R BT.709"
... )
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS**(*RGB: ArrayLike, colourspace: colour.models.rgb.rgb_colourspace | str | collections.abc.Sequence[colour.models. | str] = 'sRGB', chromaticity_diagram_callable_CIE1960UCS: Callable = plot_RGB_colourspaces_in_chromat scatter_kwargs: dict | None = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *RGB* colourspace array in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

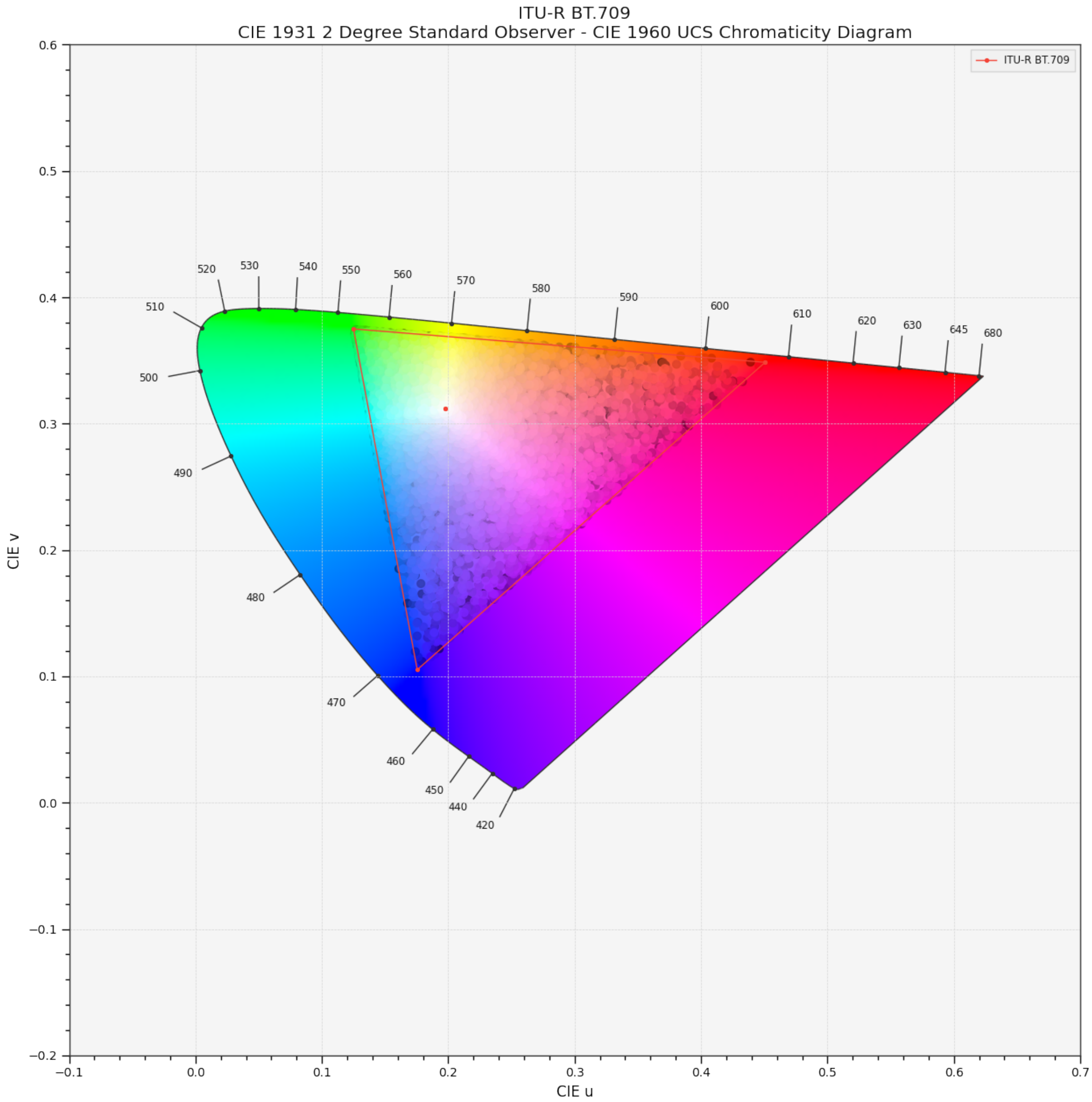
- **RGB** (ArrayLike) – *RGB* colourspace array.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.common.filter_RGB_colourspaces() definition.
- **chromaticity_diagram_callable_CIE1960UCS** (Callable) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **scatter_kwargs** (dict | None) – Keyword arguments for the matplotlib.pyplot.scatter() definition. The following special keyword arguments can also be used:
  - c : If c is set to *RGB*, the scatter will use the colours as given by the RGB argument.
  - apply_cctf_encoding : If apply_cctf_encoding is set to *False*, the encoding colour component transfer function / opto-electronic transfer function is not applied when encoding the samples to the plotting space.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS(
...     RGB, "ITU-R BT.709"
... )
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS**(*RGB: ArrayLike, colourspace: colour.models.rgb.rgb_colourspace | str | collections.abc.Sequence[colour.models. | str] = 'sRGB', chromaticity_diagram_callable_CIE1976UCS: Callable = plot_RGB_colourspaces_in_chromat scatter_kwargs: dict | None = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *RGB* colourspace array in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

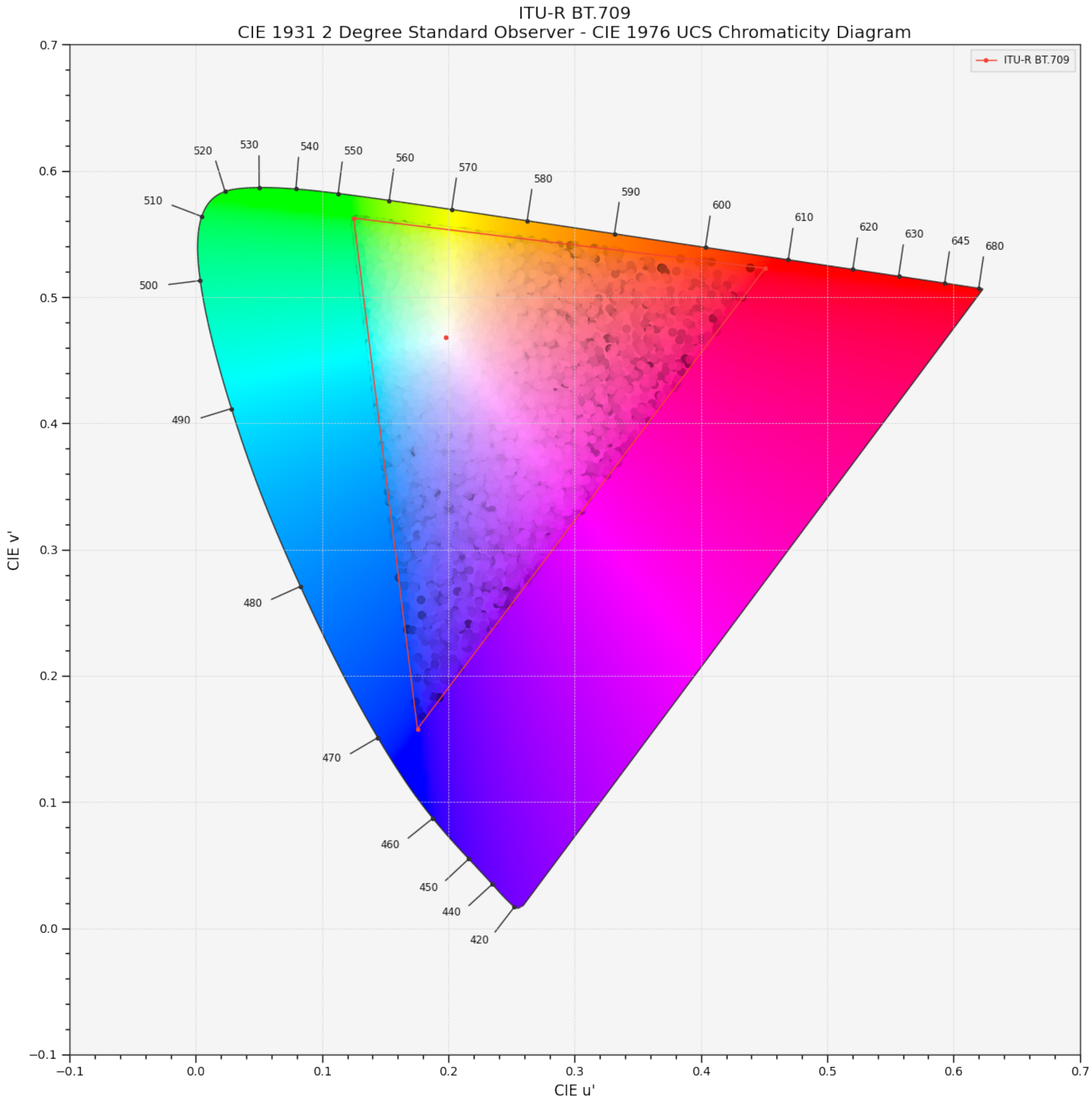
- **RGB** (ArrayLike) – *RGB* colourspace array.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.common.filter_RGB_colourspaces() definition.
- **chromaticity_diagram_callable_CIE1976UCS** (Callable) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **scatter_kwargs** (dict | None) – Keyword arguments for the matplotlib.pyplot.scatter() definition. The following special keyword arguments can also be used:
  - c : If c is set to *RGB*, the scatter will use the colours as given by the RGB argument.
  - apply_cctf_encoding : If apply_cctf_encoding is set to *False*, the encoding colour component transfer function / opto-electronic transfer function is not applied when encoding the samples to the plotting space.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS(
...     RGB, "ITU-R BT.709"
... )
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931**(*chromaticity_diagram_callable_CIE1931: Callable = plot_chromaticity_diagram_CIE1931, chromaticity_diagram_clipping: bool = False, ellipse_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1931 Chromaticity Diagram*.

Parameters

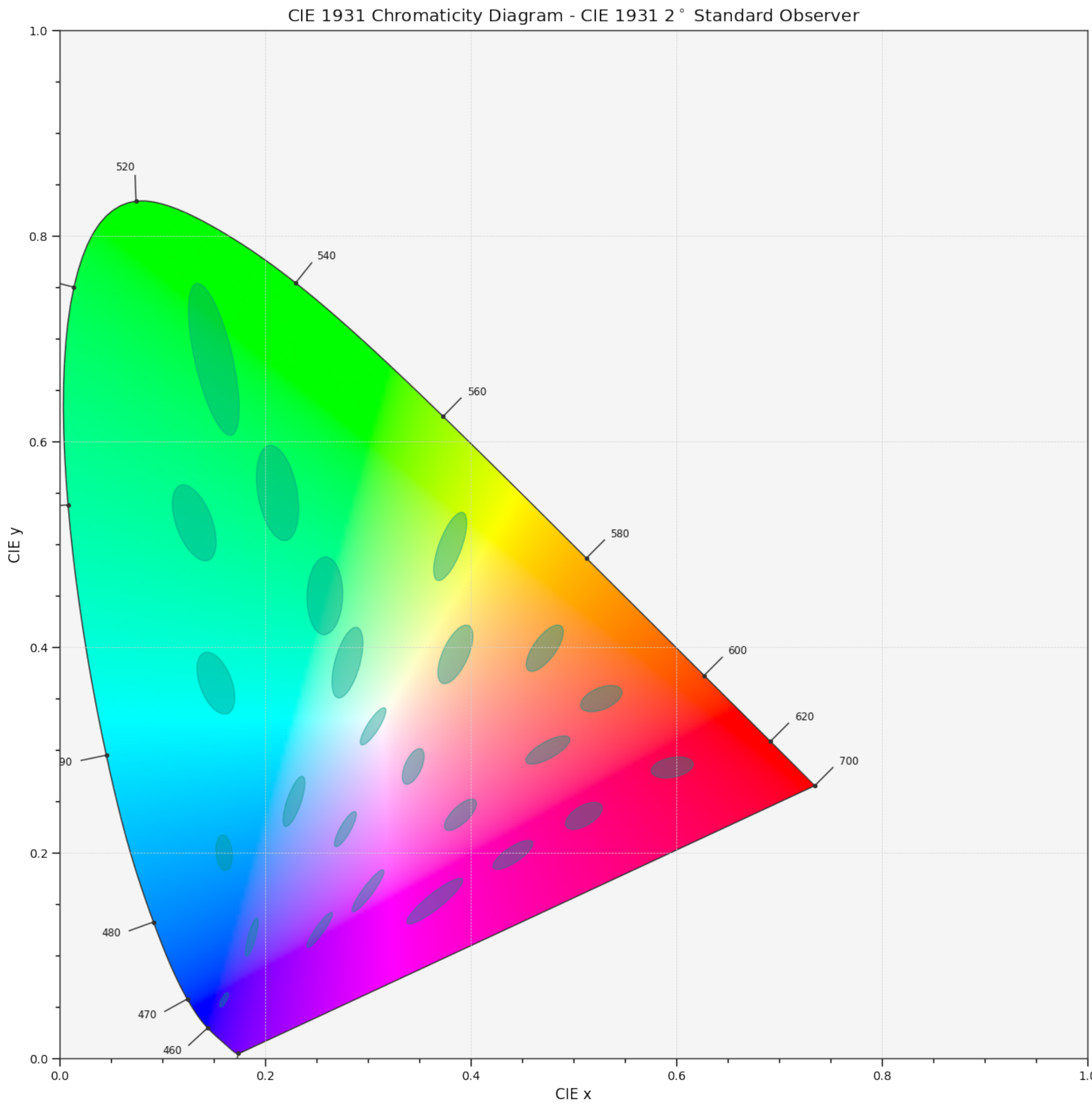
- **chromaticity_diagram_callable_CIE1931** (Callable) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **chromaticity_diagram_clipping** (bool) – Whether to clip the *CIE 1931 Chromaticity Diagram* colours with the ellipses.
- **ellipse_kwargs** (Optional[Union[dict, List[dict]]]) – Parameters for the Ellipse class, ellipse_kwargs can be either a single dictionary applied to all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()}, colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931()
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS**(*chromaticity_diagram_callable_C Callable = plot_chromaticity_diagram_CIE1 chromaticity_diagram_clipping: bool = False, ellipse_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

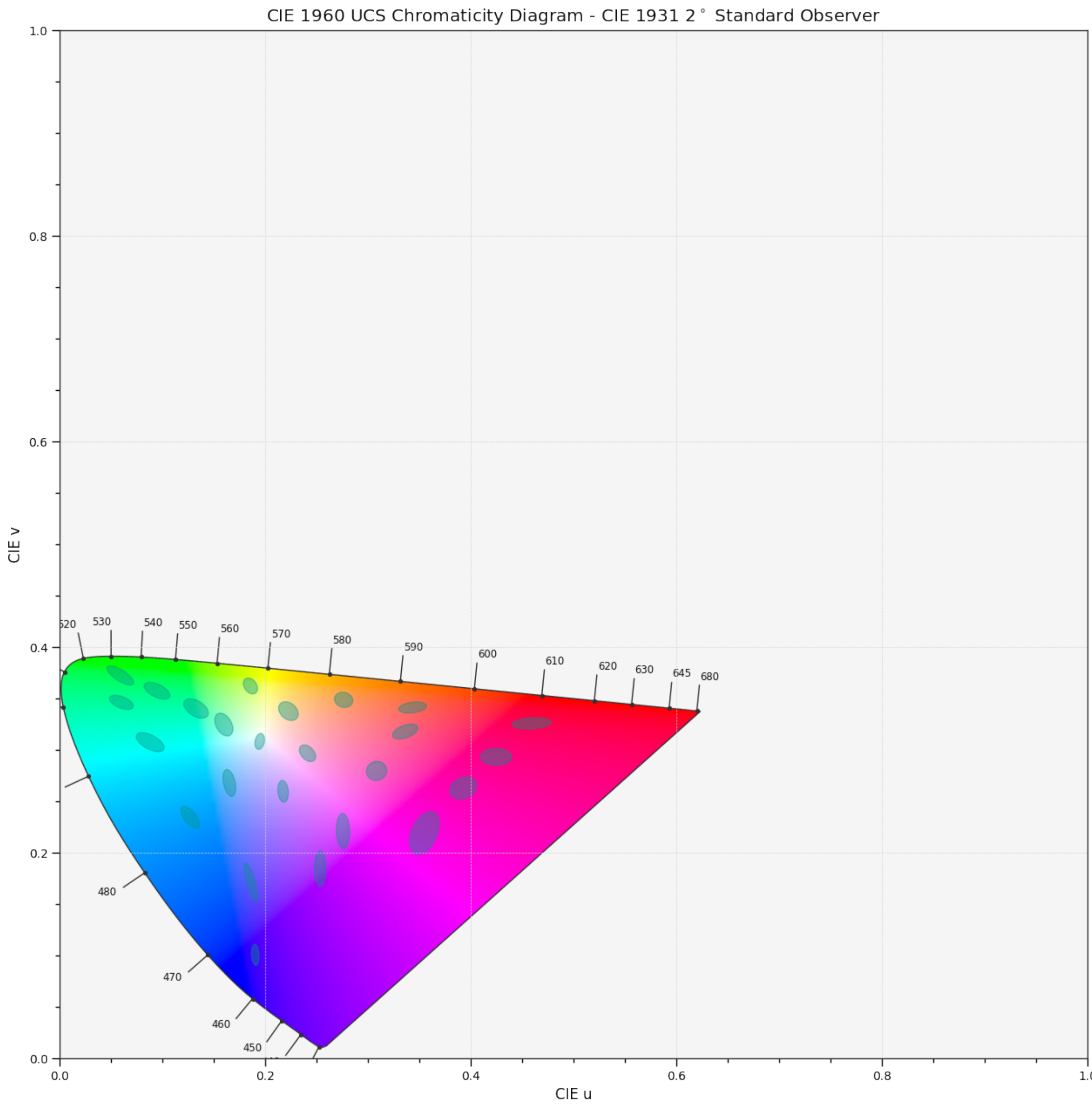
- **chromaticity_diagram_callable_CIE1960UCS** (Callable) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **chromaticity_diagram_clipping** (bool) – Whether to clip the *CIE 1960 UCS Chromaticity Diagram* colours with the ellipses.
- **ellipse_kwargs** (Optional[Union[dict, List[dict]]]) – Parameters for the Ellipse class, ellipse_kwargs can be either a single dictionary applied to all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()}, colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS()
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS**(*chromaticity_diagram_callable_C Callable = plot_chromaticity_diagram_CIE1 chromaticity_diagram_clipping: bool = False, ellipse_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

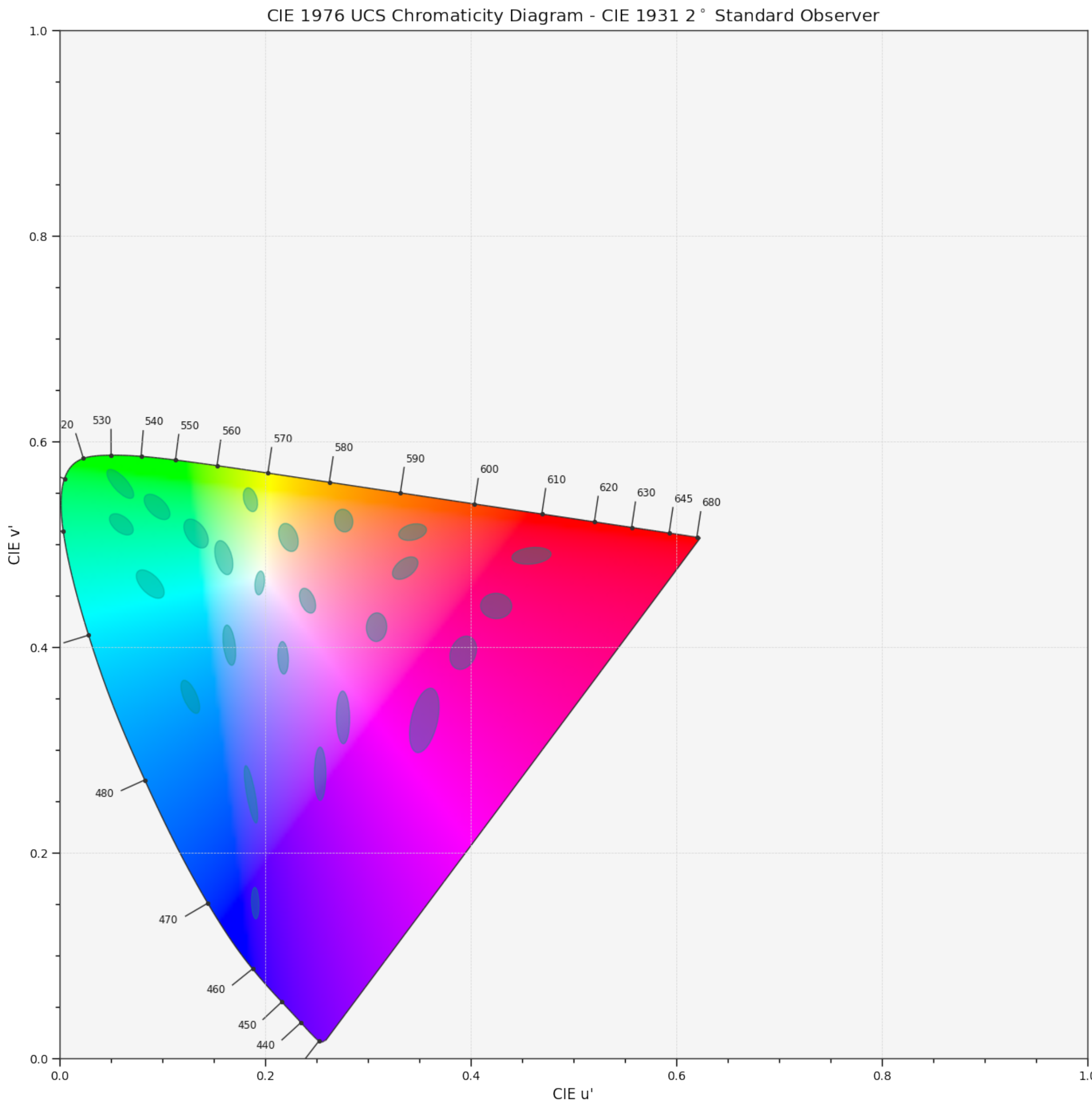
- **chromaticity_diagram_callable_CIE1976UCS** (Callable) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **chromaticity_diagram_clipping** (bool) – Whether to clip the *CIE 1976 UCS Chromaticity Diagram* colours with the ellipses.
- **ellipse_kwargs** (Optional[Union[dict, List[dict]]]) – Parameters for the Ellipse class, ellipse_kwargs can be either a single dictionary applied to all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()}, colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS()
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_single_cctf

colour.plotting.**plot_single_cctf**(*cctf: Union[Callable, str]*, *cctf_decoding: bool = False*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given colourspace colour component transfer function.

Parameters

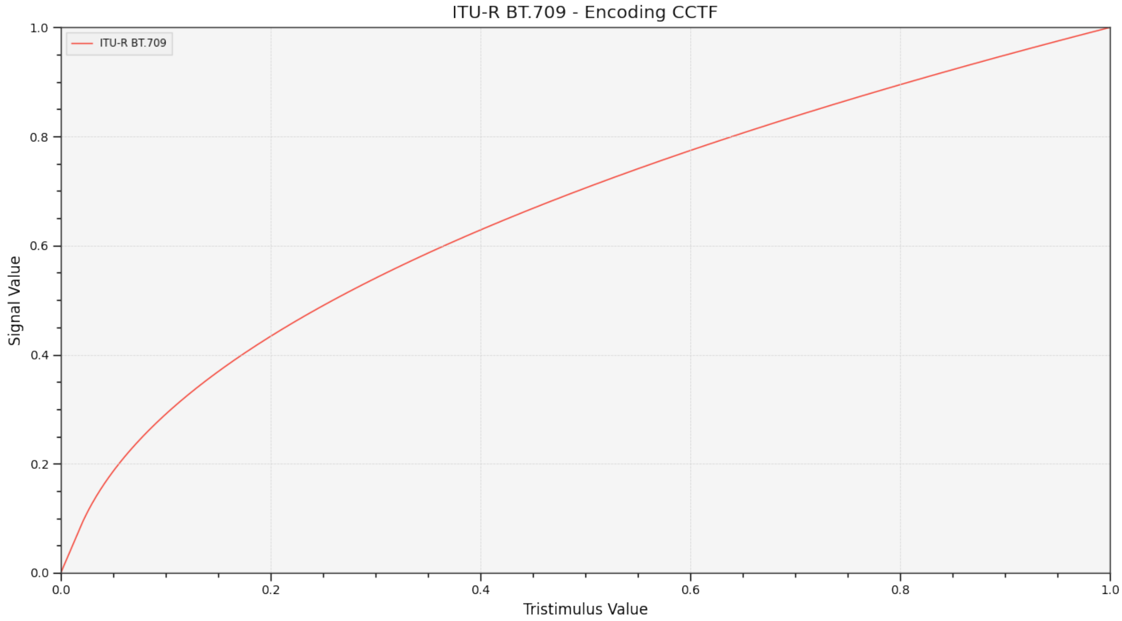
- **cctf** (`Union[Callable, str]`) – Colour component transfer function to plot. `function` can be of any type or form supported by the `colour.plotting.common.filter_passthrough()` definition.
- **cctf_decoding** (`bool`) – Plot the decoding colour component transfer function instead.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_cctf("ITU-R BT.709")
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_multi_cctfs

colour.plotting.**plot_multi_cctfs**(*cctfs: Union[Callable, str, collections.abc.Sequence[Union[Callable, str]]], cctf_decoding: bool = False, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given colour component transfer functions.

Parameters

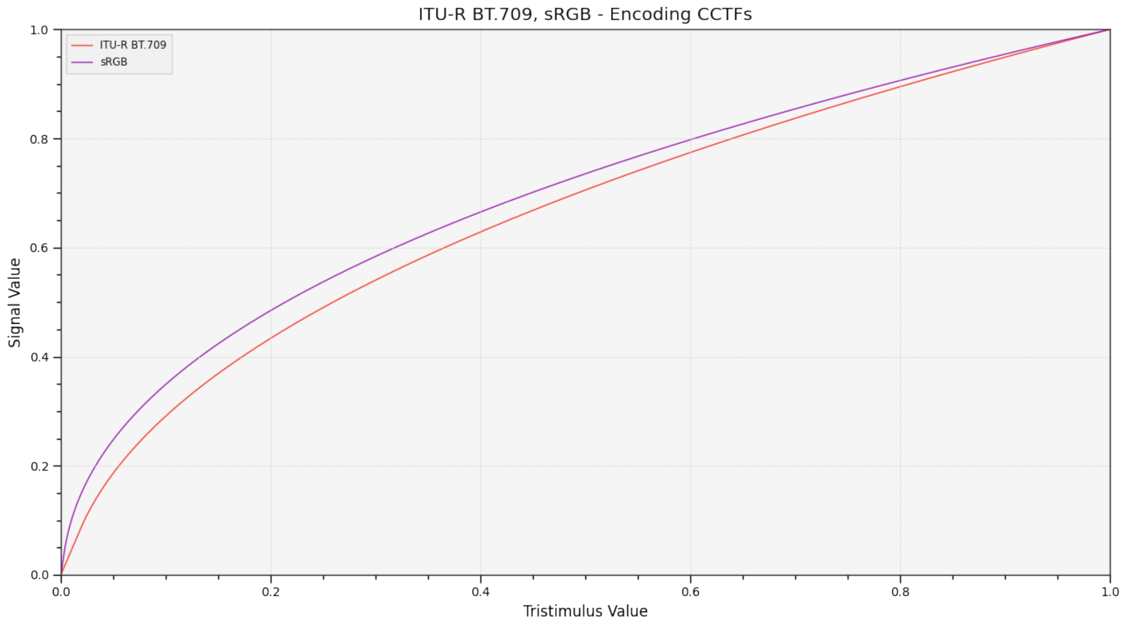
- **cctfs** (Union[Callable, str, collections.abc.Sequence[Union[Callable, str]]]) – Colour component transfer function to plot. cctfs elements can be of any type or form supported by the colour.plotting.common.filter_passthrough() definition.
- **cctf_decoding** (bool) – Plot the decoding colour component transfer function instead.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_cctfs(["ITU-R BT.709", "sRGB"])
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_constant_hue_loci

colour.plotting.**plot_constant_hue_loci**(*data: ArrayLike*, *model: Union[Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IPT Ragoo 2021', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'], str] = 'CIE Lab'*, *scatter_kwargs: dict | None = None*, *convert_kwargs: dict | None = None*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given constant hue loci colour matches data such as that from [HB95] or [EF98] that are easily loaded with Colour - Datasets.

Parameters

- **data** (ArrayLike) – Constant hue loci colour matches data expected to be an *ArrayLike* as follows:

```
[
    ('name', XYZ_r, XYZ_cr, (XYZ_ct, XYZ_ct, XYZ_ct, ...),
↪{metadata}),
    ('name', XYZ_r, XYZ_cr, (XYZ_ct, XYZ_ct, XYZ_ct, ...),
↪{metadata}),
    ('name', XYZ_r, XYZ_cr, (XYZ_ct, XYZ_ct, XYZ_ct, ...),
↪{metadata}),
    ...
]
```

where `name` is the hue angle or name, `XYZ_r` the *CIE XYZ* tristimulus values of the reference illuminant, `XYZ_cr` the *CIE XYZ* tristimulus values of the reference colour under the reference illuminant, `XYZ_ct` the *CIE XYZ* tristimulus values of the colour matches under the reference illuminant and `metadata` the dataset metadata.

- **model** (`Union[Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IPT Ragoo 2021', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'], str]`) – Colourspace model, see `colour.COLOURSPACE_MODELS` attribute for the list of supported colourspace models.
- **scatter_kwargs** (`dict | None`) – Keyword arguments for the `matplotlib.pyplot.scatter()` definition. The following special keyword arguments can also be used:
  - `c` : If `c` is set to *RGB*, the scatter will use the colours as given by the `RGB` argument.
- **convert_kwargs** (`dict | None`) – Keyword arguments for the `colour.convert()` definition.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

References

[EF98], [HB95], [Man19]

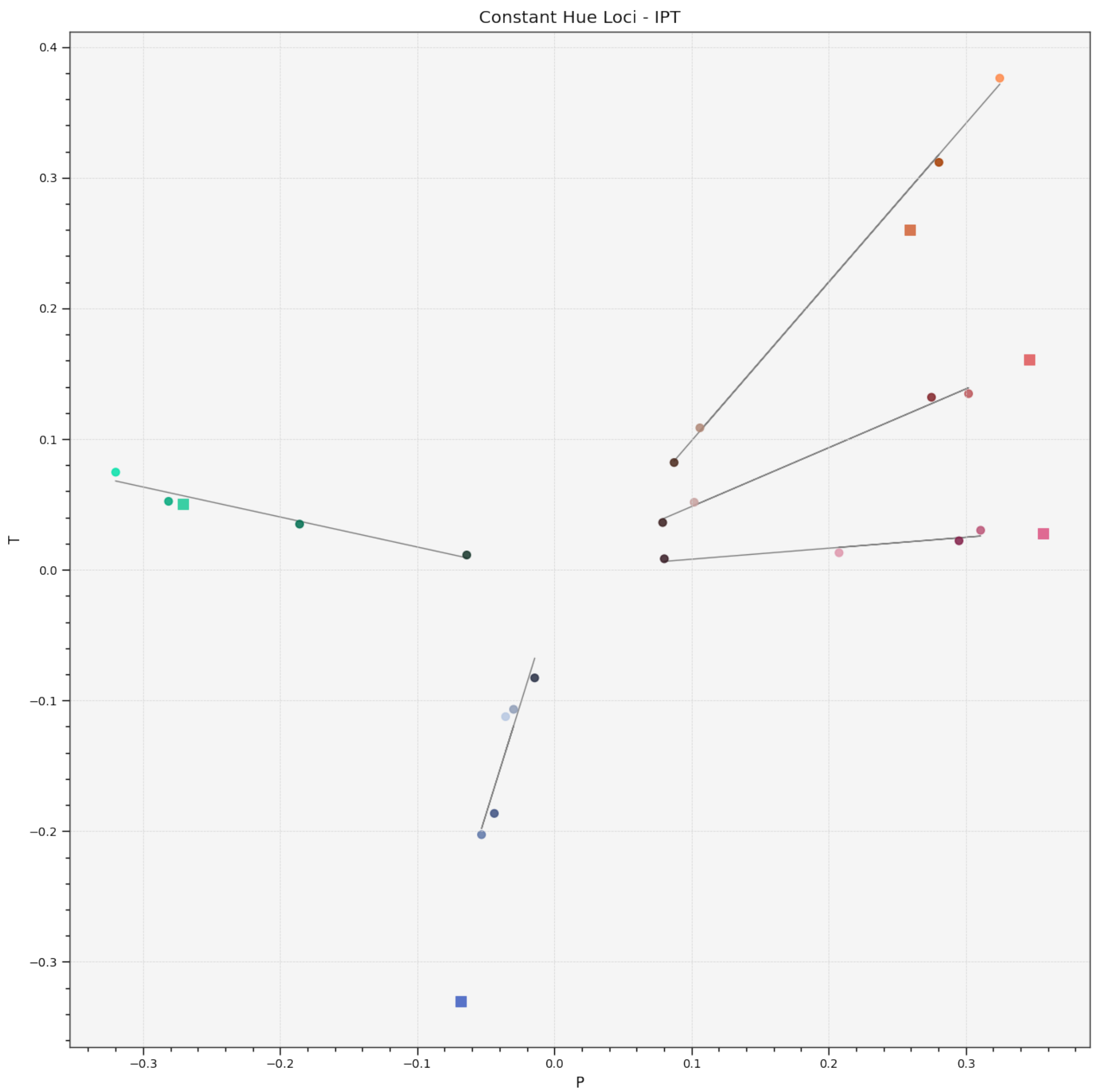
Examples

```
>>> data = [
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.40920000, 0.28120000, 0.30600000]),
...         np.array(
...             [
...                 [0.02495100, 0.01908600, 0.02032900],
...                 [0.10944300, 0.06235900, 0.06788100],
...                 [0.27186500, 0.18418700, 0.19565300],
...                 [0.48898900, 0.40749400, 0.44854600],
...             ]
...         ),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.30760000, 0.48280000, 0.42770000]),
...         np.array(
...             [
...                 [0.02108000, 0.02989100, 0.02790400],
...                 [0.06194700, 0.11251000, 0.09334400],
...                 [0.15255800, 0.28123300, 0.23234900],
...                 [0.34157700, 0.56681300, 0.47035300],
```

(continues on next page)

(continued from previous page)

```
...             ]
...         ),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.39530000, 0.28120000, 0.18450000]),
...         np.array(
...             [
...                 [0.02436400, 0.01908600, 0.01468800],
...                 [0.10331200, 0.06235900, 0.02854600],
...                 [0.26311900, 0.18418700, 0.12109700],
...                 [0.43158700, 0.40749400, 0.39008600],
...             ]
...         ),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.20510000, 0.18420000, 0.57130000]),
...         np.array(
...             [
...                 [0.03039800, 0.02989100, 0.06123300],
...                 [0.08870000, 0.08498400, 0.21843500],
...                 [0.18405800, 0.18418700, 0.40111400],
...                 [0.32550100, 0.34047200, 0.50296900],
...                 [0.53826100, 0.56681300, 0.80010400],
...             ]
...         ),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.35770000, 0.28120000, 0.11250000]),
...         np.array(
...             [
...                 [0.03678100, 0.02989100, 0.01481100],
...                 [0.17127700, 0.11251000, 0.01229900],
...                 [0.30080900, 0.28123300, 0.21229800],
...                 [0.52976000, 0.40749400, 0.11720000],
...             ]
...         ),
...         None,
...     ],
... ]
>>> plot_constant_hue_loci(data, "CIE Lab")
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Ancillary Objects

`colour.plotting.models`

colourspace_model_axis_reorder(a, model[, ...])	Reorder the axes of given colourspace model a array according to the most common volume plotting axes order.
plot_pointer_gamut([pointer_gamut_colours, ...])	Plot *Pointer's Gamut* according to given method.
plot_RGB_colourspaces_in_chromaticity_diagram(...)	Plot given *RGB* colourspaces in the *Chromaticity Diagram* according to given method.
plot_RGB_chromaticities_in_chromaticity_diagram(RGB)	Plot given *RGB* colourspace array in the *Chromaticity Diagram* according to given method.
plot_ellipses_MacAdam1942_in_chromaticity_diagram(...)	Plot *MacAdam (1942) Ellipses (Observer PGN)* in the *Chromaticity Diagram* according to given method.

colour.plotting.models.colourspace_model_axis_reorder

colour.plotting.models.**colourspace_model_axis_reorder**(*a: ArrayLike*, *model: Union[Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE LCHab', 'CIE Luv', 'CIE LCHuv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IPT Ragoo 2021', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT', 'Yrg'], str]*, *direction: Union[Literal['Forward', 'Inverse'], str] = 'Forward'*) → NDArrayFloat

Reorder the axes of given colourspace model a array according to the most common volume plotting axes order.

Parameters

- **a** (ArrayLike) – Colourspace model a array.
- **model** (Union[Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE LCHab', 'CIE Luv', 'CIE LCHuv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IPT Ragoo 2021', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT', 'Yrg'], str]) – Colourspace model, see colour.COLOURSPACE_MODELS attribute for the list of supported colourspace models.
- **direction** (Union[Literal['Forward', 'Inverse'], str]) – Reordering direction.

Returns Reordered colourspace model a array.

Return type numpy.ndarray

Examples

```
>>> a = np.array([0, 1, 2])
>>> colourspace_model_axis_reorder(a, "CIE Lab")
array([ 1.,  2.,  0.])
>>> colourspace_model_axis_reorder(a, "IPT")
array([ 1.,  2.,  0.])
>>> colourspace_model_axis_reorder(a, "OSA UCS")
array([ 1.,  2.,  0.])
>>> b = np.array([1, 2, 0])
>>> colourspace_model_axis_reorder(b, "OSA UCS", "Inverse")
array([ 0.,  1.,  2.])
```

colour.plotting.models.plot_pointer_gamut

colour.plotting.models.**plot_pointer_gamut**(*pointer_gamut_colours: ArrayLike | str | None = None, pointer_gamut_opacity: float = 1, method: Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str = 'CIE 1931', **kwargs: Any*) → Tuple[plt.Figure, plt.Axes]

Plot *Pointer's Gamut* according to given method.

Parameters

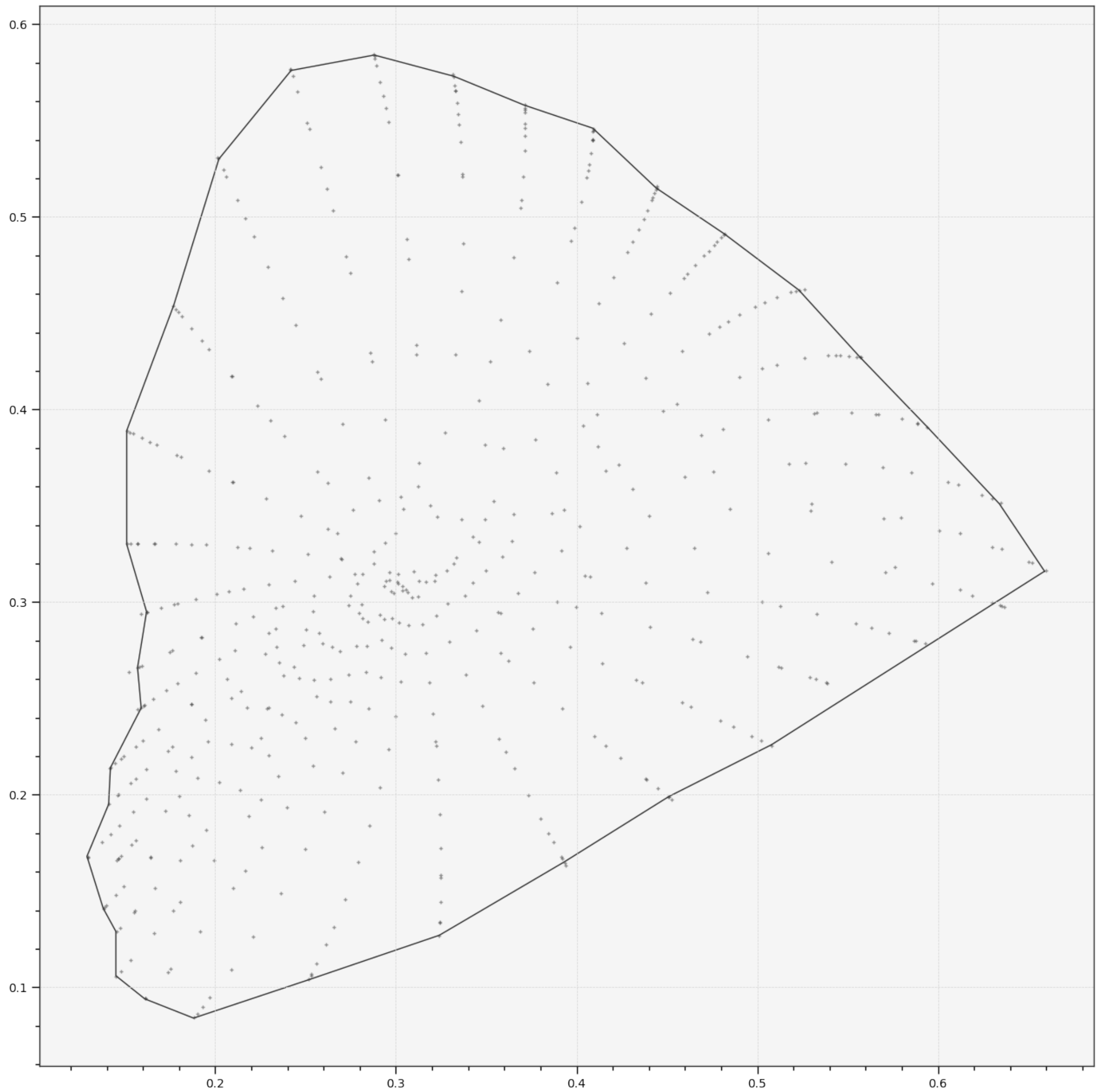
- **pointer_gamut_colours** (ArrayLike | str | None) – Colours of the *Pointer's Gamut*.
- **pointer_gamut_opacity** (float) – Opacity of the *Pointer's Gamut*.
- **method** (Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str) – Plotting method.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_pointer_gamut()
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram

colour.plotting.models.**plot_RGB_colourspaces_in_chromaticity_diagram**(*colourspaces: colour.models.rgb.rgb_colourspace.RGB_C | str | collections.abc.Sequence[colour.models.rgb.rgb | str], cmfs: colour.colorimetry.spectrum.MultiSpectra | str | collections.abc.Sequence[colour.colorimetry.spe | str] = 'CIE 1931 2 Degree Standard Observer', chromaticity_diagram_callable: Callable = plot_chromaticity_diagram, method: Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str] = 'CIE 1931', show_whitepoints: bool = True, show_pointer_gamut: bool = False, chromatically_adapt: bool = False, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *RGB* colourspaces in the *Chromaticity Diagram* according to given method.

Parameters

- **colourspaces** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace` | `str` | `collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]`) – *RGB* colourspaces to plot. `colourspaces` elements can be of any type or form supported by the `colour.plotting.common.filter_RGB_colourspaces()` definition.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions` | `str` | `collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]`) – Standard observer colour matching functions used for computing the spectral locus boundaries. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **chromaticity_diagram_callable** (`Callable`) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (`Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str]`) – *Chromaticity Diagram* method.
- **show_whitepoints** (`bool`) – Whether to display the *RGB* colourspaces whitepoints.
- **show_pointer_gamut** (`bool`) – Whether to display the *Pointer's Gamut*.
- **chromatically_adapt** (`bool`) – Whether to chromatically adapt the *RGB* colourspaces given in `colourspaces` to the whitepoint of the default plotting colourspace.

- **plot_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for the matplotlib.pyplot.plot() definition, used to control the style of the plotted *RGB* colourspaces. plot_kwargs can be either a single dictionary applied to all the plotted *RGB* colourspaces with the same settings or a sequence of dictionaries with different settings for each plotted *RGB* colourspace.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_pointer_gamut(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_kwargs = [
...     {"color": "r"},
...     {"linestyle": "dashed"},
...     {"marker": None},
... ]
>>> plot_RGB_colourspaces_in_chromaticity_diagram(
...     ["ITU-R BT.709", "ACEScg", "S-Gamut"], plot_kwargs=plot_kwargs
... )
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```

colour.plotting.models.plot_RGB_chromaticities_in_chromaticity_diagram

colour.plotting.models.**plot_RGB_chromaticities_in_chromaticity_diagram**(*RGB: ArrayLike, colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str] = 'sRGB', chromaticity_diagram_callable: Callable = plot_RGB_colourspaces_in_chromaticity_diagram, method: Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str] = 'CIE 1931', scatter_kwargs: dict | None = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *RGB* colourspace array in the *Chromaticity Diagram* according to given method.

Parameters

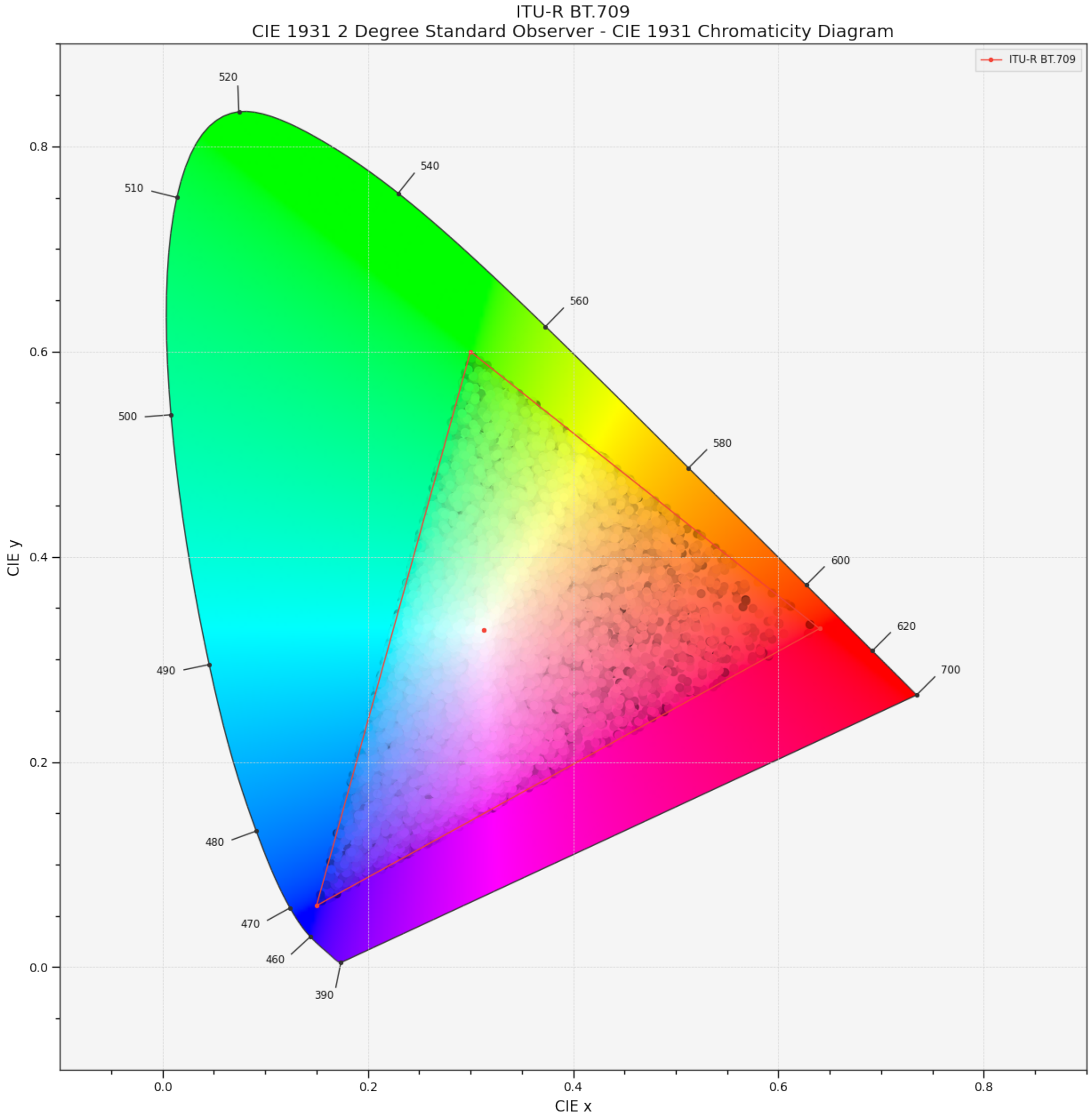
- **RGB** (ArrayLike) – *RGB* colourspace array.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.common.filter_RGB_colourspaces() definition.
- **chromaticity_diagram_callable** (Callable) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str]) – *Chromaticity Diagram* method.
- **scatter_kwargs** (dict | None) – Keyword arguments for the matplotlib.pyplot.scatter() definition. The following special keyword arguments can also be used:
  - c : If c is set to *RGB*, the scatter will use the colours as given by the RGB argument.
  - apply_cctf_encoding : If apply_cctf_encoding is set to *False*, the encoding colour component transfer function / opto-electronic transfer function is not applied when encoding the samples to the plotting space.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram(RGB, "ITU-R BT.709")
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram

colour.plotting.models.**plot_ellipses_MacAdam1942_in_chromaticity_diagram**(*chromaticity_diagram_callable: Callable = plot_chromaticity_diagram, method: Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str] = 'CIE 1931', chromaticity_diagram_clipping: bool = False, ellipse_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot *MacAdam (1942) Ellipses (Observer PGN)* in the *Chromaticity Diagram* according to given method.

Parameters

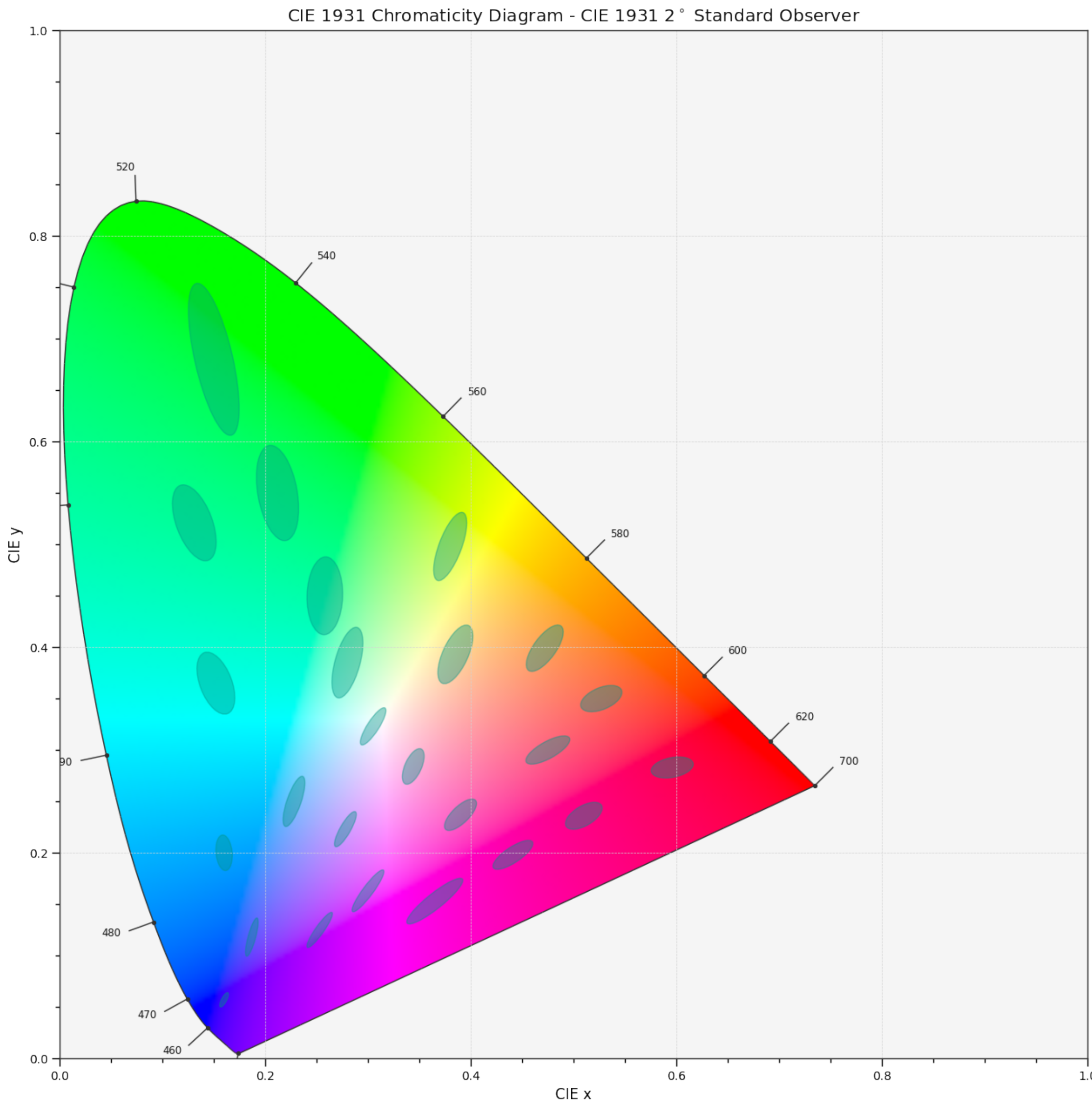
- **chromaticity_diagram_callable** (`Callable`) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (`Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str]`) – *Chromaticity Diagram* method.
- **chromaticity_diagram_clipping** (`bool`) – Whether to clip the *Chromaticity Diagram* colours with the ellipses.
- **ellipse_kwargs** (`Optional[Union[dict, List[dict]]]`) – Parameters for the `Ellipse` class, `ellipse_kwargs` can be either a single dictionary applied to all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram()
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Colour Notation Systems

`colour.plotting`

`plot_single_munsell_value_function`(function, ...)	Plot given *Lightness* function.
`plot_multi_munsell_value_functions`(...)	Plot given *Munsell* value functions.

colour.plotting.plot_single_munsell_value_function

`colour.plotting.`**`plot_single_munsell_value_function`**(*function: Union[Callable, str], **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *Lightness* function.

Parameters

- **function** (`Union[Callable, str]`) – *Munsell* value function to plot. `function` can be of any type or form supported by the `colour.plotting.common.filter_passthrough()` definition.

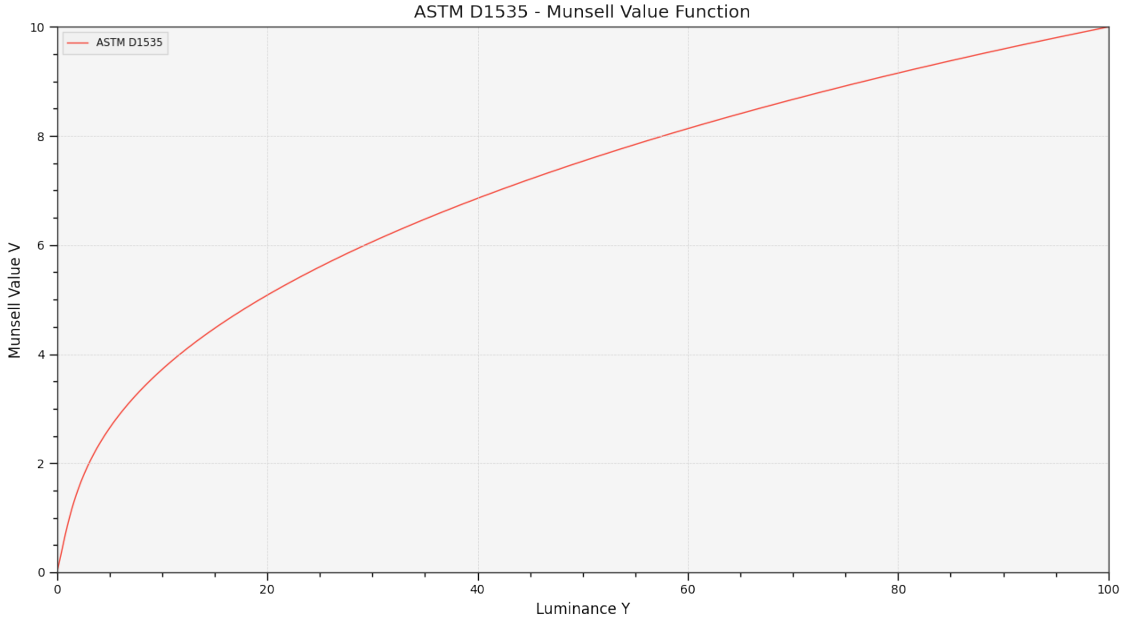
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_munsell_value_function("ASTM D1535")
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_multi_munsell_value_functions

colour.plotting.**plot_multi_munsell_value_functions**(*functions: Union[Callable, str, collections.abc.Sequence[Union[Callable, str]]], **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *Munsell* value functions.

Parameters

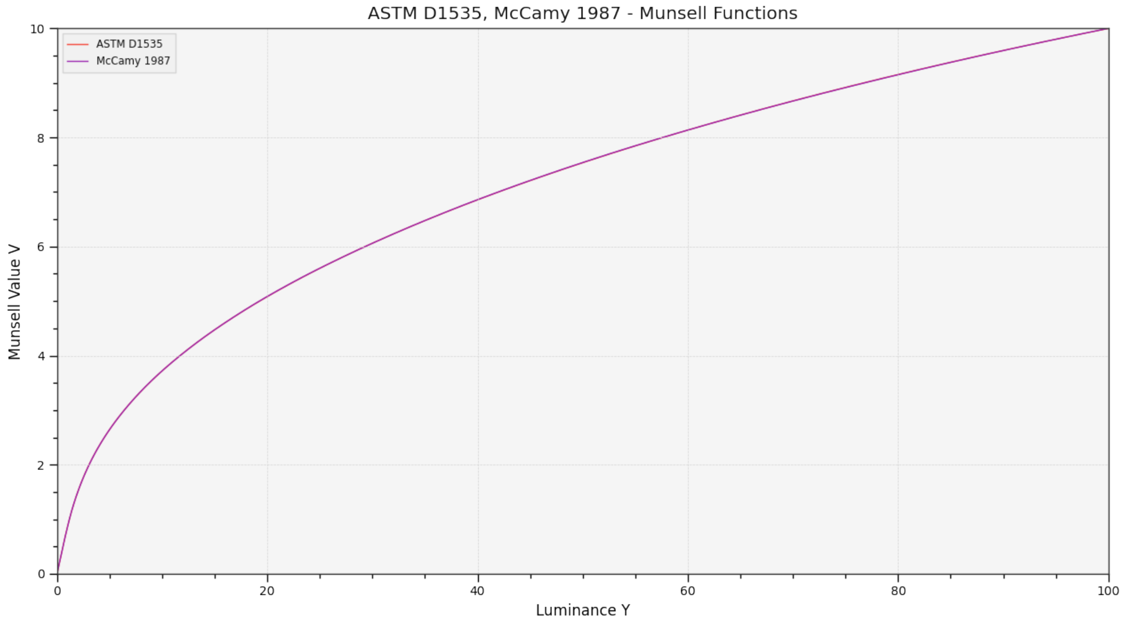
- **functions** (Union[Callable, str, collections.abc.Sequence[Union[Callable, str]]]) – *Munsell* value functions to plot. functions elements can be of any type or form supported by the colour.plotting.common.filter_passthrough() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_munsell_value_functions(["ASTM D1535", "McCamy 1987"])
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Optical Phenomena

`colour.plotting`

`plot_single_sd_rayleigh_scattering`([...])	Plot a single *Rayleigh* scattering spectral distribution.
`plot_the_blue_sky`([cmfs])	Plot the blue sky.

colour.plotting.plot_single_sd_rayleigh_scattering

`colour.plotting.`**`plot_single_sd_rayleigh_scattering`**(*CO2_concentration: ArrayLike = CONSTANT_STANDARD_CO2_CONCENTRATION, temperature: ArrayLike = CONSTANT_STANDARD_AIR_TEMPERATURE, pressure: ArrayLike = CONSTANT_AVERAGE_PRESSURE_MEAN_SEA_LEVEL, latitude: ArrayLike = CONSTANT_DEFAULT_LATITUDE, altitude: ArrayLike = CONSTANT_DEFAULT_ALTITUDE, cmfs:* colour.colorimetry.spectrum.MultiSpectralDistributions | *str* | collections.abc.Sequence*[*colour.colorimetry.spectrum.MultiSpectral | *str] = 'CIE 1931 2 Degree Standard Observer', **kwargs:* Any) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot a single *Rayleigh* scattering spectral distribution.

Parameters

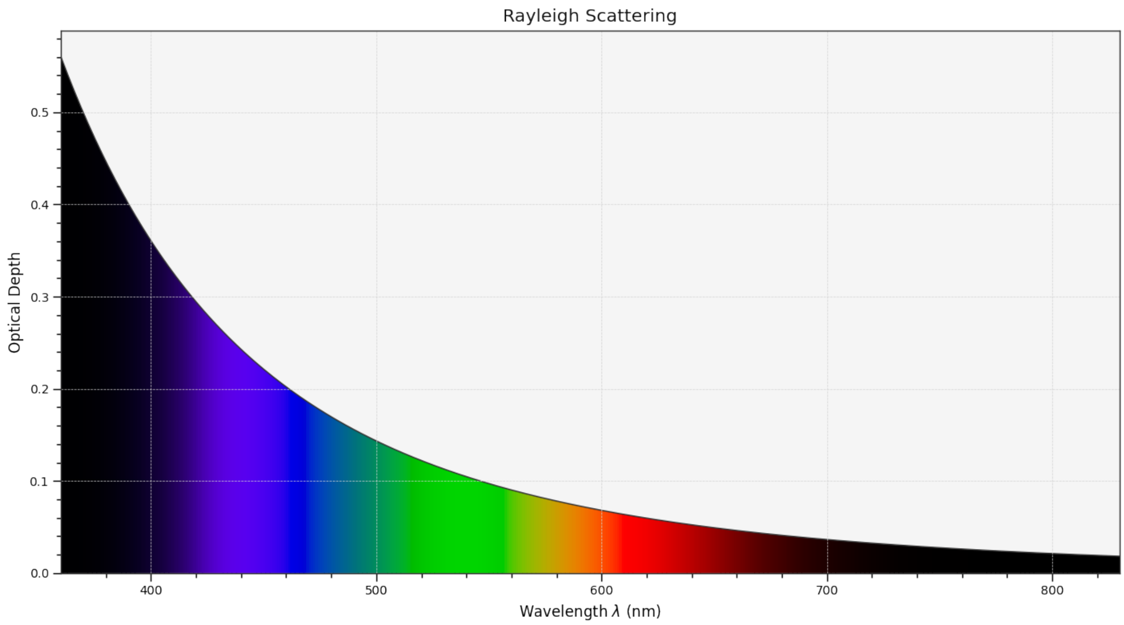
- **CO2_concentration** (ArrayLike) – CO_2 concentration in parts per million (ppm).
- **temperature** (ArrayLike) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (ArrayLike) – Surface pressure P of the measurement site.
- **latitude** (ArrayLike) – Latitude of the site in degrees.
- **altitude** (ArrayLike) – Altitude of the site in meters.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_sd_rayleigh_scattering()
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_the_blue_sky

colour.plotting.**plot_the_blue_sky**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer', **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the blue sky.

Parameters

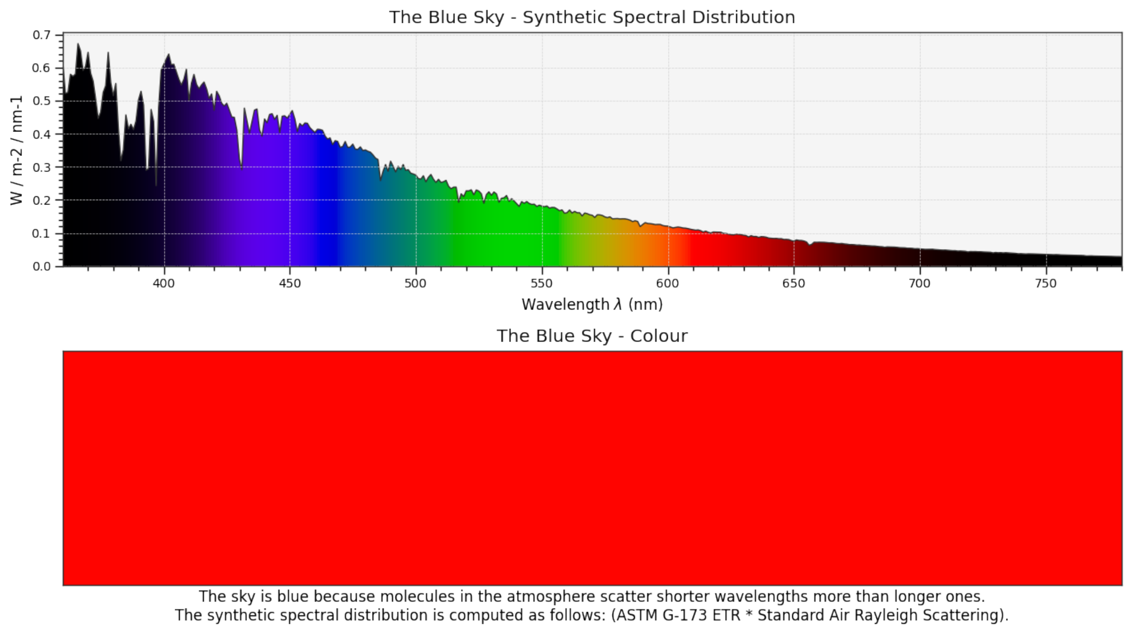
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectrum domain and colours. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **kwargs** (Any) – {`colour.plotting.artist()`, `colour.plotting.plot_single_sd()`, `colour.plotting.plot_multi_colour_swatches()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_the_blue_sky()
(<Figure size ... with 2 Axes>, <...Axes...>)
```



Colour Quality

colour.plotting

plot_single_sd_colour_rendering_index_bars(sd, ...)	Plot the *Colour Rendering Index* (CRI) of given illuminant or light source spectral distribution.
plot_multi_sds_colour_rendering_indexes_bars(...)	Plot the *Colour Rendering Index* (CRI) of given illuminants or light sources spectral distributions.
plot_single_sd_colour_quality_scale_bars(sd)	Plot the *Colour Quality Scale* (CQS) of given illuminant or light source spectral distribution.
plot_multi_sds_colour_quality_scales_bars(sds)	Plot the *Colour Quality Scale* (CQS) of given illuminants or light sources spectral distributions.

colour.plotting.plot_single_sd_colour_rendering_index_bars

colour.plotting.**plot_single_sd_colour_rendering_index_bars**(*sd: colour.colorimetry.spectrum.SpectralDistribution*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *Colour Rendering Index* (CRI) of given illuminant or light source spectral distribution.

Parameters

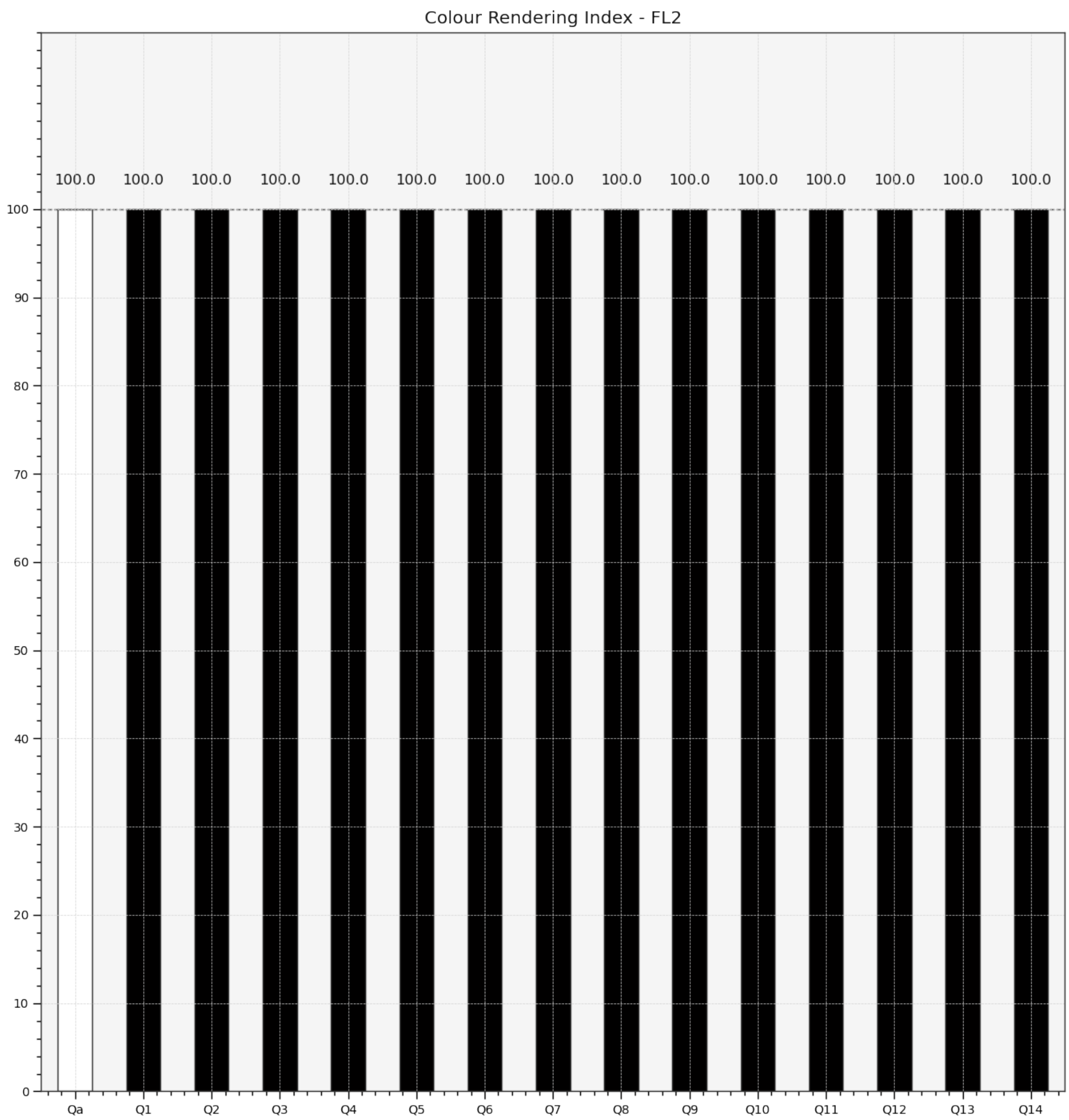
- **sd** (colour.colorimetry.spectrum.SpectralDistribution) – Illuminant or light source spectral distribution to plot the *Colour Rendering Index* (CRI).
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> illuminant = SDS_ILLUMINANTS["FL2"]
>>> plot_single_sd_colour_rendering_index_bars(illuminant)
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_multi_sds_colour_rendering_indexes_bars

`colour.plotting.`**`plot_multi_sds_colour_rendering_indexes_bars`**(*sds:* collections.abc.Sequence[colour.colorimetry.spectrum.Sp | colour.colorimetry.spectrum.MultiSpectralDistribut | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistribut ***kwargs:* Any) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *Colour Rendering Index* (CRI) of given illuminants or light sources spectral distributions.

Parameters

- **sds** (`collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.`

MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a List of colour.SpectralDistribution class instances.

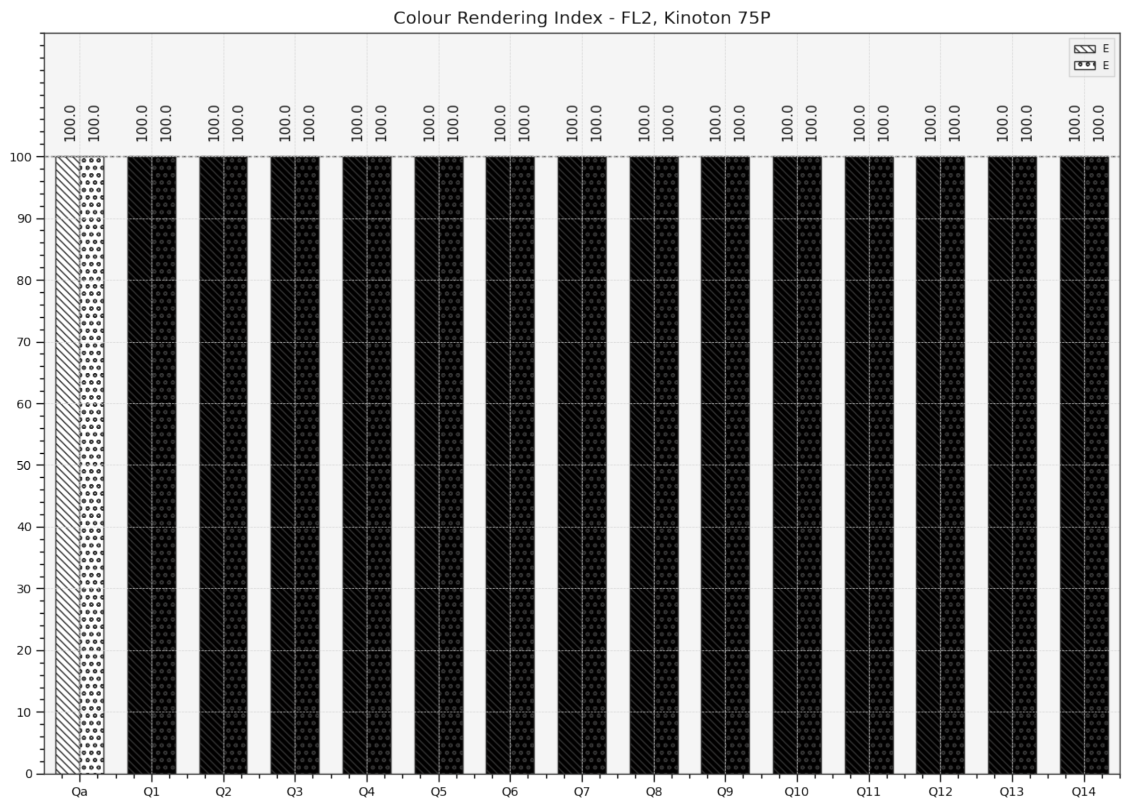
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import SDS_ILLUMINANTS, SDS_LIGHT_SOURCES
>>> illuminant = SDS_ILLUMINANTS["FL2"]
>>> light_source = SDS_LIGHT_SOURCES["Kinoton 75P"]
>>> plot_multi_sds_colour_rendering_indexes_bars(
...     [illuminant, light_source]
... )
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_single_sd_colour_quality_scale_bars

colour.plotting.**plot_single_sd_colour_quality_scale_bars**(*sd: colour.colorimetry.spectrum.SpectralDistribution, method: Union[Literal['NIST CQS 7.4', 'NIST CQS 9.0'], str] = 'NIST CQS 9.0', **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *Colour Quality Scale* (CQS) of given illuminant or light source spectral distribution.

Parameters

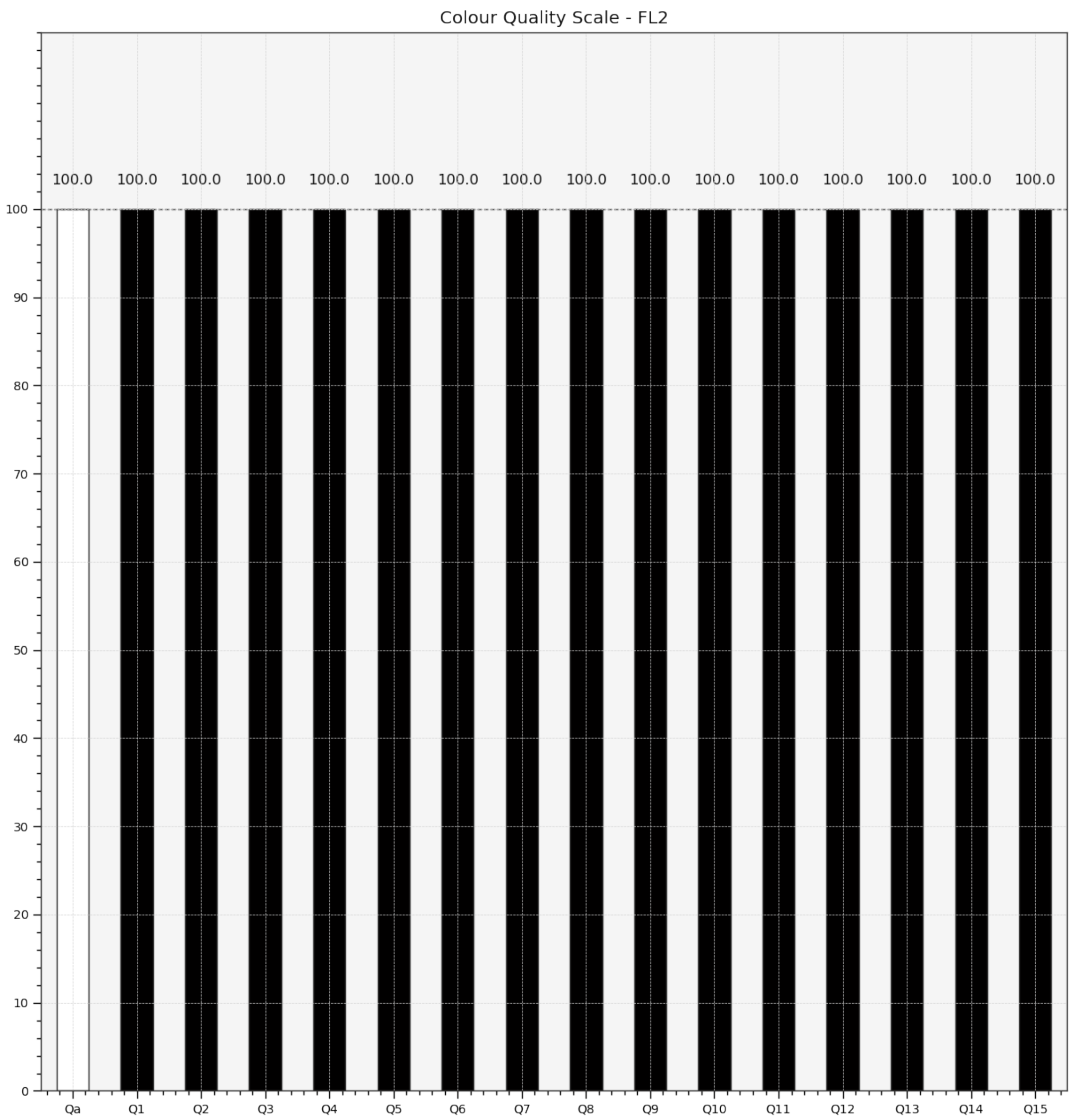
- **sd** (`colour.colorimetry.spectrum.SpectralDistribution`) – Illuminant or light source spectral distribution to plot the *Colour Quality Scale* (CQS).
- **method** (`Union[Literal['NIST CQS 7.4', 'NIST CQS 9.0'], str]`) – *Colour Quality Scale* (CQS) computation method.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.quality.plot_colour_quality_bars()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> illuminant = SDS_ILLUMINANTS["FL2"]
>>> plot_single_sd_colour_quality_scale_bars(illuminant)
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_multi_sds_colour_quality_scales_bars

colour.plotting.**plot_multi_sds_colour_quality_scales_bars**(*sds: collections.abc.Sequence[colour.colorimetry.spectrum.Spect | colour.colorimetry.spectrum.MultiSpectralDistribution | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistribution*, *method: Union[Literal['NIST CQS 7.4', 'NIST CQS 9.0'], str] = 'NIST CQS 9.0'*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *Colour Quality Scale* (CQS) of given illuminants or light sources spectral distributions.

Parameters

- **sds** (`collections.abc.Sequence[colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.`

MultiSpectralDistributions] | colour.colorimetry.spectrum.SpectralDistribution | colour.colorimetry.spectrum.MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a List of colour.SpectralDistribution class instances.

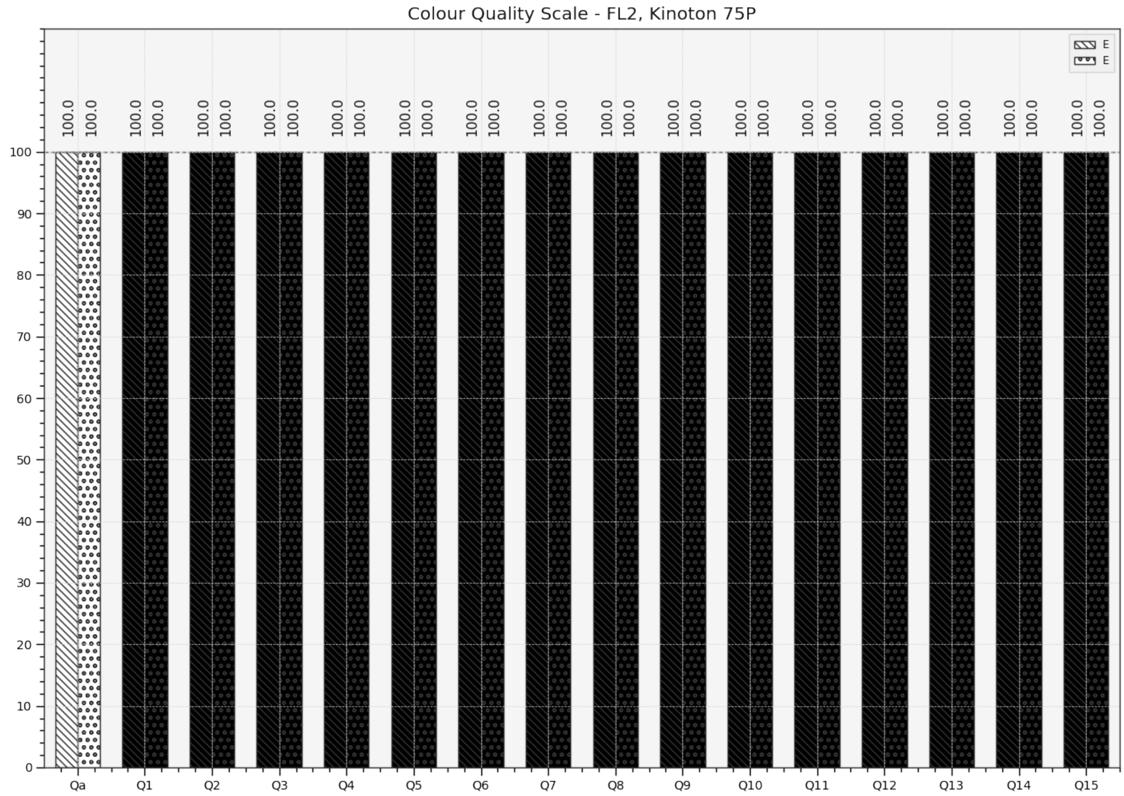
- **method** (Union[Literal['NIST CQS 7.4', 'NIST CQS 9.0'], str]) – *Colour Quality Scale* (CQS) computation method.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import SDS_ILLUMINANTS, SDS_LIGHT_SOURCES
>>> illuminant = SDS_ILLUMINANTS["FL2"]
>>> light_source = SDS_LIGHT_SOURCES["Kinoton 75P"]
>>> plot_multi_sds_colour_quality_scales_bars([illuminant, light_source])
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Ancillary Objects

`colour.plotting.quality`

`plot_colour_quality_bars`(specifications[, ...])	Plot the colour quality data of given illuminants or light sources colour quality specifications.

colour.plotting.quality.plot_colour_quality_bars

`colour.plotting.quality.`**`plot_colour_quality_bars`**(*specifications: collections.abc.Sequence[colour.quality.cqs.ColourRendering_Specifica | colour.quality.cri.ColourRendering_Specification_CRI], labels: bool = True, hatching: bool | None = None, hatching_repeat: int = 2, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the colour quality data of given illuminants or light sources colour quality specifications.

Parameters

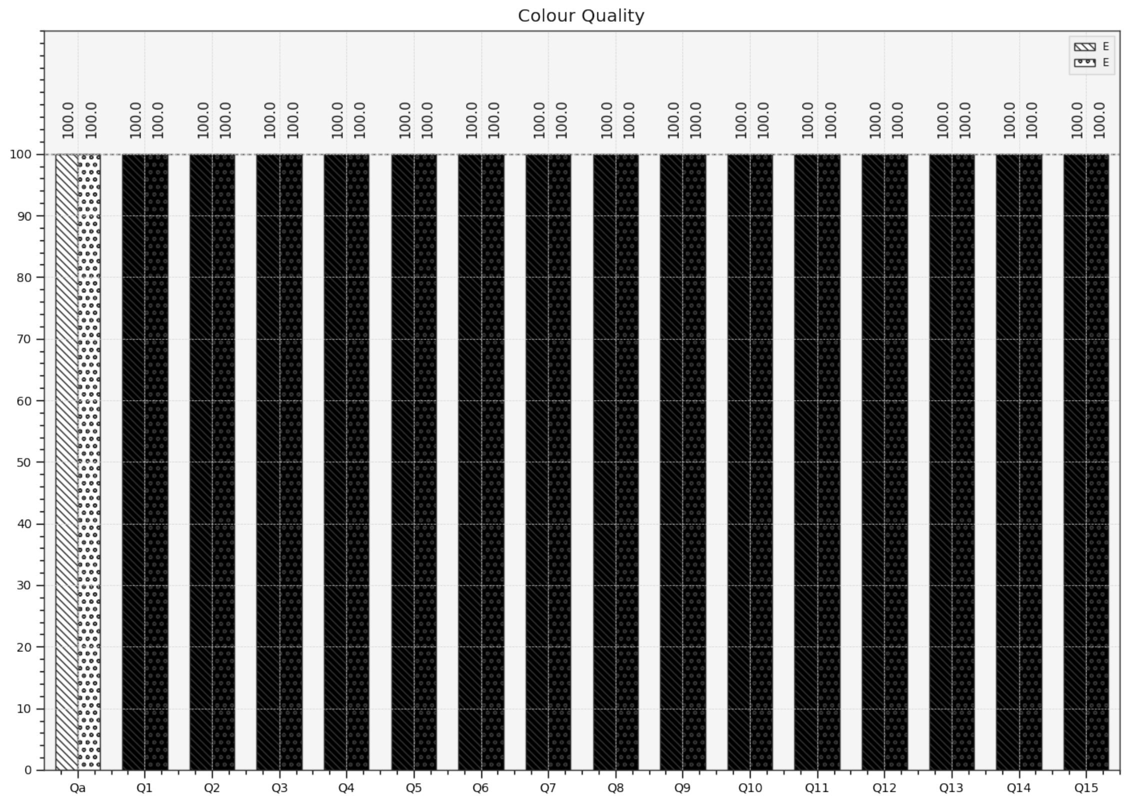
- **specifications** (collections.abc.Sequence[colour.quality.cqs.ColourRendering_Specification_CQS | colour.quality.cri.ColourRendering_Specification_CRI]) – Array of illuminants or light sources colour quality specifications.
- **labels** (bool) – Add labels above bars.
- **hatching** (bool | None) – Use hatching for the bars.
- **hatching_repeat** (int) – Hatching pattern repeat.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import SDS_ILLUMINANTS, SDS_LIGHT_SOURCES, SpectralShape
>>> illuminant = SDS_ILLUMINANTS["FL2"]
>>> light_source = SDS_LIGHT_SOURCES["Kinoton 75P"]
>>> light_source = light_source.copy().align(SpectralShape(360, 830, 1))
>>> cqs_i = colour_quality_scale(illuminant, additional_data=True)
>>> cqs_l = colour_quality_scale(light_source, additional_data=True)
>>> plot_colour_quality_bars([cqs_i, cqs_l])
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Gamut Section Plotting

`colour.plotting`

plot_visible_spectrum_section([cmfs, ...])	Plot the visible spectrum volume, i.e. *Ro?sch-MacAdam* colour solid, section colours along given axis and origin.
plot_RGB_colourspace_section(colourspace[, ...])	Plot given *RGB* colourspace section colours along given axis and origin.

colour.plotting.plot_visible_spectrum_section

colour.plotting.**plot_visible_spectrum_section**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistril | str] = 'CIE 1931 2 Degree Standard Observer', illuminant: colour.colorimetry.spectrum.SpectralDistribution | str = 'D65', model: Union[Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'], str] = 'CIE xyY', axis: Union[Literal['+z', '+x', '+y'], str] = '+z', origin: float = 0.5, normalise: bool = True, show_section_colours: bool = True, show_section_contour: bool = True, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the visible spectrum volume, i.e. *Ro?sch-MacAdam* colour solid, section colours along given axis and origin.

Parameters

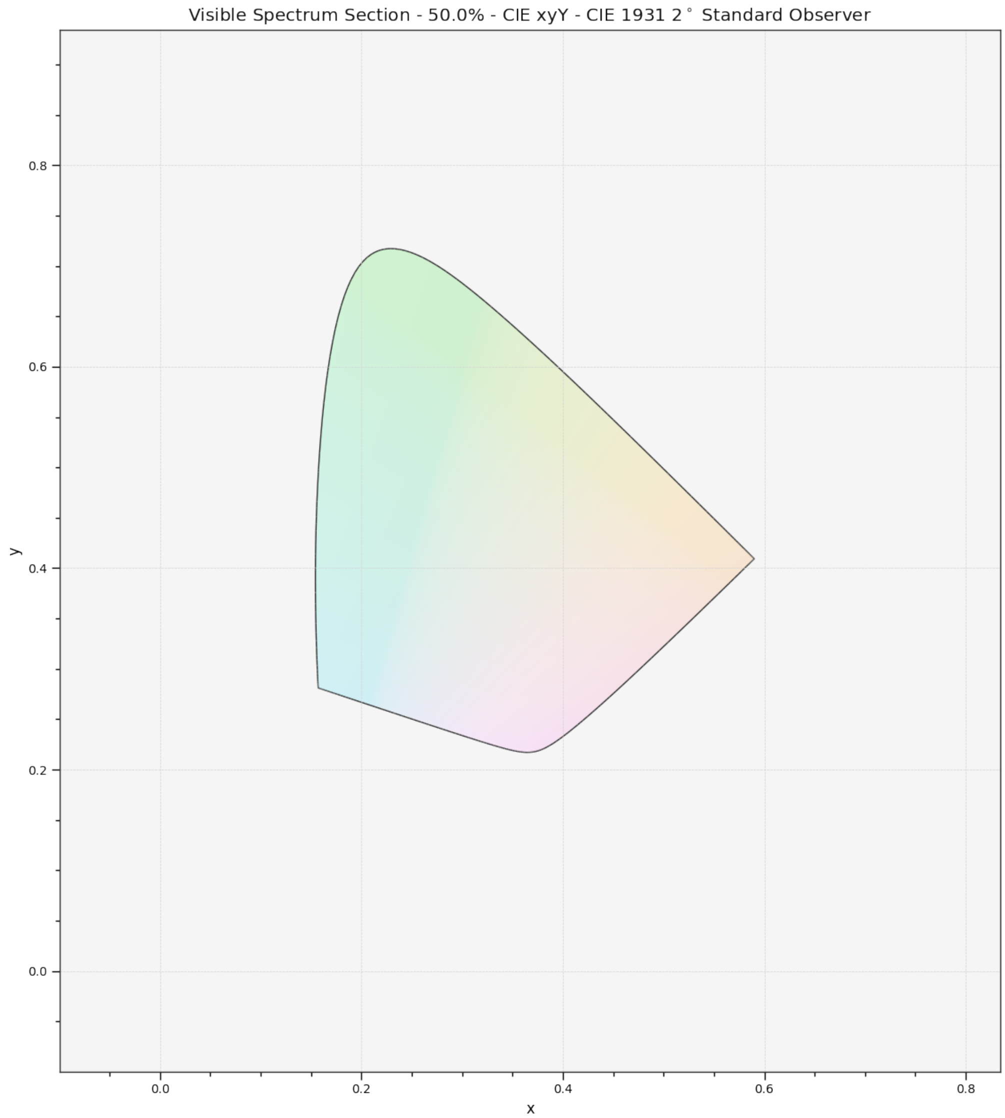
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | str | collections.abc.Sequence[colour.colorimetry.spectrum.MultiSpectralDistributions | str]`) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*. `cmfs` can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **illuminant** (`colour.colorimetry.spectrum.SpectralDistribution | str`) – Illuminant spectral distribution, default to *CIE Illuminant D65*. `illuminant` can be of any type or form supported by the `colour.plotting.common.filter_illuminants()` definition.
- **model** (`Union[Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'], str]`) – Colourspace model, see `colour.COLOURSPACE_MODELS` attribute for the list of supported colourspace models.
- **axis** (`Union[Literal['+z', '+x', '+y'], str]`) – Axis the hull section will be normal to.
- **origin** (`float`) – Coordinate along `axis` at which to plot the hull section.
- **normalise** (`bool`) – Whether to normalise `axis` to the extent of the hull along it.
- **show_section_colours** (`bool`) – Whether to show the hull section colours.
- **show_section_contour** (`bool`) – Whether to show the hull section contour.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.render()`, `colour.plotting.section.plot_hull_section_colours()` `colour.plotting.section.plot_hull_section_contour()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour.utilities import is_trimesh_installed
>>> if is_trimesh_installed:
...     plot_visible_spectrum_section(
...         section_colours="RGB", section_opacity=0.15
...     )
...
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_RGB_colourspace_section

colour.plotting.**plot_RGB_colourspace_section**(*colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str], model: Union[Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'], str] = 'CIE xyY', axis: Union[Literal['+z', '+x', '+y'], str] = '+z', origin: float = 0.5, normalise: bool = True, show_section_colours: bool = True, show_section_contour: bool = True, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given *RGB* colourspace section colours along given axis and origin.

Parameters

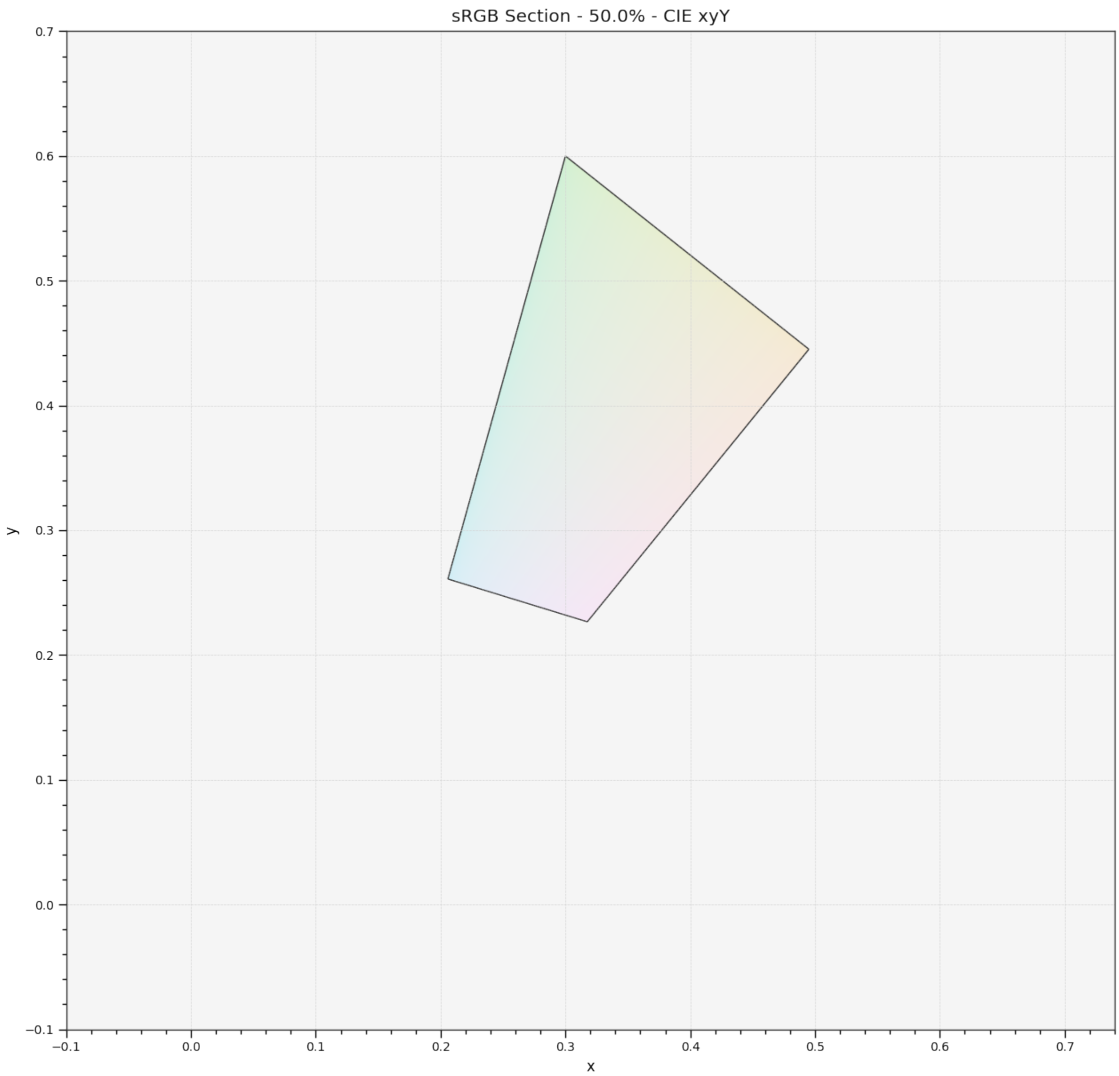
- **colourspace** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace | str | collections.abc.Sequence[colour.models.rgb.rgb_colourspace.RGB_Colourspace | str]`) – *RGB* colourspace of the *RGB* array. `colourspace` can be of any type or form supported by the `colour.plotting.common.filter_RGB_colourspaces()` definition.
- **model** (`Union[Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'], str]`) – Colourspace model, see `colour.COLOURSPACE_MODELS` attribute for the list of supported colourspace models.
- **axis** (`Union[Literal['+z', '+x', '+y'], str]`) – Axis the hull section will be normal to.
- **origin** (`float`) – Coordinate along `axis` at which to plot the hull section.
- **normalise** (`bool`) – Whether to normalise `axis` to the extent of the hull along it.
- **show_section_colours** (`bool`) – Whether to show the hull section colours.
- **show_section_contour** (`bool`) – Whether to show the hull section contour.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.render()`, `colour.plotting.section.plot_hull_section_colours()` `colour.plotting.section.plot_hull_section_contour()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour.utilities import is_trimesh_installed
>>> if is_trimesh_installed:
...     plot_RGB_colourspace_section(
...         "sRGB", section_colours="RGB", section_opacity=0.15
...     )
...
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Ancillary Objects

colour.plotting.section

plot_hull_section_colours(hull[, model, ...])	Plot the section colours of given *trimesh* hull along given axis and origin.
plot_hull_section_contour(hull[, model, ...])	Plot the section contour of given *trimesh* hull along given axis and origin.

colour.plotting.section.plot_hull_section_colours

colour.plotting.section.**plot_hull_section_colours**(*hull: trimesh.Trimesh, model: Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'] | str = 'CIE xyY', axis: Literal['+z', '+x', '+y'] | str = '+z', origin: float = 0.5, normalise: bool = True, section_colours: ArrayLike | str | None = None, section_opacity: float = 1, convert_kwargs: dict | None = None, samples: int = 256, **kwargs: Any*) → Tuple[plt.Figure, plt.Axes]

Plot the section colours of given *trimesh* hull along given axis and origin.

Parameters

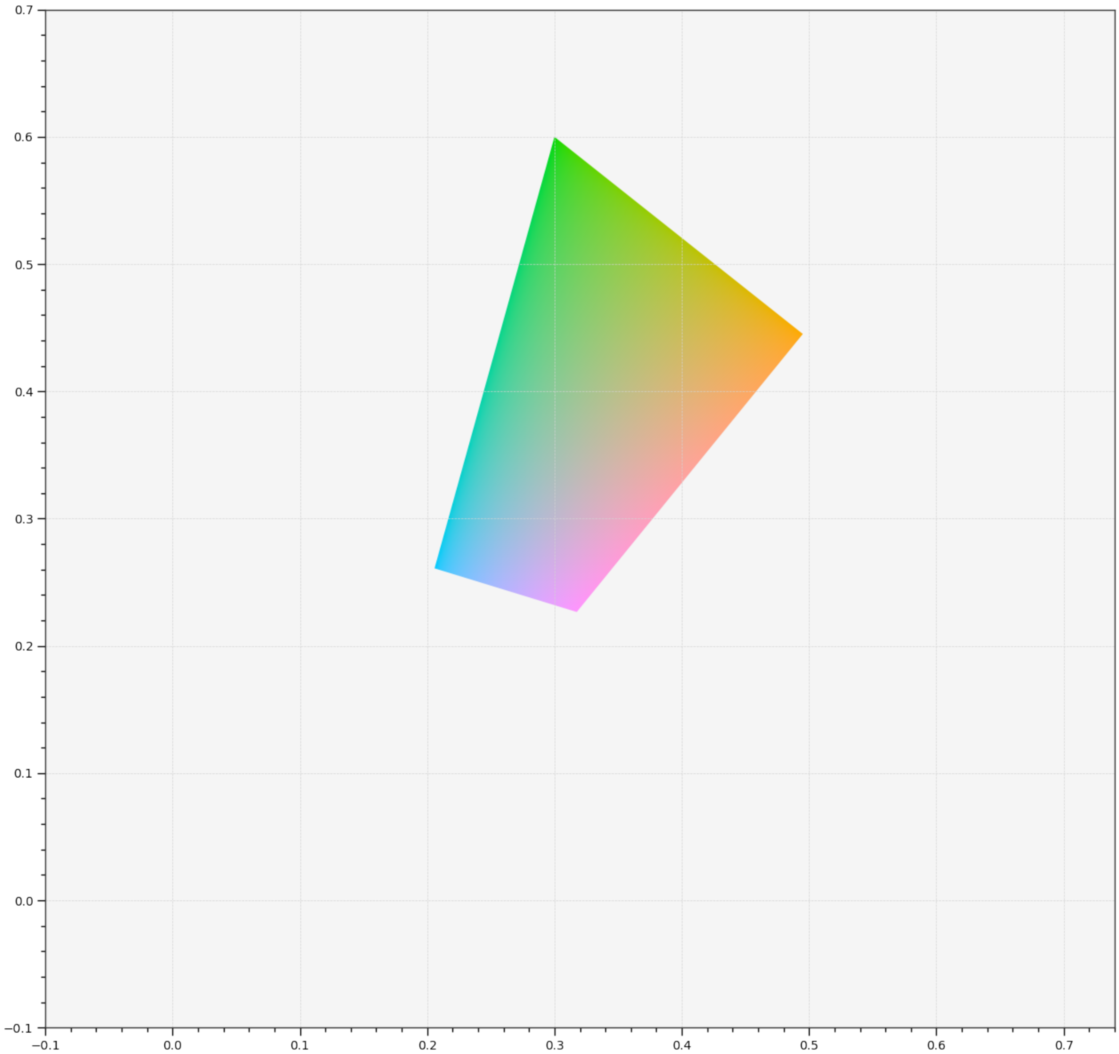
- **hull** (trimesh.Trimesh) – *Trimesh* hull.
- **model** (Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'] | str) – Colourspace model, see colour.COLOURSPACE_MODELS attribute for the list of supported colourspace models.
- **axis** (Literal['+z', '+x', '+y'] | str) – Axis the hull section will be normal to.
- **origin** (float) – Coordinate along axis at which to plot the hull section.
- **normalise** (bool) – Whether to normalise axis to the extent of the hull along it.
- **section_colours** (ArrayLike | str | None) – Colours of the hull section, if section_colours is set to *RGB*, the colours will be computed according to the corresponding coordinates.
- **section_opacity** (float) – Opacity of the hull section colours.
- **convert_kwargs** (dict | None) – Keyword arguments for the colour.convert() definition.
- **samples** (int) – Samples count on one axis when computing the hull section colours.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB
>>> from colour.utilities import is_trimesh_installed
>>> vertices, faces, _outline = primitive_cube(1, 1, 1, 64, 64, 64)
>>> XYZ_vertices = RGB_to_XYZ(
...     vertices["position"] + 0.5, RGB_COLOURSPACE_sRGB
... )
>>> if is_trimesh_installed:
...     from trimesh import Trimesh
...
...     hull = Trimesh(XYZ_vertices, faces, process=False)
...     plot_hull_section_colours(hull, section_colours="RGB")
...
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.section.plot_hull_section_contour

colour.plotting.section.**plot_hull_section_contour**(*hull: trimesh.Trimesh, model: Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'] | str = 'CIE xyY', axis: Literal['+z', '+x', '+y'] | str = '+z', origin: float = 0.5, normalise: bool = True, contour_colours: ArrayLike | str | None = None, contour_opacity: float = 1, convert_kwargs: dict | None = None, **kwargs: Any*) → Tuple[plt.Figure, plt.Axes]

Plot the section contour of given *trimesh* hull along given axis and origin.

Parameters

- **hull** (trimesh.Trimesh) – *Trimesh* hull.
- **model** (Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'] | str) – Colourspace model, see colour.COLOURSPACE_MODELS attribute for the list of supported colourspace models.
- **axis** (Literal['+z', '+x', '+y'] | str) – Axis the hull section will be normal to.
- **origin** (float) – Coordinate along axis at which to plot the hull section.
- **normalise** (bool) – Whether to normalise axis to the extent of the hull along it.
- **contour_colours** (ArrayLike | str | None) – Colours of the hull section contour, if contour_colours is set to *RGB*, the colours will be computed according to the corresponding coordinates.
- **contour_opacity** (float) – Opacity of the hull section contour.
- **convert_kwargs** (dict | None) – Keyword arguments for the colour.convert() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

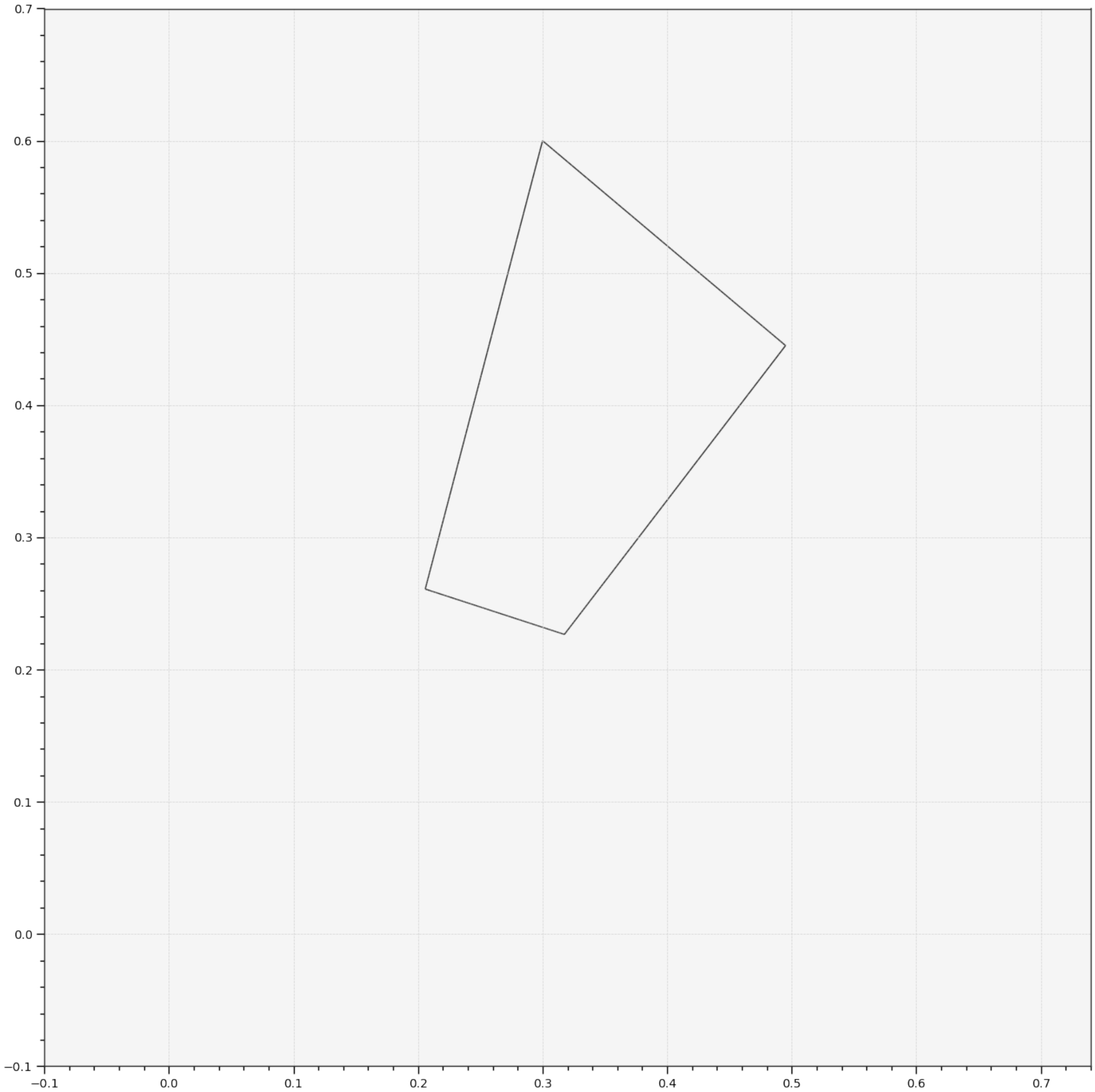
Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB
>>> from colour.utilities import is_trimesh_installed
>>> vertices, faces, _outline = primitive_cube(1, 1, 1, 64, 64, 64)
>>> XYZ_vertices = RGB_to_XYZ(
...     vertices["position"] + 0.5, RGB_COLOURSPACE_sRGB
... )
>>> if is_trimesh_installed:
...     from trimesh import Trimesh
```

(continues on next page)

(continued from previous page)

```
...
...     hull = Trimesh(XYZ_vertices, faces, process=False)
...     plot_hull_section_contour(hull, contour_colours="RGB")
...
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Colour Temperature & Correlated Colour Temperature

`colour.plotting`

`plot_planckian_locus_in_chromaticity_diagram_CIE1931(...)`	Plot the *Planckian Locus* and given illuminants in *CIE 1931 Chromaticity Diagram*.
`plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(...)`	Plot the *Planckian Locus* and given illuminants in *CIE 1960 UCS Chromaticity Diagram*.
`plot_planckian_locus_in_chromaticity_diagram_CIE1976UCS(...)`	Plot the *Planckian Locus* and given illuminants in *CIE 1976 UCS Chromaticity Diagram*.

colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_planckian_locus_in_chromaticity_diagram_CIE1931**(*illuminants: str | collections.abc.Sequence[str], chromaticity_diagram_callable_CIE1931: Callable = plot_chromaticity_diagram_CIE1931, annotate_kwargs: Optional[Union[dict, List[dict]]] = None, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *Planckian Locus* and given illuminants in *CIE 1931 Chromaticity Diagram*.

Parameters

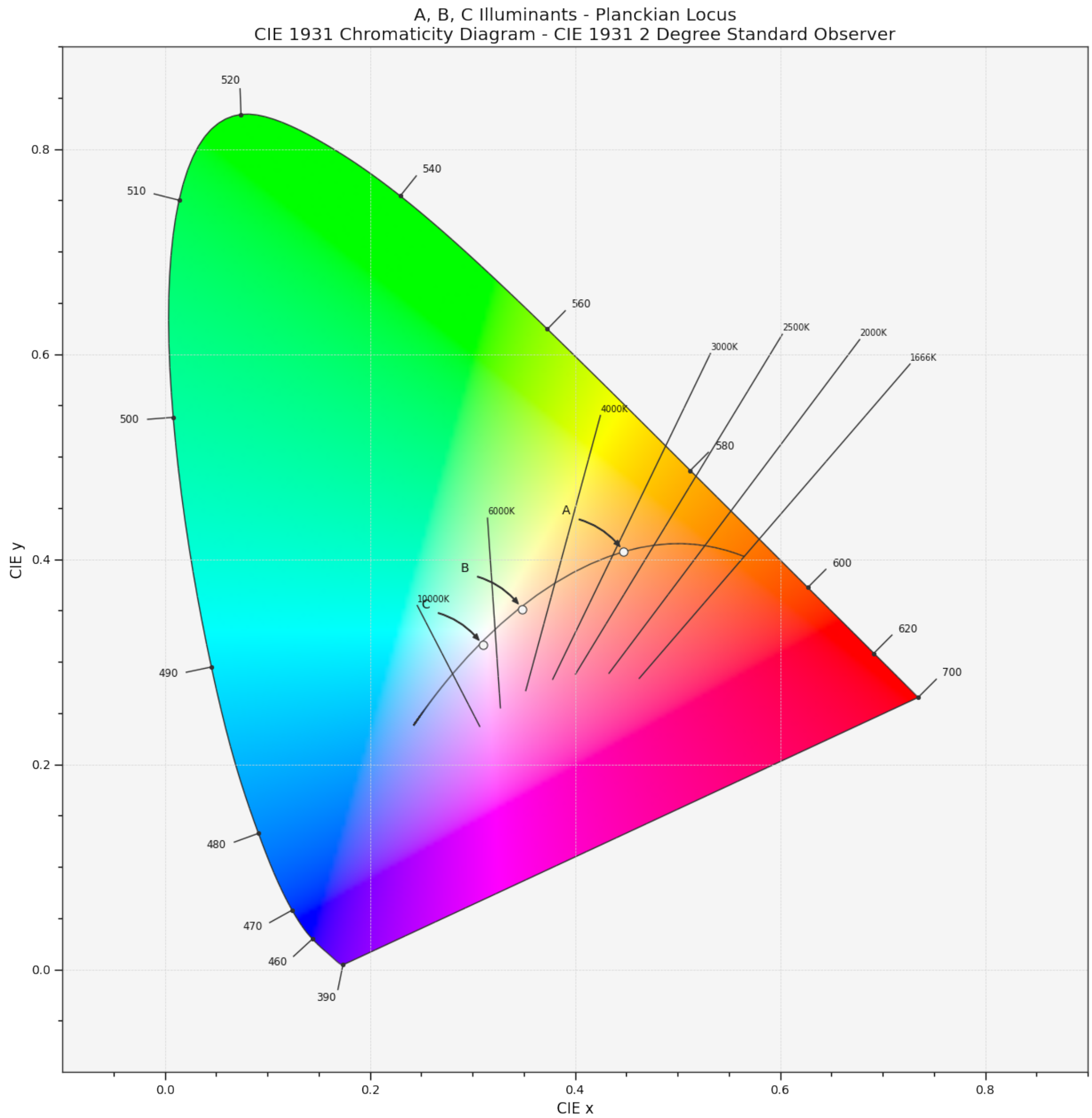
- **illuminants** (str | collections.abc.Sequence[str]) – Illuminants to plot. illuminants elements can be of any type or form supported by the colour.plotting.common.filter_passthrough() definition.
- **chromaticity_diagram_callable_CIE1931** (Callable) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **annotate_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for the matplotlib.pyplot.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. annotate_kwargs can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution. The following special keyword arguments can also be used:
  - annotate : Whether to annotate the spectral distributions.
- **plot_kwargs** (Optional[Union[dict, List[dict]]]) – Keyword arguments for the matplotlib.pyplot.plot() definition, used to control the style of the plotted illuminants. plot_kwargs can be either a single dictionary applied to all the plotted illuminants with the same settings or a sequence of dictionaries with different settings for eachplotted illuminant.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.temperature.plot_planckian_locus(), colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1931(["A", "B", "C"])
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS**(*illuminants: str | collections.abc.Sequence[str], chromaticity_diagram_callable_CIE1960UCS: Callable = plot_chromaticity_diagram_CIE1960UC…, annotate_kwargs: Optional[Union[dict, List[dict]]] = None, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *Planckian Locus* and given illuminants in *CIE 1960 UCS Chromaticity Diagram*.

Parameters

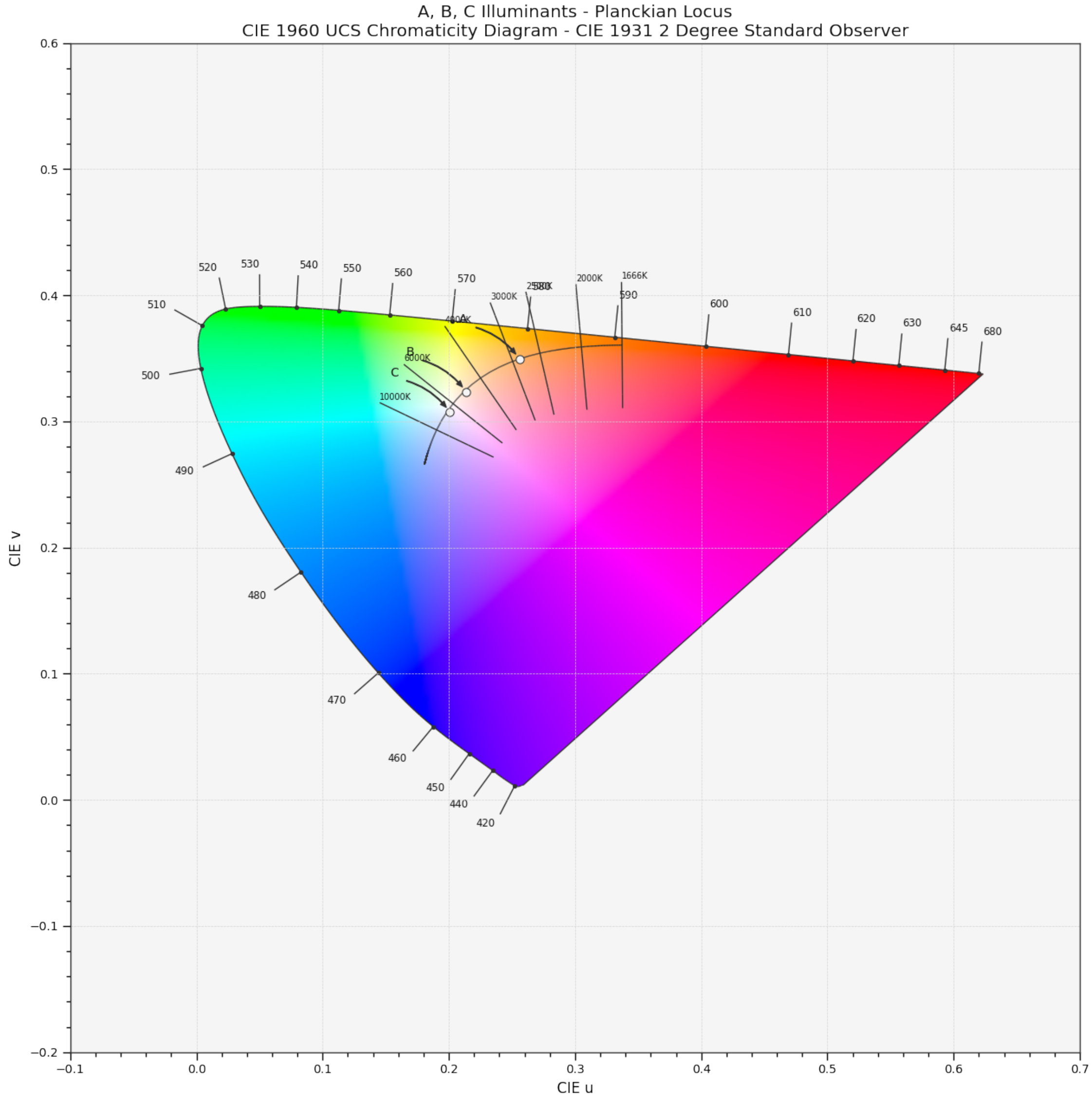
- **illuminants** (`str | collections.abc.Sequence[str]`) – Illuminants to plot. `illuminants` elements can be of any type or form supported by the `colour.plotting.common.filter_passthrough()` definition.
- **chromaticity_diagram_callable_CIE1960UCS** (`Callable`) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **annotate_kwargs** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. `annotate_kwargs` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution. The following special keyword arguments can also be used:
  - `annotate` : Whether to annotate the spectral distributions.
- **plot_kwargs** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.plot()` definition, used to control the style of the plotted illuminants. `plot_kwargs` can be either a single dictionary applied to all the plotted illuminants with the same settings or a sequence of dictionaries with different settings for eachplotted illuminant.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.temperature.plot_planckian_locus()`, `colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(
...     ["A", "C", "E"]
... )
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_planckian_locus_in_chromaticity_diagram_CIE1976UCS**(*illuminants: str* | collections.abc.Sequence[*str*], *chromaticity_diagram_callable_CIE1976UCS: Callable = plot_chromaticity_diagram_CIE1976UCS*, *annotate_kwargs: Optional[Union[dict, List[dict]]] = None*, *plot_kwargs: Optional[Union[dict, List[dict]]] = None*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *Planckian Locus* and given illuminants in *CIE 1976 UCS Chromaticity Diagram*.

Parameters

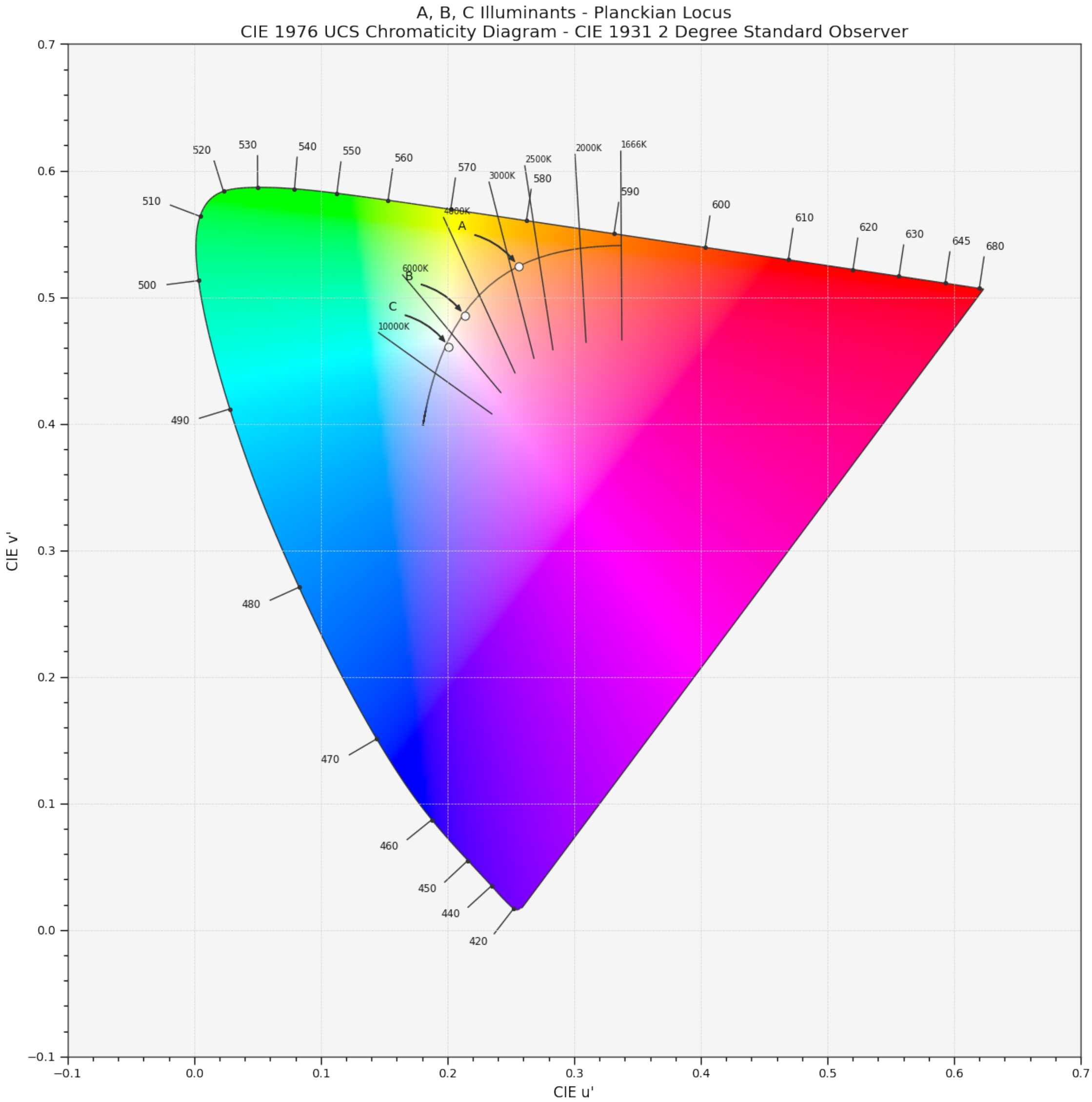
- **illuminants** (`str | collections.abc.Sequence[str]`) – Illuminants to plot. `illuminants` elements can be of any type or form supported by the `colour.plotting.common.filter_passthrough()` definition.
- **chromaticity_diagram_callable_CIE1976UCS** (`Callable`) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **annotate_kwargs** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. `annotate_kwargs` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution. The following special keyword arguments can also be used:
  - `annotate` : Whether to annotate the spectral distributions.
- **plot_kwargs** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.plot()` definition, used to control the style of the plotted illuminants. `plot_kwargs` can be either a single dictionary applied to all the plotted illuminants with the same settings or a sequence of dictionaries with different settings for eachplotted illuminant.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.temperature.plot_planckian_locus()`, `colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1976UCS(
...     ["A", "C", "E"]
... )
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Ancillary Objects

`colour.plotting.temperature`

plot_daylight_locus([...])	Plot the *Daylight Locus* according to given method.
plot_planckian_locus([...])	Plot the *Planckian Locus* according to given method.
plot_planckian_locus_in_chromaticity_diagram(...)	Plot the *Planckian Locus* and given illuminants in the *Chromaticity Diagram* according to given method.

colour.plotting.temperature.plot_daylight_locus

colour.plotting.temperature.**plot_daylight_locus**(*daylight_locus_colours: ArrayLike | str | None = None, daylight_locus_opacity: float = 1, daylight_locus_use_mireds: bool = False, method: Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str = 'CIE 1931', **kwargs: Any*) → Tuple[plt.Figure, plt.Axes]

Plot the *Daylight Locus* according to given method.

Parameters

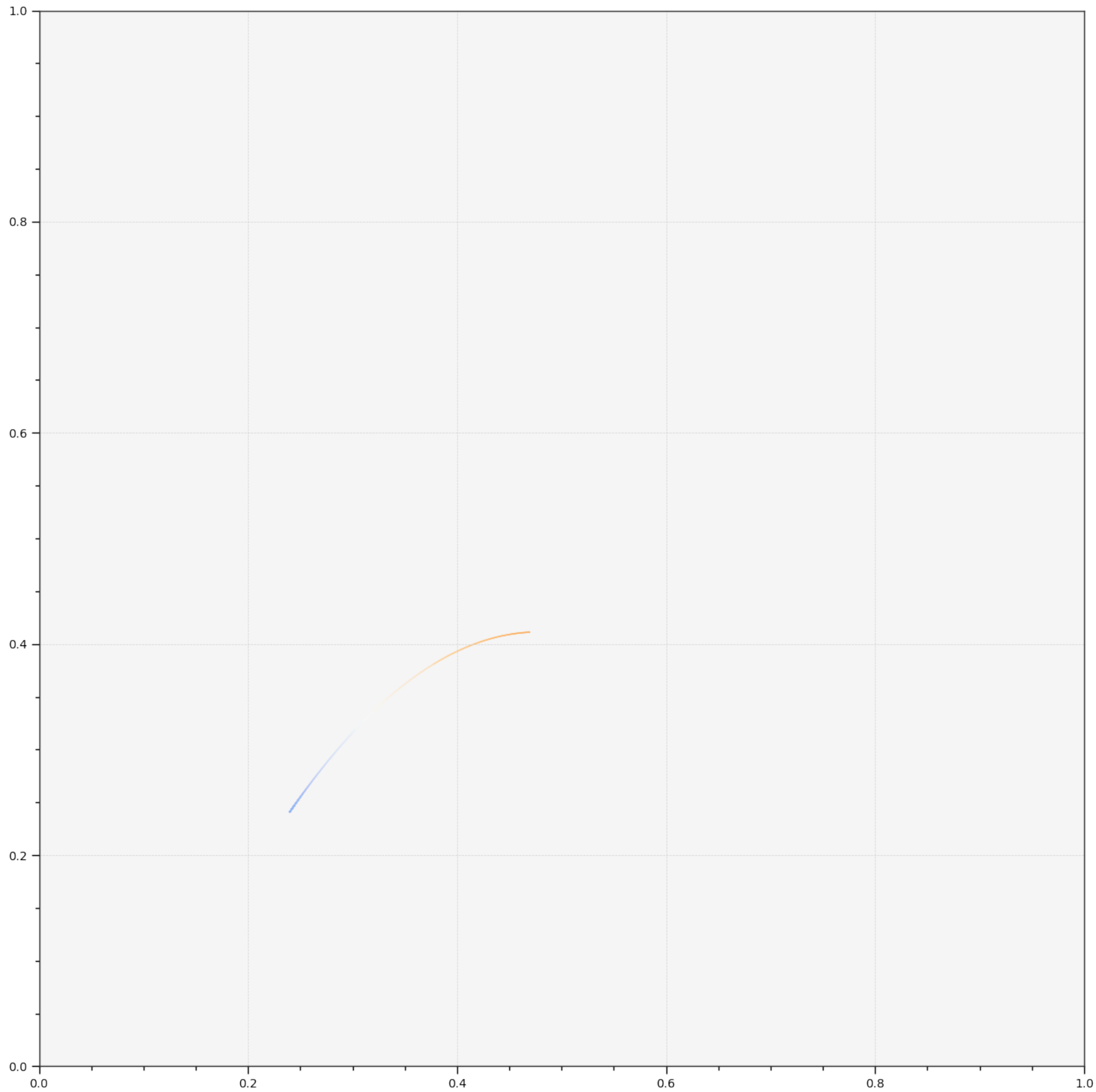
- **daylight_locus_colours** (ArrayLike | str | None) – Colours of the *Daylight Locus*, if daylight_locus_colours is set to *RGB*, the colours will be computed according to the corresponding chromaticity coordinates.
- **daylight_locus_opacity** (float) – Opacity of the *Daylight Locus*.
- **daylight_locus_use_mireds** (bool) – Whether to use micro reciprocal degrees for the iso-temperature lines.
- **method** (Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str) – *Chromaticity Diagram* method.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_daylight_locus(daylight_locus_colours="RGB")
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.temperature.plot_planckian_locus

colour.plotting.temperature.**plot_planckian_locus**(*planckian_locus_colours: ArrayLike | str | None = None, planckian_locus_opacity: float = 1, planckian_locus_labels: Sequence | None = None, planckian_locus_use_mireds: bool = False, planckian_locus_iso_temperature_lines_D_uv: float = 0.05, method: Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str = 'CIE 1931', **kwargs: Any*) → Tuple[plt.Figure, plt.Axes]

Plot the *Planckian Locus* according to given method.

Parameters

- **planckian_locus_colours** (ArrayLike | str | None) – Colours of the *Planckian Locus*, if planckian_locus_colours is set to *RGB*, the colours will be computed according to the corresponding chromaticity coordinates.
- **planckian_locus_opacity** (float) – Opacity of the *Planckian Locus*.
- **planckian_locus_labels** (Sequence | None) – Array of labels used to customise which iso-temperature lines will be drawn along the *Planckian Locus*. Passing an

empty array will result in no iso-temperature lines being drawn.

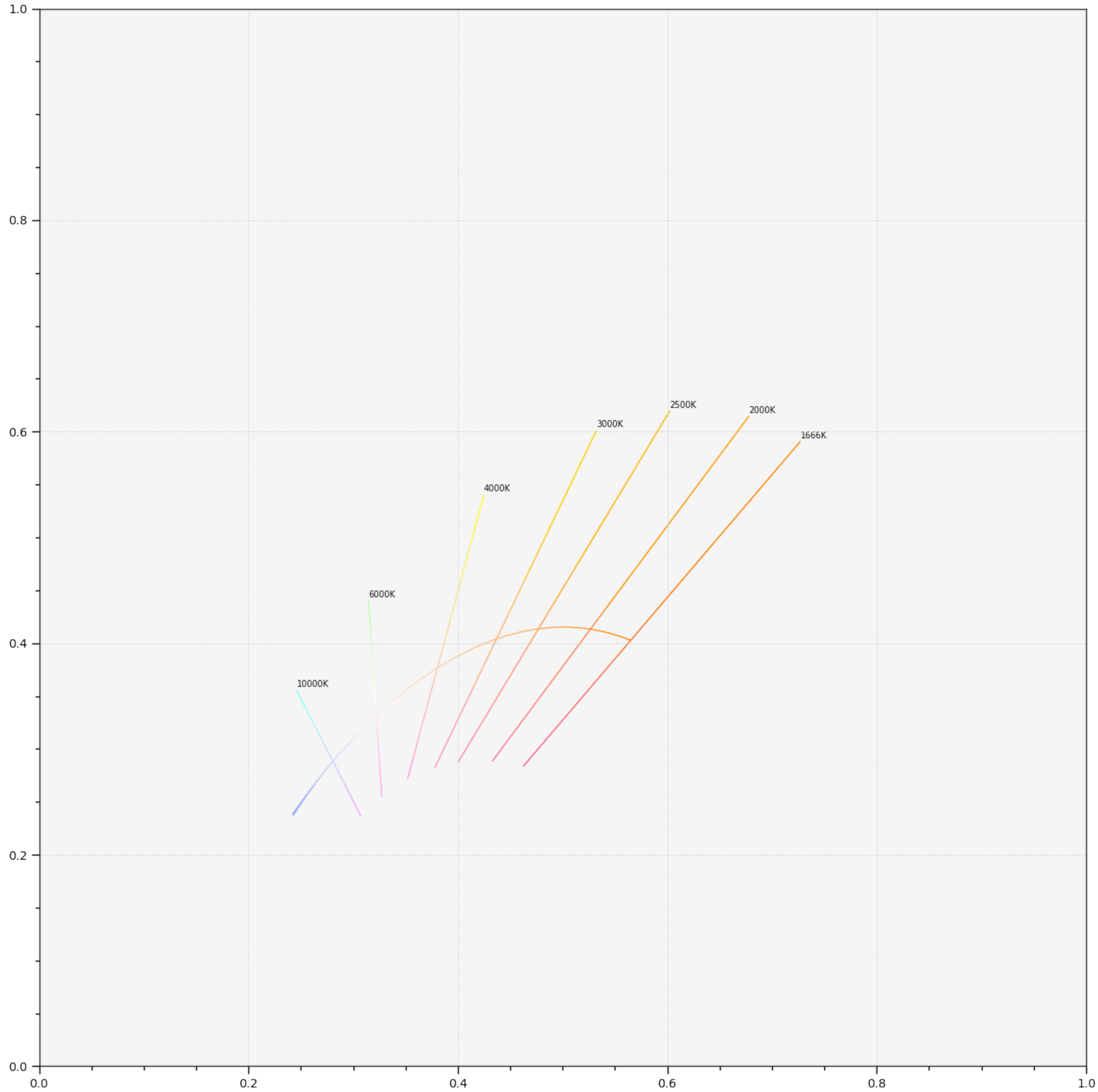
- **planckian_locus_use_mireds** (bool) – Whether to use micro reciprocal degrees for the iso-temperature lines.
- **planckian_locus_iso_temperature_lines_D_uv** (float) – Iso-temperature lines Δ_{uv} length on each side of the *Planckian Locus*.
- **method** (Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'] | str) – *Chromaticity Diagram* method.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus(planckian_locus_colours="RGB")
...
(<Figure size ... with 1 Axes>, <...Axes...>)
```



colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram

`colour.plotting.temperature.`**`plot_planckian_locus_in_chromaticity_diagram`**(*illuminants: str | collections.abc.Sequence[str], chromaticity_diagram_callable: Callable = plot_chromaticity_diagram, method: Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str] = 'CIE 1931', annotate_kwargs: Optional[Union[dict, List[dict]]] = None, plot_kwargs: Optional[Union[dict, List[dict]]] = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot the *Planckian Locus* and given illuminants in the *Chromaticity Diagram* according to given method.

Parameters

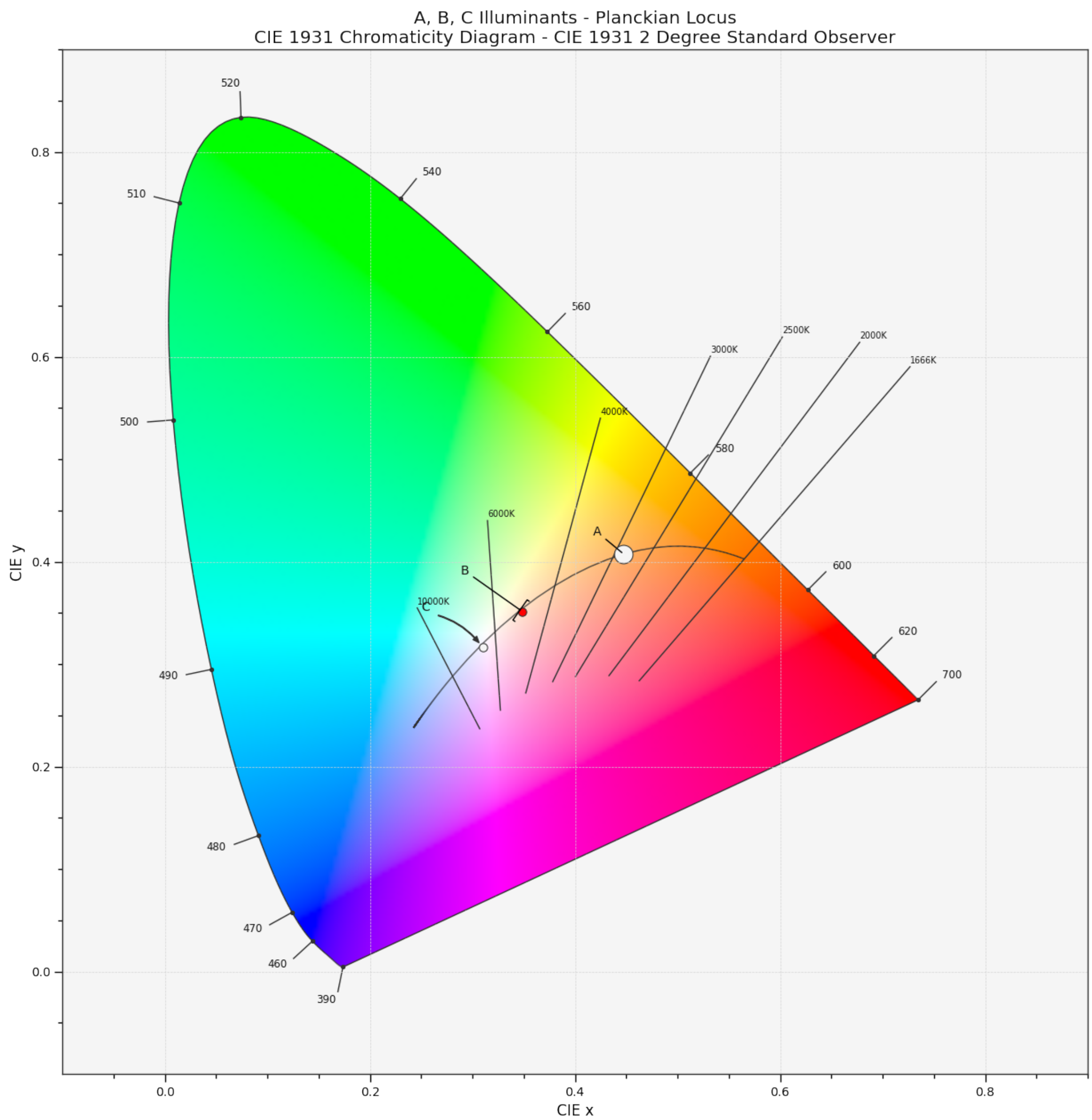
- **`illuminants`** (`str | collections.abc.Sequence[str]`) – Illuminants to plot. `illuminants` elements can be of any type or form supported by the `colour.plotting.common.filter_passthrough()` definition.
- **`chromaticity_diagram_callable`** (`Callable`) – Callable responsible for drawing the *Chromaticity Diagram*.
- **`method`** (`Union[Literal['CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'], str]`) – *Chromaticity Diagram* method.
- **`annotate_kwargs`** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. `annotate_kwargs` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution. The following special keyword arguments can also be used:
  - `annotate` : Whether to annotate the spectral distributions.
- **`plot_kwargs`** (`Optional[Union[dict, List[dict]]]`) – Keyword arguments for the `matplotlib.pyplot.plot()` definition, used to control the style of the plotted illuminants. `plot_kwargs` can be either a single dictionary applied to all the plotted illuminants with the same settings or a sequence of dictionaries with different settings for eachplotted illuminant.
- **`kwargs`** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.temperature.plot_planckian_locus()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> annotate_kwargs = [
...     {"xytext": (-25, 15), "arrowprops": {"arrowstyle": "-"}},
...     {"arrowprops": {"arrowstyle": "-["}},
...     {},
... ]
>>> plot_kwargs = [
...     {
...         "markersize": 15,
...     },
...     {"color": "r"},
...     {},
... ]
>>> plot_planckian_locus_in_chromaticity_diagram(
...     ["A", "B", "C"],
...     annotate_kwargs=annotate_kwargs,
...     plot_kwargs=plot_kwargs,
... )
(<Figure size ... with 1 Axes>, <...Axes...>)
```



Colour Models Volume

colour.plotting

plot_RGB_colourspaces_gamuts(colourspaces[, ...])	Plot given *RGB* colourspaces gamuts in given reference colourspace.
plot_RGB_scatter(RGB[, colourspace, model, ...])	Plot given *RGB* colourspace array in a scatter plot.

colour.plotting.plot_RGB_colourspaces_gamuts

colour.plotting.**plot_RGB_colourspaces_gamuts**(*colourspaces: RGB_Colourspace | str | Sequence[RGB_Colourspace | str], model: Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'] | str = 'CIE xyY', segments: int = 8, show_grid: bool = True, grid_segments: int = 10, show_spectral_locus: bool = False, spectral_locus_colour: ArrayLike | str | None = None, cmfs: MultiSpectralDistributions | str | Sequence[MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer', chromatically_adapt: bool = False, convert_kwargs: dict | None = None, **kwargs: Any*) → Tuple[plt.Figure, Axes3D]

Plot given *RGB* colourspaces gamuts in given reference colourspace.

Parameters

- **colourspaces** (RGB_Colourspace | str | Sequence[RGB_Colourspace | str]) – *RGB* colourspaces to plot the gamuts. colourspaces elements can be of any type or form supported by the `colour.plotting.common.filter_RGB_colourspaces()` definition.
- **model** (Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'] | str) – Colourspace model, see `colour.COLOURSPACE_MODELS` attribute for the list of supported colourspace models.
- **segments** (int) – Edge segments count for each *RGB* colourspace cubes.
- **show_grid** (bool) – Whether to show a grid at the bottom of the *RGB* colourspace cubes.
- **grid_segments** (int) – Edge segments count for the grid.
- **show_spectral_locus** (bool) – Whether to show the spectral locus.
- **spectral_locus_colour** (ArrayLike | str | None) – Spectral locus colour.
- **cmfs** (MultiSpectralDistributions | str | Sequence[MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the `colour.plotting.common.filter_cmfs()` definition.
- **chromatically_adapt** (bool) – Whether to chromatically adapt the *RGB* colourspaces given in colourspaces to the whitepoint of the default plotting colourspace.
- **convert_kwargs** (dict | None) – Keyword arguments for the `colour.convert()` definition.
- **edge_colours** – Edge colours array such as *edge_colours = (None, (0.5, 0.5, 1.0))*.
- **edge_alpha** – Edge opacity value such as *edge_alpha = (0.0, 1.0)*.

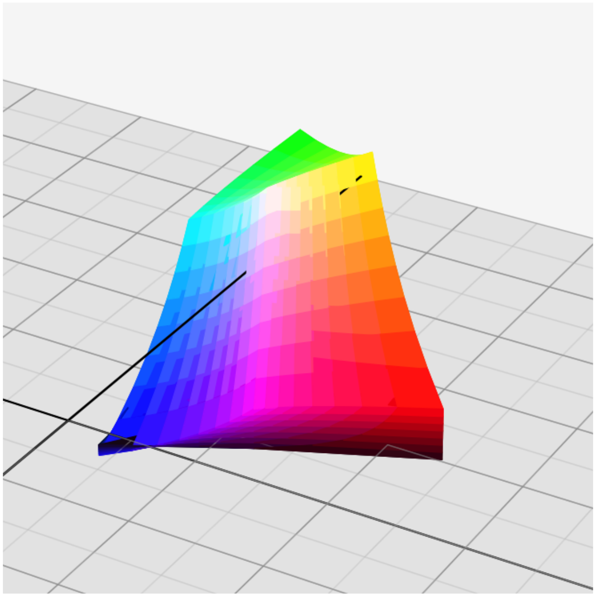
- **face_alpha** – Face opacity value such as *face_alpha = (0.5, 1.0)*.
- **face_colours** – Face colours array such as *face_colours = (None, (0.5, 0.5, 1.0))*.
- **kwargs** (Any) – {`colour.plotting.artist()`, `colour.plotting.volume.nadir_grid()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_RGB_colourspaces_gamuts(["ITU-R BT.709", "ACEScg", "S-Gamut"])
...
(<Figure size ... with 1 Axes>, <...Axes3D...>)
```



colour.plotting.plot_RGB_scatter

`colour.plotting.plot_RGB_scatter`(*RGB: ArrayLike, colourspace: RGB_Colourspace | str | Sequence[RGB_Colourspace | str] = 'sRGB', model: Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'] | str = 'CIE xyY', colourspaces: RGB_Colourspace | str | Sequence[RGB_Colourspace | str] | None = None, segments: int = 8, show_grid: bool = True, grid_segments: int = 10, show_spectral_locus: bool = False, spectral_locus_colour: ArrayLike | str | None = None, points_size: float = 12, cmfs: MultiSpectralDistributions | str | Sequence[MultiSpectralDistributions | str] = 'CIE 1931 2 Degree Standard Observer', chromatically_adapt: bool = False, convert_kwargs: dict | None = None, **kwargs: Any*) → Tuple[plt.Figure, Axes3D]

Plot given *RGB* colourspace array in a scatter plot.

Parameters

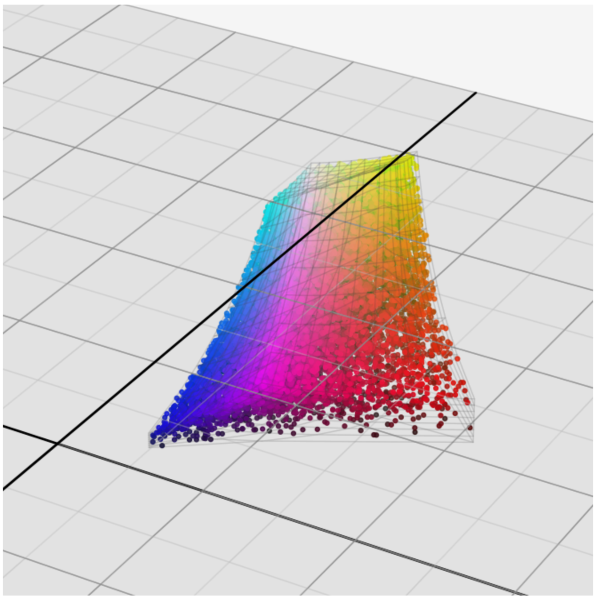
- **RGB** (ArrayLike) – *RGB* colourspace array.
- **colourspace** (RGB_Colourspace | str | Sequence[RGB_Colourspace | str]) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.common.filter_RGB_colourspaces() definition.
- **model** (Literal['CAM02LCD', 'CAM02SCD', 'CAM02UCS', 'CAM16LCD', 'CAM16SCD', 'CAM16UCS', 'CIE XYZ', 'CIE xyY', 'CIE Lab', 'CIE Luv', 'CIE UCS', 'CIE UVW', 'DIN99', 'Hunter Lab', 'Hunter Rdab', 'ICaCb', 'ICtCp', 'IPT', 'IgPgTg', 'Jzazbz', 'OSA UCS', 'Oklab', 'hdr-CIELAB', 'hdr-IPT'] | str) – Colourspace model, see colour.COLOURSPACE_MODELS attribute for the list of supported colourspace models.
- **colourspaces** (RGB_Colourspace | str | Sequence[RGB_Colourspace | str] | None) – *RGB* colourspaces to plot the gamuts. colourspaces elements can be of any type or form supported by the colour.plotting.common.filter_RGB_colourspaces() definition.
- **segments** (int) – Edge segments count for each *RGB* colourspace cubes.
- **show_grid** (bool) – Whether to show a grid at the bottom of the *RGB* colourspace cubes.
- **grid_segments** (int) – Edge segments count for the grid.
- **show_spectral_locus** (bool) – Whether to show the spectral locus.
- **spectral_locus_colour** (ArrayLike | str | None) – Spectral locus colour.
- **points_size** (float) – Scatter points size.
- **cmfs** (MultiSpectralDistributions | str | Sequence[MultiSpectralDistributions | str]) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.common.filter_cmfs() definition.
- **chromatically_adapt** (bool) – Whether to chromatically adapt the *RGB* colourspaces given in colourspaces to the whitepoint of the default plotting colourspace.
- **convert_kwargs** (dict | None) – Keyword arguments for the colour.convert() definition.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.plot_RGB_colourspaces_gamuts()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_scatter(RGB, "ITU-R BT.709")
(<Figure size ... with 1 Axes>, <...Axes3D...>)
```



ANSI/IES TM-30-18 Colour Rendition Report

colour.plotting

plot_single_sd_colour_rendition_report(sd[, ...])	Generate the *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution according to given method.

colour.plotting.plot_single_sd_colour_rendition_report

colour.plotting.**plot_single_sd_colour_rendition_report**(*sd: colour.colorimetry.spectrum.SpectralDistribution, method: Union[Literal['Full', 'Intermediate', 'Simple'], str] = 'Full', **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Generate the *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution according to given method.

Parameters

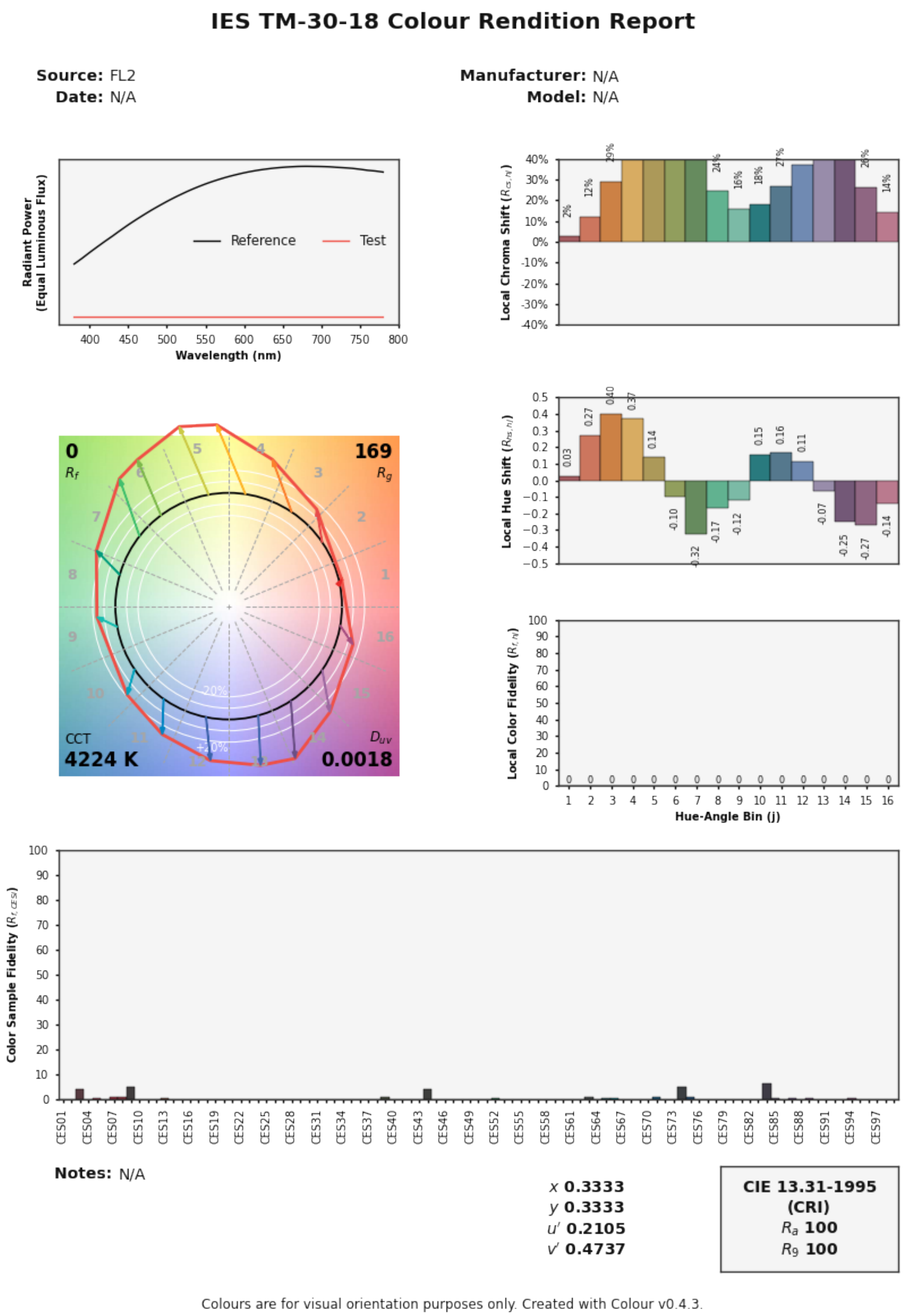
- **sd** (colour.colorimetry.spectrum.SpectralDistribution) – Spectral distribution of the emission source to generate the report for.
- **method** (Union[Literal['Full', 'Intermediate', 'Simple'], str]) – Report plotting method.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render(), colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full(), colour.plotting.tm3018.plot_single_sd_colour_rendition_report_intermediate(), colour.plotting.tm3018.plot_single_sd_colour_rendition_report_simple()} See the documentation of the previously listed definitions.

Returns Current figure and axes.

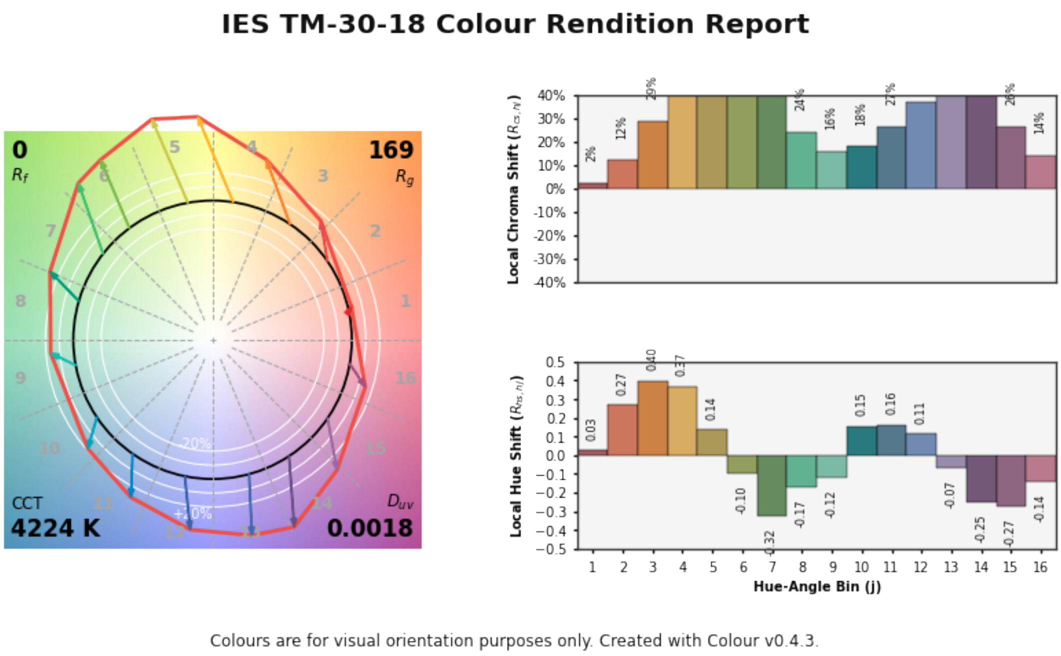
Return type tuple

Examples

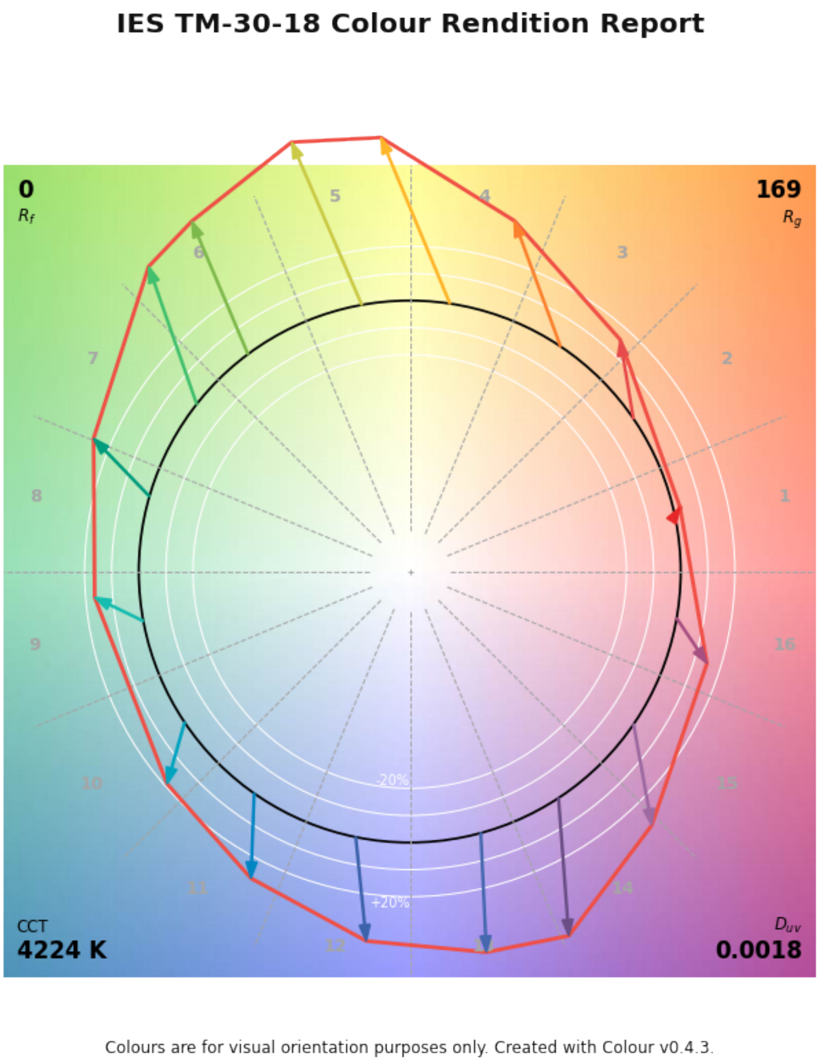
```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS["FL2"]
>>> plot_single_sd_colour_rendition_report(sd)
...
(<Figure size ... with ... Axes>, <...Axes...>)
```




```
>>> plot_single_sd_colour_rendition_report(sd, "Intermediate")
...
(<Figure size ... with ... Axes>, <...Axes...>)
```




```
>>> plot_single_sd_colour_rendition_report(sd, "Simple")
...
(<Figure size ... with ... Axes>, <...Axes...>)
```



Ancillary Objects

`colour.plotting.tm3018`

`plot_single_sd_colour_rendition_report_full(sd)`	Generate the full *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.
`plot_single_sd_colour_rendition_report_intermediate(sd)`	Generate the intermediate *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.
`plot_single_sd_colour_rendition_report_simple(sd)`	Generate the simple *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.

colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full

colour.plotting.tm3018.**plot_single_sd_colour_rendition_report_full**(*sd: colour.colorimetry.spectrum.SpectralDistribu source: str | None = None, date: str | None = None, manufacturer: str | None = None, model: str | None = None, notes: str | None = None, report_size: tuple = CONSTANT_REPORT_SIZE_FULL, report_row_height_ratios: tuple = CONSTANT_REPORT_ROW_HEIGHT_RATIOS_FU report_box_padding: dict | None = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Generate the full *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.

Parameters

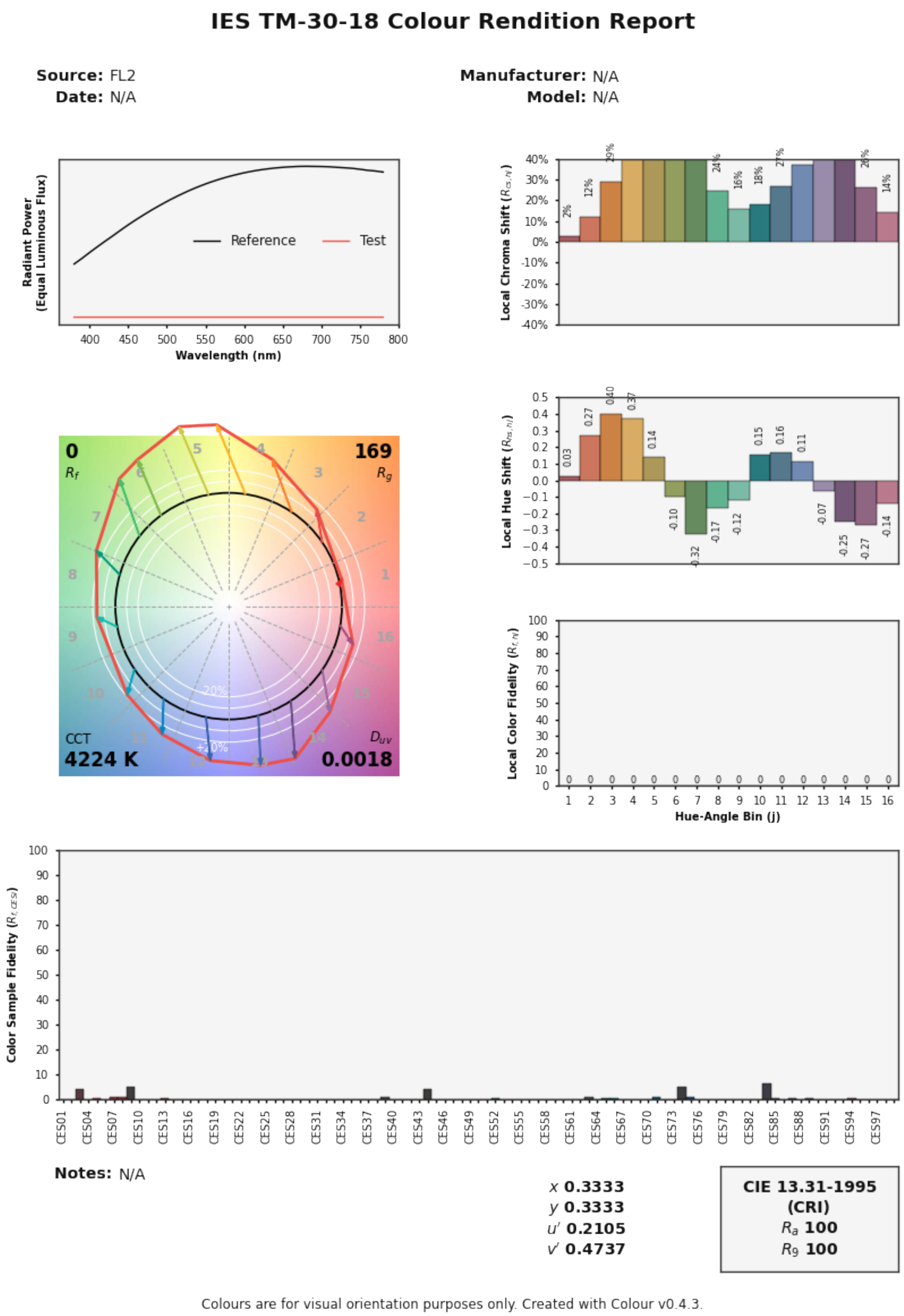
- **sd** (`colour.colorimetry.spectrum.SpectralDistribution`) – Spectral distribution of the emission source to generate the report for.
- **source** (`str | None`) – Emission source name, defaults to *colour.SpectralDistribution_IESTM2714.header.description* or *colour.SpectralDistribution_IESTM2714.name* properties value.
- **date** (`str | None`) – Emission source measurement date, defaults to *colour.SpectralDistribution_IESTM2714.header.report_date* property value.
- **manufacturer** (`str | None`) – Emission source manufacturer, defaults to *colour.SpectralDistribution_IESTM2714.header.manufacturer* property value.
- **model** (`str | None`) – Emission source model, defaults to *colour.SpectralDistribution_IESTM2714.header.catalog_number* property value.
- **notes** (`str | None`) – Notes pertaining to the emission source, defaults to *colour.SpectralDistribution_IESTM2714.header.comments* property value.
- **report_size** (`tuple`) – Report size, default to A4 paper size in inches.
- **report_row_height_ratios** (`tuple`) – Report size row height ratios.
- **report_box_padding** (`dict | None`) – Report box padding, tries to define the padding around the figure and in-between the axes.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS["FL2"]
>>> plot_single_sd_colour_rendition_report_full(sd)
...
(<Figure size ... with ... Axes>, <...Axes...>)
```


IES TM-30-18 Colour Rendition Report
Source: FL2
Date: N/A
Manufacturer: N/A
Model: N/A
Radiant Power (Equal Luminous Flux)
Reference
Test
Wavelength (nm)
Local Chroma Shift ($R_{cs,hj}$)
Local Hue Shift ($R_{hs,hj}$)
0
R_f
169
R_g
CCT
4224 K
D_{uv}
0.0018
Local Color Fidelity ($R_{f,hj}$)
Hue-Angle Bin (j)
Color Sample Fidelity ($R_{f,CESi}$)
Notes: N/A
x 0.3333
y 0.3333
u′ 0.2105
v′ 0.4737
CIE 13.31-1995
(CRI)
R_a 100
R_9 100
Colours are for visual orientation purposes only. Created with Colour v0.4.3.

colour.plotting.tm3018.plot_single_sd_colour_rendition_report_intermediate

colour.plotting.tm3018.**plot_single_sd_colour_rendition_report_intermediate**(*sd: colour.colorimetry.spectrum.Spectr report_size: tuple = CONSTANT_REPORT_SIZE_INTERMEDI report_row_height_ratios: tuple = CONSTANT_REPORT_ROW_HEIGHT_RA report_box_padding: dict | None = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Generate the intermediate *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.

Parameters

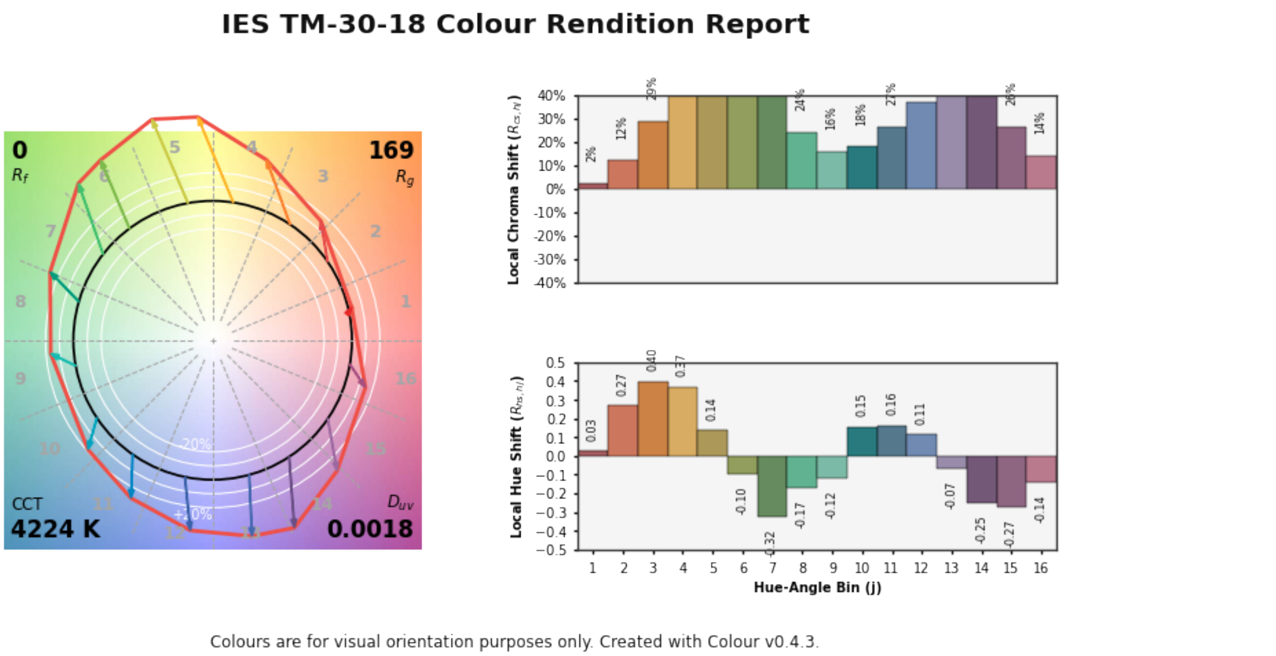
- **sd** (colour.colorimetry.spectrum.SpectralDistribution) – Spectral distribution of the emission source to generate the report for.
- **report_size** (tuple) – Report size, default to A4 paper size in inches.
- **report_row_height_ratios** (tuple) – Report size row height ratios.
- **report_box_padding** (dict | None) – Report box padding, tries to define the padding around the figure and in-between the axes.
- **kwargs** (Any) – {colour.plotting.artist(), colour.plotting.render()}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS["FL2"]
>>> plot_single_sd_colour_rendition_report_intermediate(sd)
...
(<Figure size ... with ... Axes>, <...Axes...>)
```



colour.plotting.tm3018.plot_single_sd_colour_rendition_report_simple

colour.plotting.tm3018.**plot_single_sd_colour_rendition_report_simple**(*sd: colour.colorimetry.spectrum.SpectralDistr*[...] *report_size: tuple = CONSTANT_REPORT_SIZE_SIMPLE, report_row_height_ratios: tuple = CONSTANT_REPORT_ROW_HEIGHT_RATIOS_*[...] *report_box_padding: dict | None = None, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Generate the simple *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.

Parameters

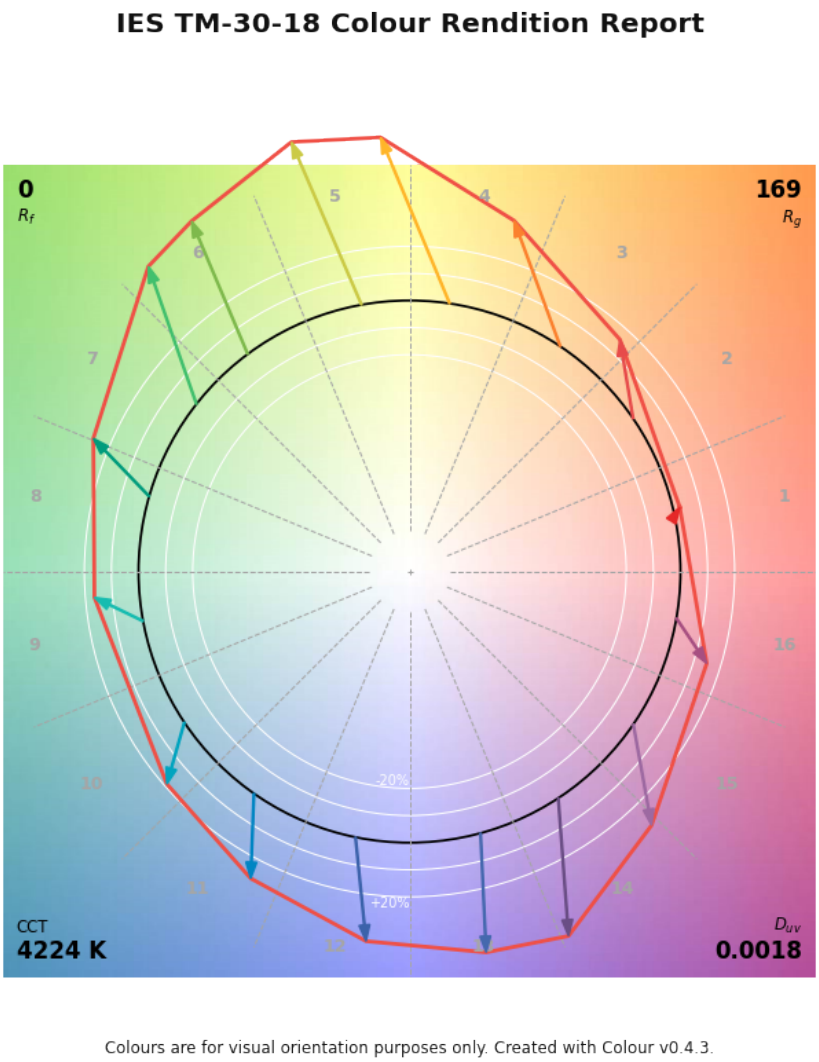
- **sd** (`colour.colorimetry.spectrum.SpectralDistribution`) – Spectral distribution of the emission source to generate the report for.
- **report_size** (`tuple`) – Report size, default to A4 paper size in inches.
- **report_row_height_ratios** (`tuple`) – Report size row height ratios.
- **report_box_padding** (`dict | None`) – Report box padding, tries to define the padding around the figure and in-between the axes.
- **kwargs** (`Any`) – {`colour.plotting.artist()`, `colour.plotting.render()`}, See the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS["FL2"]
>>> plot_single_sd_colour_rendition_report_simple(sd)
...
(<Figure size ... with ... Axes>, <...Axes...>)
```



Automatic Colour Conversion Graph

`colour.plotting`

`plot_automatic_colour_conversion_graph`(filename[, ...])	Plot *Colour* automatic colour conversion graph using Graphviz and pyraphviz.

colour.plotting.plot_automatic_colour_conversion_graph

colour.plotting.**plot_automatic_colour_conversion_graph**(*filename: str, prog: Literal['circo', 'dot', 'fdp', 'neato', 'nop', 'twopi'] | str = 'fdp', args: str = ''*) → AGraph

Plot *Colour* automatic colour conversion graph using Graphviz and pyraphviz.

Parameters

- **filename** (str) – Filename to use to save the image.
- **prog** (Literal['circo', 'dot', 'fdp', 'neato', 'nop', 'twopi'] | str) – *Graphviz* layout method.
- **args** (str) – Additional arguments for *Graphviz*.

Returns *Pyraphviz* graph.

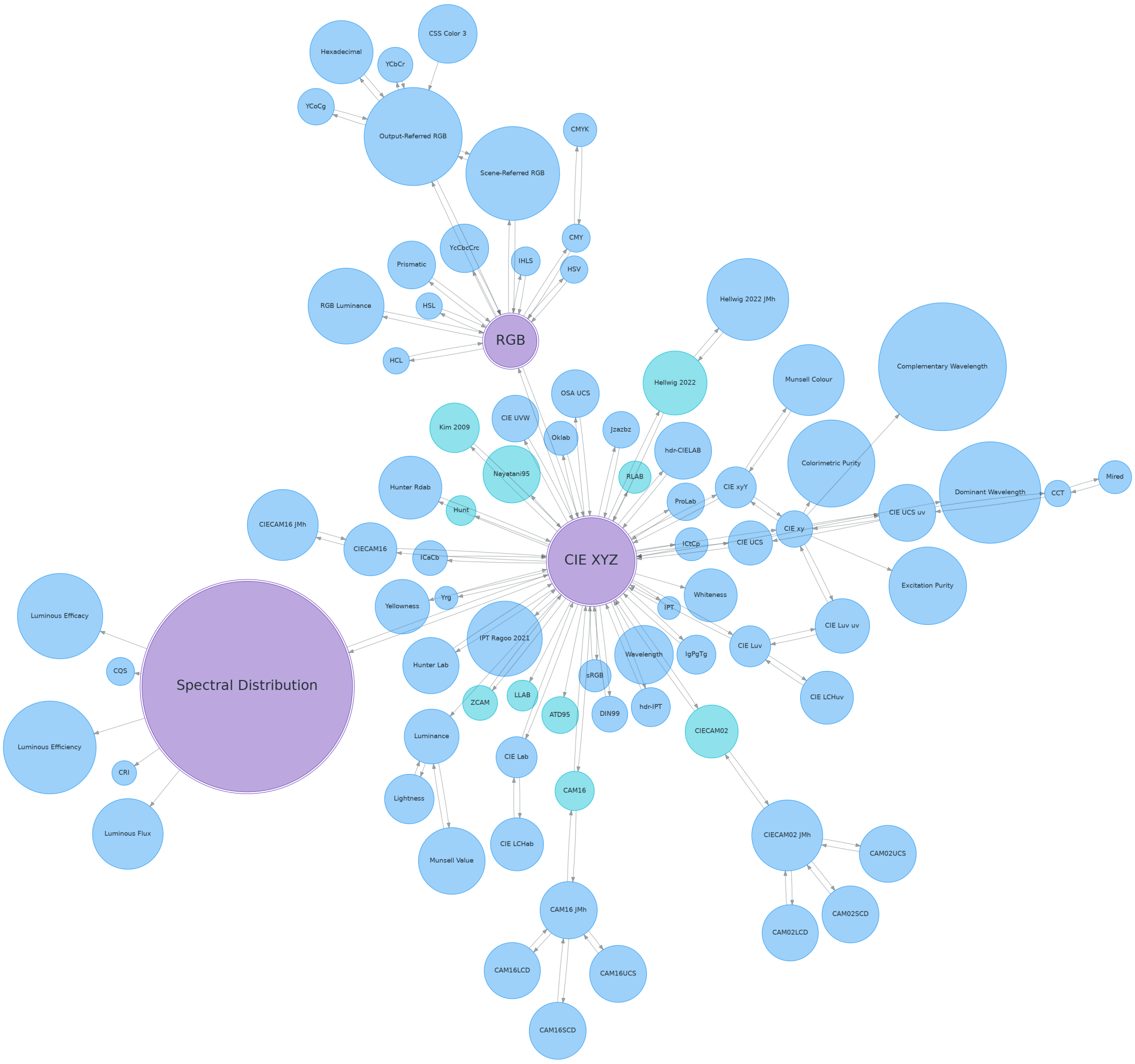
Return type AGraph

Notes

- This definition does not directly plot the *Colour* automatic colour conversion graph but instead write it to an image.

Examples

```
>>> import tempfile
>>> import colour
>>> from colour import read_image
>>> from colour.plotting import plot_image
>>> filename = "{0}.png".format(tempfile.mkstemp()[-1])
>>> _ = plot_automatic_colour_conversion_graph(filename, "dot")
...
>>> plot_image(read_image(filename))
```



Colour Quality

Colour Fidelity Index

colour

COLOUR_FIDELITY_INDEX_METHODS	Supported *Colour Fidelity Index* (CFI) computation methods.
colour_fidelity_index(sd_test[, ...])	Return the *Colour Fidelity Index* (CFI) R_f of given spectral distribution using given method.

colour.COLOUR_FIDELITY_INDEX_METHODS

colour.COLOUR_FIDELITY_INDEX_METHODS = CanonicalMapping({'CIE 2017': ..., 'ANSI/IES TM-30-18': ...})

Supported *Colour Fidelity Index* (CFI) computation methods.

References

[CIET19017], [ANSIIESCCommittee18]

colour.colour_fidelity_index

colour.**colour_fidelity_index**(*sd_test: colour.colorimetry.spectrum.SpectralDistribution, additional_data=False, method: Union[Literal['CIE 2017', 'ANSI/IES TM-30-18'], str] = 'CIE 2017'*) → float | *colour.quality.cfi2017.ColourRendering_Specification_CIE2017* | *colour.quality.tm3018.ColourQuality_Specification_ANSIIESTM3018*

Return the *Colour Fidelity Index* (CFI) R_f of given spectral distribution using given method.

Parameters

- **sd_test** (colour.colorimetry.spectrum.SpectralDistribution) – Test spectral distribution.
- **additional_data** – Whether to output additional data.
- **method** (Union[Literal['CIE 2017', 'ANSI/IES TM-30-18'], str]) – Computation method.

Returns *Colour Fidelity Index* (CFI) R_f.

Return type float or colour.quality.ColourRendering_Specification_CIE2017 or colour.quality.ColourQuality_Specification_ANSIIESTM3018

References

[CIET19017], [ANSIIESCCommittee18]

Examples

```
>>> from colour.colorimetry import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS["FL2"]
>>> colour_fidelity_index(sd)
70.1208244...
```

colour.quality

ColourRendering_Specification_CIE2017(name, ...)	Define the *CIE 2017 Colour Fidelity Index* (CFI) colour quality specification.
colour_fidelity_index_CIE2017(sd_test[, ...])	Return the *CIE 2017 Colour Fidelity Index* (CFI) R_f of given spectral distribution.
ColourQuality_Specification_ANSIIESTM3018(...)	Define the *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI) colour quality specification.
colour_fidelity_index_ANSIIESTM3018(sd_test)	Return the *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI) R_f of given spectral distribution.

colour.quality.ColourRendering_Specification_CIE2017

class colour.quality.**ColourRendering_Specification_CIE2017**(*name: str, sd_reference: colour.colorimetry.spectrum.SpectralDistribution, R_f: float, R_s: NDArrayFloat, CCT: float, D_uv: float, colorimetry_data: Tuple[colour.quality.cfi2017.DataColorimetry_TCS_CIE2017, colour.quality.cfi2017.DataColorimetry_TCS_CIE2017], delta_E_s: NDArrayFloat*)

Define the *CIE 2017 Colour Fidelity Index* (CFI) colour quality specification.

Parameters

- **name** (str) – Name of the test spectral distribution.
- **sd_reference** (colour.colorimetry.spectrum.SpectralDistribution) – Spectral distribution of the reference illuminant.
- **R_f** (float) – *CIE 2017 Colour Fidelity Index* (CFI) R_f.
- **R_s** (NDArrayFloat) – Individual *colour fidelity indexes* data for each sample.
- **CCT** (float) – Correlated colour temperature T_{cp}.
- **D_uv** (float) – Distance from the Planckian locus Δ_{uv}.
- **colorimetry_data** (Tuple[colour.quality.cfi2017.DataColorimetry_TCS_CIE2017, colour.quality.cfi2017.DataColorimetry_TCS_CIE2017]) – Colorimetry data for the test and reference computations.
- **delta_E_s** (NDArrayFloat) – Colour shifts of samples.

Return type None

__init__(*name: str, sd_reference: colour.colorimetry.spectrum.SpectralDistribution, R_f: float, R_s: NDArrayFloat, CCT: float, D_uv: float, colorimetry_data: Tuple[colour.quality.cfi2017.DataColorimetry_TCS_CIE2017, colour.quality.cfi2017.DataColorimetry_TCS_CIE2017], delta_E_s: NDArrayFloat*) → None

Parameters

- **name** (str) –
- **sd_reference** (colour.colorimetry.spectrum.SpectralDistribution) –
- **R_f** (float) –
- **R_s** (NDArrayFloat) –
- **CCT** (float) –
- **D_uv** (float) –
- **colorimetry_data** (Tuple[colour.quality.cfi2017.DataColorimetry_TCS_CIE2017, colour.quality.cfi2017.DataColorimetry_TCS_CIE2017]) –
- **delta_E_s** (NDArrayFloat) –

Return type None

Methods

`__init__`(name, sd_reference, R_f, R_s, CCT, ...)

Attributes

`name`
`sd_reference`
`R_f`
`R_s`
`CCT`
`D_uv`
`colorimetry_data`
`delta_E_s`

colour.quality.colour_fidelity_index_CIE2017

`colour.quality.colour_fidelity_index_CIE2017`(*sd_test:* colour.colorimetry.spectrum.SpectralDistribution, *additional_data:* bool = *False*) → float | *colour.quality.cfi2017.ColourRendering_Specification_CIE2017*

Return the *CIE 2017 Colour Fidelity Index* (CFI) R_f of given spectral distribution.

Parameters

- **sd_test** (`colour.colorimetry.spectrum.SpectralDistribution`) – Test spectral distribution.
- **additional_data** (`bool`) – Whether to output additional data.

Returns *CIE 2017 Colour Fidelity Index* (CFI) R_f.

Return type `float` or `colour.quality.ColourRendering_Specification_CIE2017`

References

[CIET19017]

Examples

```
>>> from colour.colorimetry import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS["FL2"]
>>> colour_fidelity_index_CIE2017(sd)
70.1208244...
```

colour.quality.ColourQuality_Specification_ANSIIESTM3018

class colour.quality.**ColourQuality_Specification_ANSIIESTM3018**(*name: str*, *sd_test: colour.colorimetry.spectrum.SpectralDistribution*, *sd_reference: colour.colorimetry.spectrum.SpectralDistribution*, *R_f: float*, *R_s: NDArrayFloat*, *CCT: float*, *D_uv: float*, *colorimetry_data: Tuple[colour.quality.cfi2017.DataColorimetry_TCS_CIE2017, colour.quality.cfi2017.DataColorimetry_TCS_CIE2017]*, *R_g: float*, *bins: NDArrayInt*, *averages_test: NDArrayFloat*, *averages_reference: NDArrayFloat*, *average_norms: NDArrayFloat*, *R_fs: NDArrayFloat*, *R_cs: NDArrayFloat*, *R_hs: NDArrayFloat*)

Define the *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI) colour quality specification.

Parameters

- **name** (str) – Name of the test spectral distribution.
- **sd_test** (colour.colorimetry.spectrum.SpectralDistribution) – Spectral distribution of the tested illuminant.
- **sd_reference** (colour.colorimetry.spectrum.SpectralDistribution) – Spectral distribution of the reference illuminant.
- **R_f** (float) – *Colour Fidelity Index* (CFI) R_f.
- **R_s** (NDArrayFloat) – Individual *colour fidelity indexes* data for each sample.
- **CCT** (float) – Correlated colour temperature T_{cp}.
- **D_uv** (float) – Distance from the Planckian locus Δ_{uv}.
- **colorimetry_data** (Tuple[colour.quality.cfi2017.DataColorimetry_TCS_CIE2017, colour.quality.cfi2017.DataColorimetry_TCS_CIE2017]) – Colorimetry data for the test and reference computations.
- **R_g** (float) – Gamut index R_g.
- **bins** (NDArrayInt) – List of 16 lists, each containing the indexes of colour samples that lie in the respective hue bin.
- **averages_test** (NDArrayFloat) – Averages of *CAM02-UCS* a', b' coordinates for each hue bin for test samples.
- **averages_reference** (NDArrayFloat) – Averages for reference samples.
- **average_norms** (NDArrayFloat) – Distance of averages for reference samples from the origin.

- **R_fs** (NDArrayFloat) – Local colour fidelities for each hue bin.
- **R_cs** (NDArrayFloat) – Local chromaticity shifts for each hue bin, in percents.
- **R_hs** (NDArrayFloat) – Local hue shifts for each hue bin.

Return type None

__init__(*name: str, sd_test: colour.colorimetry.spectrum.SpectralDistribution, sd_reference: colour.colorimetry.spectrum.SpectralDistribution, R_f: float, R_s: NDArrayFloat, CCT: float, D_uv: float, colorimetry_data: Tuple[colour.quality.cfi2017.DataColorimetry_TCS_CIE2017, colour.quality.cfi2017.DataColorimetry_TCS_CIE2017], R_g: float, bins: NDArrayInt, averages_test: NDArrayFloat, averages_reference: NDArrayFloat, average_norms: NDArrayFloat, R_fs: NDArrayFloat, R_cs: NDArrayFloat, R_hs: NDArrayFloat*) → None

Parameters

- **name** (str) –
- **sd_test** (colour.colorimetry.spectrum.SpectralDistribution) –
- **sd_reference** (colour.colorimetry.spectrum.SpectralDistribution) –
- **R_f** (float) –
- **R_s** (NDArrayFloat) –
- **CCT** (float) –
- **D_uv** (float) –
- **colorimetry_data** (Tuple[colour.quality.cfi2017.DataColorimetry_TCS_CIE2017, colour.quality.cfi2017.DataColorimetry_TCS_CIE2017]) –
- **R_g** (float) –
- **bins** (NDArrayInt) –
- **averages_test** (NDArrayFloat) –
- **averages_reference** (NDArrayFloat) –
- **average_norms** (NDArrayFloat) –
- **R_fs** (NDArrayFloat) –
- **R_cs** (NDArrayFloat) –
- **R_hs** (NDArrayFloat) –

Return type None

Methods

`__init__`(name, sd_test, sd_reference, R_f, ...)

Attributes

name
sd_test
sd_reference
R_f
R_s
CCT
D_uv
colorimetry_data
R_g
bins
averages_test
averages_reference
average_norms
R_fs
R_cs
R_hs

colour.quality.colour_fidelity_index_ANSIIESTM3018

colour.quality.**colour_fidelity_index_ANSIIESTM3018**(*sd_test: colour.colorimetry.spectrum.SpectralDistribution, additional_data: bool = False*) → float | *colour.quality.tm3018.ColourQuality_Specification_ANSIIESTM3018* | *colour.quality.cfi2017.ColourRendering_Specification_CIE2017*

Return the *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI) R_f of given spectral distribution.

Parameters

- **sd_test** (colour.colorimetry.spectrum.SpectralDistribution) – Test spectral distribution.
- **additional_data** (bool) – Whether to output additional data.

Returns *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI).

Return type float or colour.quality.ColourQuality_Specification_ANSIIESTM3018

References

[ANSIIESCCommittee18], [VincentJ17]

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS["FL2"]
>>> colour_fidelity_index_ANSIIESTM3018(sd)
70.1208244...
```

Colour Rendering Index

`colour`

`colour_rendering_index`(sd_test[, ...])	Return the *Colour Rendering Index* (CRI) Q_a of given spectral distribution.

colour.colour_rendering_index

`colour.`**`colour_rendering_index`**(*sd_test: colour.colorimetry.spectrum.SpectralDistribution, additional_data: bool = False*) → float | *colour.quality.cri.ColourRendering_Specification_CRI*

Return the *Colour Rendering Index* (CRI) Q_a of given spectral distribution.

Parameters

- **`sd_test`** (`colour.colorimetry.spectrum.SpectralDistribution`) – Test spectral distribution.
- **`additional_data`** (`bool`) – Whether to output additional data.

Returns *Colour Rendering Index* (CRI).

Return type `float` or `colour.quality.ColourRendering_Specification_CRI`

References

[OD08]

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS["FL2"]
>>> colour_rendering_index(sd)
64.2337241...
```

`colour.quality`

`ColourRendering_Specification_CRI`(name, Q_a, ...)	Define the *Colour Rendering Index* (CRI) colour quality specification.

colour.quality.ColourRendering_Specification_CRI

class colour.quality.**ColourRendering_Specification_CRI**(*name: str, Q_a: float, Q_as: Dict[int, colour.quality.cri.DataColourQualityScale_TCS], colorimetry_data: Tuple[Tuple[colour.quality.cri.DataColorimetry_TCS, ...], Tuple[colour.quality.cri.DataColorimetry_TCS, ...]]*)

Define the *Colour Rendering Index* (CRI) colour quality specification.

Parameters

- **name** (str) – Name of the test spectral distribution.
- **Q_a** (float) – *Colour Rendering Index* (CRI) Q_a.
- **Q_as** (Dict[int, colour.quality.cri.DataColourQualityScale_TCS]) – Individual *colour rendering indexes* data for each sample.
- **colorimetry_data** (Tuple[Tuple[colour.quality.cri.DataColorimetry_TCS, ...], Tuple[colour.quality.cri.DataColorimetry_TCS, ...]]) – Colorimetry data for the test and reference computations.

Return type None

References

[OD08]

__init__(*name: str, Q_a: float, Q_as: Dict[int, colour.quality.cri.DataColourQualityScale_TCS], colorimetry_data: Tuple[Tuple[colour.quality.cri.DataColorimetry_TCS, ...], Tuple[colour.quality.cri.DataColorimetry_TCS, ...]]*) → None

Parameters

- **name** (str) –
- **Q_a** (float) –
- **Q_as** (Dict[int, colour.quality.cri.DataColourQualityScale_TCS]) –
- **colorimetry_data** (Tuple[Tuple[colour.quality.cri.DataColorimetry_TCS, ...], Tuple[colour.quality.cri.DataColorimetry_TCS, ...]]) –

Return type None

Methods

__init__(name, Q_a, Q_as, colorimetry_data)	

Attributes

name
Q_a
Q_as
colorimetry_data

Colour Quality Scale

colour

COLOUR_QUALITY_SCALE_METHODS	Supported *Colour Quality Scale* (CQS) computation methods.
colour_quality_scale(sd_test[, ...])	Return the *Colour Quality Scale* (CQS) of given spectral distribution using given method.

colour.COLOUR_QUALITY_SCALE_METHODS

colour.**COLOUR_QUALITY_SCALE_METHODS = ('NIST CQS 7.4', 'NIST CQS 9.0')**

Supported *Colour Quality Scale* (CQS) computation methods.

References

[DO10], [OD08], [OD13]

colour.colour_quality_scale

colour.**colour_quality_scale**(*sd_test: colour.colorimetry.spectrum.SpectralDistribution, additional_data: bool = False, method: Union[Literal['NIST CQS 7.4', 'NIST CQS 9.0'], str] = 'NIST CQS 9.0'*) → float | *colour.quality.cqs.ColourRendering_Specification_CQS*

Return the *Colour Quality Scale* (CQS) of given spectral distribution using given method.

Parameters

- **sd_test** (colour.colorimetry.spectrum.SpectralDistribution) – Test spectral distribution.
- **additional_data** (bool) – Whether to output additional data.
- **method** (Union[Literal['NIST CQS 7.4', 'NIST CQS 9.0'], str]) – Computation method.

Returns *Colour Quality Scale* (CQS).

Return type float or colour.quality.ColourRendering_Specification_CQS

References

[DO10], [OD08], [OD13]

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS["FL2"]
>>> colour_quality_scale(sd)
64.1118220...
```

colour.quality

ColourRendering_Specification_CQS(name, Q_a, ...)	Define the *Colour Quality Scale* (CQS) colour rendering (quality) specification.

colour.quality.ColourRendering_Specification_CQS

class colour.quality.**ColourRendering_Specification_CQS**(*name: str, Q_a: float, Q_f: float, Q_p: float | None, Q_g: float, Q_d: float | None, Q_as: Dict[int, colour.quality.cqs.DataColourQualityScale_VS], colorimetry_data: Tuple[Tuple[colour.quality.cqs.DataColorimetry_VS, ...], Tuple[colour.quality.cqs.DataColorimetry_VS, ...]]*)

Define the *Colour Quality Scale* (CQS) colour rendering (quality) specification.

Parameters

- **name** (str) – Name of the test spectral distribution.
- **Q_a** (float) – Colour quality scale Q_a.
- **Q_f** (float) – 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** (float | None) – Colour preference scale Q_p similar to colour quality scale Q_a but placing additional weight on preference of object colour appearance, set to *None* in *NIST CQS 9.0* method. This metric is based on the notion that increases in chroma are generally preferred and should be rewarded.
- **Q_g** (float) – 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** (float | None) – Relative gamut area scale Q_d, set to *None* in *NIST CQS 9.0* method.
- **Q_as** (Dict[int, colour.quality.cqs.DataColourQualityScale_VS]) – Individual *Colour Quality Scale* (CQS) data for each sample.
- **colorimetry_data** (Tuple[Tuple[colour.quality.cqs.DataColorimetry_VS, ...], Tuple[colour.quality.cqs.DataColorimetry_VS, ...]]) – Colorimetry data for the test and reference computations.

Return type None

References

[DO10], [OD08], [OD13]

__init__(*name: str, Q_a: float, Q_f: float, Q_p: float | None, Q_g: float, Q_d: float | None, Q_as: Dict[int, colour.quality.cqs.DataColourQualityScale_VS], colorimetry_data: Tuple[Tuple[colour.quality.cqs.DataColorimetry_VS, ...], Tuple[colour.quality.cqs.DataColorimetry_VS, ...]]*) → None

Parameters

- **name** (str) –
- **Q_a** (float) –
- **Q_f** (float) –
- **Q_p** (float | None) –
- **Q_g** (float) –
- **Q_d** (float | None) –
- **Q_as** (Dict[int, colour.quality.cqs.DataColourQualityScale_VS]) –
- **colorimetry_data** (Tuple[Tuple[colour.quality.cqs.DataColorimetry_VS, ...], Tuple[colour.quality.cqs.DataColorimetry_VS, ...]]) –

Return type None

Methods

`__init__`(name, Q_a, Q_f, Q_p, Q_g, Q_d, ...)

Attributes

name
Q_a
Q_f
Q_p
Q_g
Q_d
Q_as
colorimetry_data

Academy Spectral Similarity Index (SSI)

`colour`

spectral_similarity_index(sd_test, sd_reference)	Return the *Academy Spectral Similarity Index* (SSI) of given test spectral distribution with given reference spectral distribution.

colour.spectral_similarity_index

colour.**spectral_similarity_index**(*sd_test: colour.colorimetry.spectrum.SpectralDistribution*, *sd_reference: colour.colorimetry.spectrum.SpectralDistribution*, *round_result: bool = True*) → NDArrayFloat

Return the *Academy Spectral Similarity Index* (SSI) of given test spectral distribution with given reference spectral distribution.

Parameters

- **sd_test** (colour.colorimetry.spectrum.SpectralDistribution) – Test spectral distribution.
- **sd_reference** (colour.colorimetry.spectrum.SpectralDistribution) – Reference spectral distribution.
- **round_result** (bool) – Whether to round the result/output. This is particularly useful when using SSI in an optimisation routine. Default is *True*.

Returns *Academy Spectral Similarity Index* (SSI).

Return type numpy.ndarray

References

[TheAoMPAaSciences20]

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd_test = SDS_ILLUMINANTS["C"]
>>> sd_reference = SDS_ILLUMINANTS["D65"]
>>> spectral_similarity_index(sd_test, sd_reference)
94.0
```

Spectral Recovery and Up-sampling

Reflectance Recovery

CIE XYZ Colourspace to Spectral

`colour`

XYZ_to_sd(XYZ[, method])	Recover the spectral distribution of given *CIE XYZ* tristimulus values using given method.
XYZ_TO_SD_METHODS	Supported spectral distribution recovery methods.

colour.XYZ_to_sd

colour.**XYZ_to_sd**(*XYZ: ArrayLike, method: Union[Literal['Jakob 2019', 'Mallett 2019', 'Meng 2015', 'Otsu 2018', 'Smits 1999'], str] = 'Meng 2015', **kwargs: Any*) → *colour.colorimetry.spectrum.SpectralDistribution*

Recover the spectral distribution of given *CIE XYZ* tristimulus values using given method.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values to recover the spectral distribution from.
- **method** (Union[Literal['Jakob 2019', 'Mallett 2019', 'Meng 2015', 'Otsu 2018', 'Smits 1999'], str]) – Computation method.
- **additional_data** – {colour.recovery.XYZ_to_sd_Jakob2019()}, If *True*, error will be returned alongside sd.
- **basis_functions** – {colour.recovery.RGB_to_sd_Mallett2019()}, Basis functions for the method. The default is to use the built-in *sRGB* basis functions, i.e. colour.recovery.MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019.
- **clip** – {colour.recovery.XYZ_to_sd_Otsu2018()}, If *True*, the default, values below zero and above unity in the recovered spectral distributions will be clipped. This ensures that the returned reflectance is physical and conserves energy, but will cause noticeable colour differences in case of very saturated colours.
- **cmfs** – {colour.recovery.XYZ_to_sd_Meng2015()}, Standard observer colour matching functions.
- **colourspace** – {colour.recovery.XYZ_to_sd_Jakob2019()}, *RGB* colourspace of the target colour. Note that no chromatic adaptation is performed between illuminant and the colourspace whitepoint.
- **dataset** – {colour.recovery.XYZ_to_sd_Otsu2018()}, Dataset to use for reconstruction. The default is to use the published data.
- **illuminant** – {colour.recovery.XYZ_to_sd_Jakob2019(), colour.recovery.XYZ_to_sd_Meng2015()}, Illuminant spectral distribution, default to *CIE Standard Illuminant D65*.
- **interval** – {colour.recovery.XYZ_to_sd_Meng2015()}, Wavelength λ_i range interval in nm. The smaller interval is, the longer the computations will be.
- **optimisation_kwargs** – {colour.recovery.XYZ_to_sd_Jakob2019(), colour.recovery.XYZ_to_sd_Meng2015()}, Parameters for scipy.optimize.minimize() and colour.recovery.find_coefficients_Jakob2019() definitions.
- **kwargs** (Any) –

Returns Recovered spectral distribution.

Return type colour.SpectralDistribution

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

- *Smits (1999)* method will internally convert given *CIE XYZ* tristimulus values to *sRGB* colourspace array assuming equal energy illuminant *E*.

References

[JH19], [MY19], [MSHD15], [OYH18], [Smi99]

Examples

Jakob and Hanika (2009) reflectance recovery:

```
>>> import numpy as np
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS, SpectralShape
>>> from colour.colorimetry import sd_to_XYZ_integration
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(
...     XYZ, method="Jakob 2019", cmfs=cmfs, illuminant=illuminant
... )
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 360.        ,    0.4893773...],
                      [ 370.        ,    0.3258214...],
                      [ 380.        ,    0.2147792...],
                      [ 390.        ,    0.1482413...],
                      [ 400.        ,    0.1086169...],
                      [ 410.        ,    0.0841255...],
                      [ 420.        ,    0.0683114...],
                      [ 430.        ,    0.0577144...],
                      [ 440.        ,    0.0504267...],
                      [ 450.        ,    0.0453552...],
                      [ 460.        ,    0.0418520...],
                      [ 470.        ,    0.0395259...],
                      [ 480.        ,    0.0381430...],
                      [ 490.        ,    0.0375741...],
                      [ 500.        ,    0.0377685...],
                      [ 510.        ,    0.0387432...],
                      [ 520.        ,    0.0405871...],
                      [ 530.        ,    0.0434783...],
                      [ 540.        ,    0.0477225...],
                      [ 550.        ,    0.0538256...],
                      [ 560.        ,    0.0626314...],
                      [ 570.        ,    0.0755869...],
```

(continues on next page)

(continued from previous page)

```
                              [ 580.          ,    0.0952675...],
                              [ 590.          ,    0.1264265...],
                              [ 600.          ,    0.1779272...],
                              [ 610.          ,    0.2649393...],
                              [ 620.          ,    0.4039779...],
                              [ 630.          ,    0.5832105...],
                              [ 640.          ,    0.7445440...],
                              [ 650.          ,    0.8499970...],
                              [ 660.          ,    0.9094792...],
                              [ 670.          ,    0.9425378...],
                              [ 680.          ,    0.9616376...],
                              [ 690.          ,    0.9732481...],
                              [ 700.          ,    0.9806562...],
                              [ 710.          ,    0.9855873...],
                              [ 720.          ,    0.9889903...],
                              [ 730.          ,    0.9914117...],
                              [ 740.          ,    0.9931801...],
                              [ 750.          ,    0.9945009...],
                              [ 760.          ,    0.9955066...],
                              [ 770.          ,    0.9962855...],
                              [ 780.          ,    0.9968976...]],
                             interpolator=SpragueInterpolator,
                             interpolator_kwargs={},
                             extrapolator=Extrapolator,
                             extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2066217...,  0.1220128...,  0.0513958...])
```

Mallett and Yuksel (2019) reflectance recovery:

```
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SPECTRAL_SHAPE_sRGB_MALLETT2019)
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(XYZ, method="Mallett 2019")
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 380.          ,    0.1735531...],
                      [ 385.          ,    0.1720357...],
                      [ 390.          ,    0.1677721...],
                      [ 395.          ,    0.1576605...],
                      [ 400.          ,    0.1372829...],
                      [ 405.          ,    0.1170849...],
                      [ 410.          ,    0.0895694...],
                      [ 415.          ,    0.0706232...],
                      [ 420.          ,    0.0585765...],
                      [ 425.          ,    0.0523959...],
                      [ 430.          ,    0.0497598...],
                      [ 435.          ,    0.0476057...],
                      [ 440.          ,    0.0465079...],
                      [ 445.          ,    0.0460337...],
                      [ 450.          ,    0.0455839...],
                      [ 455.          ,    0.0452872...],
```

(continues on next page)

(continued from previous page)

```
               [ 460.        ,    0.0450981...],
               [ 465.        ,    0.0448895...],
               [ 470.        ,    0.0449257...],
               [ 475.        ,    0.0448987...],
               [ 480.        ,    0.0446834...],
               [ 485.        ,    0.0441372...],
               [ 490.        ,    0.0417137...],
               [ 495.        ,    0.0373832...],
               [ 500.        ,    0.0357657...],
               [ 505.        ,    0.0348263...],
               [ 510.        ,    0.0341953...],
               [ 515.        ,    0.0337683...],
               [ 520.        ,    0.0334979...],
               [ 525.        ,    0.0332991...],
               [ 530.        ,    0.0331909...],
               [ 535.        ,    0.0332181...],
               [ 540.        ,    0.0333387...],
               [ 545.        ,    0.0334970...],
               [ 550.        ,    0.0337381...],
               [ 555.        ,    0.0341847...],
               [ 560.        ,    0.0346447...],
               [ 565.        ,    0.0353993...],
               [ 570.        ,    0.0367367...],
               [ 575.        ,    0.0392007...],
               [ 580.        ,    0.0445902...],
               [ 585.        ,    0.0625633...],
               [ 590.        ,    0.2965381...],
               [ 595.        ,    0.4215576...],
               [ 600.        ,    0.4347139...],
               [ 605.        ,    0.4385134...],
               [ 610.        ,    0.4385184...],
               [ 615.        ,    0.4385249...],
               [ 620.        ,    0.4374694...],
               [ 625.        ,    0.4384672...],
               [ 630.        ,    0.4368251...],
               [ 635.        ,    0.4340867...],
               [ 640.        ,    0.4303219...],
               [ 645.        ,    0.4243257...],
               [ 650.        ,    0.4159482...],
               [ 655.        ,    0.4057443...],
               [ 660.        ,    0.3919874...],
               [ 665.        ,    0.3742784...],
               [ 670.        ,    0.3518421...],
               [ 675.        ,    0.3240127...],
               [ 680.        ,    0.2955145...],
               [ 685.        ,    0.2625658...],
               [ 690.        ,    0.2343423...],
               [ 695.        ,    0.2174830...],
               [ 700.        ,    0.2060461...],
               [ 705.        ,    0.1977437...],
               [ 710.        ,    0.1916846...],
               [ 715.        ,    0.1861020...],
               [ 720.        ,    0.1823908...],
               [ 725.        ,    0.1807923...],
               [ 730.        ,    0.1795571...],
               [ 735.        ,    0.1785623...],
```

(continues on next page)

(continued from previous page)

```
                       [ 740.        ,    0.1775758...],
                       [ 745.        ,    0.1771614...],
                       [ 750.        ,    0.1767431...],
                       [ 755.        ,    0.1764319...],
                       [ 760.        ,    0.1762597...],
                       [ 765.        ,    0.1762209...],
                       [ 770.        ,    0.1761803...],
                       [ 775.        ,    0.1761195...],
                       [ 780.        ,    0.1760763...]],
                      interpolator=SpragueInterpolator,
                      interpolator_kwargs={},
                      extrapolator=Extrapolator,
                      extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
...
array([ 0.2065436...,  0.1219996...,  0.0513764...])
```

Meng (2015) reflectance recovery:

```
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(
...     XYZ, method="Meng 2015", cmfs=cmfs, illuminant=illuminant
... )
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 360.        ,    0.0762005...],
                      [ 370.        ,    0.0761792...],
                      [ 380.        ,    0.0761363...],
                      [ 390.        ,    0.0761194...],
                      [ 400.        ,    0.0762539...],
                      [ 410.        ,    0.0761671...],
                      [ 420.        ,    0.0754649...],
                      [ 430.        ,    0.0731519...],
                      [ 440.        ,    0.0676701...],
                      [ 450.        ,    0.0577800...],
                      [ 460.        ,    0.0441993...],
                      [ 470.        ,    0.0285064...],
                      [ 480.        ,    0.0138728...],
                      [ 490.        ,    0.0033585...],
                      [ 500.        ,    0.       ...],
                      [ 510.        ,    0.       ...],
                      [ 520.        ,    0.       ...],
                      [ 530.        ,    0.       ...],
                      [ 540.        ,    0.0055767...],
                      [ 550.        ,    0.0317581...],
                      [ 560.        ,    0.0754491...],
                      [ 570.        ,    0.1314115...],
                      [ 580.        ,    0.1937649...],
                      [ 590.        ,    0.2559311...],
                      [ 600.        ,    0.3123173...],
```

(continues on next page)

(continued from previous page)

```
                                   [ 610.          ,    0.3584966...],
                                   [ 620.          ,    0.3927335...],
                                   [ 630.          ,    0.4159458...],
                                   [ 640.          ,    0.4306660...],
                                   [ 650.          ,    0.4391040...],
                                   [ 660.          ,    0.4439497...],
                                   [ 670.          ,    0.4463618...],
                                   [ 680.          ,    0.4474625...],
                                   [ 690.          ,    0.4479868...],
                                   [ 700.          ,    0.4482116...],
                                   [ 710.          ,    0.4482800...],
                                   [ 720.          ,    0.4483472...],
                                   [ 730.          ,    0.4484251...],
                                   [ 740.          ,    0.4484633...],
                                   [ 750.          ,    0.4485071...],
                                   [ 760.          ,    0.4484969...],
                                   [ 770.          ,    0.4484853...],
                                   [ 780.          ,    0.4485134...]],
                                  interpolator=SpragueInterpolator,
                                  interpolator_kwargs={},
                                  extrapolator=Extrapolator,
                                  extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

Otsu, Yamamoto and Hachisuka (2018) reflectance recovery:

```
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SPECTRAL_SHAPE_OTSU2018)
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(
...     XYZ, method="Otsu 2018", cmfs=cmfs, illuminant=illuminant
... )
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 380.          ,    0.0601939...],
                      [ 390.          ,    0.0568063...],
                      [ 400.          ,    0.0517429...],
                      [ 410.          ,    0.0495841...],
                      [ 420.          ,    0.0502007...],
                      [ 430.          ,    0.0506489...],
                      [ 440.          ,    0.0510020...],
                      [ 450.          ,    0.0493782...],
                      [ 460.          ,    0.0468046...],
                      [ 470.          ,    0.0437132...],
                      [ 480.          ,    0.0416957...],
                      [ 490.          ,    0.0403783...],
                      [ 500.          ,    0.0405197...],
                      [ 510.          ,    0.0406031...],
                      [ 520.          ,    0.0416912...],
                      [ 530.          ,    0.0430956...],
                      [ 540.          ,    0.0444474...],
```

(continues on next page)

(continued from previous page)

```
                          [ 550.        ,    0.0459336...],
                          [ 560.        ,    0.0507631...],
                          [ 570.        ,    0.0628967...],
                          [ 580.        ,    0.0844661...],
                          [ 590.        ,    0.1334277...],
                          [ 600.        ,    0.2262428...],
                          [ 610.        ,    0.3599330...],
                          [ 620.        ,    0.4885571...],
                          [ 630.        ,    0.5752546...],
                          [ 640.        ,    0.6193023...],
                          [ 650.        ,    0.6450744...],
                          [ 660.        ,    0.6610548...],
                          [ 670.        ,    0.6688673...],
                          [ 680.        ,    0.6795426...],
                          [ 690.        ,    0.6887933...],
                          [ 700.        ,    0.7003469...],
                          [ 710.        ,    0.7084128...],
                          [ 720.        ,    0.7154674...],
                          [ 730.        ,    0.7234334...]],
                         interpolator=SpragueInterpolator,
                         interpolator_kwargs={},
                         extrapolator=Extrapolator,
                         extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065494...,  0.1219712...,  0.0514002...])
```

Smits (1999) reflectance recovery:

```
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS["E"].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(XYZ, method="Smits 1999")
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 380.        ,    0.0787830...],
                      [ 417.7778    ,    0.0622018...],
                      [ 455.5556    ,    0.0446206...],
                      [ 493.3333    ,    0.0352220...],
                      [ 531.1111    ,    0.0324149...],
                      [ 568.8889    ,    0.0330105...],
                      [ 606.6667    ,    0.3207115...],
                      [ 644.4444    ,    0.3836164...],
                      [ 682.2222    ,    0.3836164...],
                      [ 720.        ,    0.3835649...]],
                     interpolator=LinearInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.1894770...,  0.1126470...,  0.0474420...])
```

colour.XYZ_TO_SD_METHODS

colour.**XYZ_TO_SD_METHODS = CanonicalMapping({'Jakob 2019': ..., 'Mallett 2019': ..., 'Meng 2015': ..., 'Otsu 2018': ..., 'Smits 1999': ...})**

Supported spectral distribution recovery methods.

References

[JH19], [MY19], [MSHD15], [Smi99]

Jakob and Hanika (2019)

colour.recovery

XYZ_to_sd_Jakob2019(XYZ[, cmfs, illuminant, ...])	Recover the spectral distribution of given *CIE XYZ* tristimulus values using *Jakob and Hanika (2019)* method.
LUT3D_Jakob2019()	Define a class for working with pre-computed lookup tables for the *Jakob and Hanika (2019)* spectral upsampling method.

colour.recovery.XYZ_to_sd_Jakob2019

colour.recovery.**XYZ_to_sd_Jakob2019**(*XYZ: ArrayLike*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*, *illuminant: colour.colorimetry.spectrum.SpectralDistribution | None = None*, *optimisation_kwargs: dict | None = None*, *additional_data: bool = False*) → Union[Tuple[*colour.colorimetry.spectrum.SpectralDistribution*, float], *colour.colorimetry.spectrum.SpectralDistribution*]

Recover the spectral distribution of given *CIE XYZ* tristimulus values using *Jakob and Hanika (2019)* method.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values to recover the spectral distribution from.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Standard Illuminant D65*.
- **optimisation_kwargs** (dict | None) – Parameters for colour.recovery.find_coefficients_Jakob2019() definition.
- **additional_data** (bool) – If *True*, error will be returned alongside the recovered spectral distribution.

Returns Tuple of recovered spectral distribution and ΔE_{76} between the target colour and the colour corresponding to the computed coefficients or recovered spectral distribution.

Return type tuple or colour.SpectralDistribution

References

[JH19]

Examples

```
>>> from colour import (
...     CCS_ILLUMINANTS,
...     MSDS_CMFS,
...     SDS_ILLUMINANTS,
...     XYZ_to_sRGB,
... )
>>> from colour.colorimetry import sd_to_XYZ_integration
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd_Jakob2019(XYZ, cmfs, illuminant)
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 360.        ,    0.4893773...],
                      [ 370.        ,    0.3258214...],
                      [ 380.        ,    0.2147792...],
                      [ 390.        ,    0.1482413...],
                      [ 400.        ,    0.1086169...],
                      [ 410.        ,    0.0841255...],
                      [ 420.        ,    0.0683114...],
                      [ 430.        ,    0.0577144...],
                      [ 440.        ,    0.0504267...],
                      [ 450.        ,    0.0453552...],
                      [ 460.        ,    0.0418520...],
                      [ 470.        ,    0.0395259...],
                      [ 480.        ,    0.0381430...],
                      [ 490.        ,    0.0375741...],
                      [ 500.        ,    0.0377685...],
                      [ 510.        ,    0.0387432...],
                      [ 520.        ,    0.0405871...],
                      [ 530.        ,    0.0434783...],
                      [ 540.        ,    0.0477225...],
                      [ 550.        ,    0.0538256...],
                      [ 560.        ,    0.0626314...],
                      [ 570.        ,    0.0755869...],
                      [ 580.        ,    0.0952675...],
                      [ 590.        ,    0.1264265...],
                      [ 600.        ,    0.1779272...],
                      [ 610.        ,    0.2649393...],
                      [ 620.        ,    0.4039779...],
                      [ 630.        ,    0.5832105...],
                      [ 640.        ,    0.7445440...],
                      [ 650.        ,    0.8499970...],
                      [ 660.        ,    0.9094792...],
```

(continues on next page)

(continued from previous page)

```
                                      [ 670.        ,    0.9425378...],
                                      [ 680.        ,    0.9616376...],
                                      [ 690.        ,    0.9732481...],
                                      [ 700.        ,    0.9806562...],
                                      [ 710.        ,    0.9855873...],
                                      [ 720.        ,    0.9889903...],
                                      [ 730.        ,    0.9914117...],
                                      [ 740.        ,    0.9931801...],
                                      [ 750.        ,    0.9945009...],
                                      [ 760.        ,    0.9955066...],
                                      [ 770.        ,    0.9962855...],
                                      [ 780.        ,    0.9968976...]],
                                     SpragueInterpolator,
                                     {},
                                     Extrapolator,
                                     {'method': 'Constant', 'left': None, 'right': None})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2066217...,  0.1220128...,  0.0513958...])
```

colour.recovery.LUT3D_Jakob2019

class colour.recovery.**LUT3D_Jakob2019**

Define a class for working with pre-computed lookup tables for the *Jakob and Hanika (2019)* spectral upsampling method. It allows significant time savings by performing the expensive numerical optimisation ahead of time and storing the results in a file.

The file format is compatible with the code and **.coeff* files in the supplemental material published alongside the article. They are directly available from Colour - Datasets under the record *4050598*.

Attributes

- `size`
- `lightness_scale`
- `coefficients`
- `interpolator`

Methods

- `__init__()`
- `generate()`
- `RGB_to_coefficients()`
- `RGB_to_sd()`
- `read()`
- `write()`

References

[JH19]

Examples

```
>>> import os
>>> import colour
>>> from colour import CCS_ILLUMINANTS, SDS_ILLUMINANTS, MSDS_CMFS
>>> from colour.colorimetry import sd_to_XYZ_integration
>>> from colour.models import RGB_COLOURSPACE_sRGB
>>> from colour.utilities import numpy_print_options
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> LUT = LUT3D_Jakob2019()
>>> LUT.generate(RGB_COLOURSPACE_sRGB, cmfs, illuminant, 3, lambda x: x)
>>> path = os.path.join(
...     colour.__path__[0],
...     "recovery",
...     "tests",
...     "resources",
...     "sRGB_Jakob2019.coeff",
... )
>>> LUT.write(path)
>>> LUT.read(path)
>>> RGB = np.array([0.70573936, 0.19248266, 0.22354169])
>>> with numpy_print_options(suppress=True):
...     LUT.RGB_to_sd(RGB, cmfs.shape)
...
SpectralDistribution([[ 360.        ,    0.7666803...],
                      [ 370.        ,    0.6251547...],
                      [ 380.        ,    0.4584310...],
                      [ 390.        ,    0.3161633...],
                      [ 400.        ,    0.2196155...],
                      [ 410.        ,    0.1596575...],
                      [ 420.        ,    0.1225525...],
                      [ 430.        ,    0.0989784...],
                      [ 440.        ,    0.0835782...],
                      [ 450.        ,    0.0733535...],
                      [ 460.        ,    0.0666049...],
                      [ 470.        ,    0.0623569...],
                      [ 480.        ,    0.06006  ...],
                      [ 490.        ,    0.0594383...],
                      [ 500.        ,    0.0604201...],
                      [ 510.        ,    0.0631195...],
                      [ 520.        ,    0.0678648...],
                      [ 530.        ,    0.0752834...],
                      [ 540.        ,    0.0864790...],
                      [ 550.        ,    0.1033773...],
                      [ 560.        ,    0.1293883...],
                      [ 570.        ,    0.1706018...],
                      [ 580.        ,    0.2374178...],
```

(continues on next page)

(continued from previous page)

```
           [ 590.        ,    0.3439472...],
           [ 600.        ,    0.4950548...],
           [ 610.        ,    0.6604253...],
           [ 620.        ,    0.7914669...],
           [ 630.        ,    0.8738724...],
           [ 640.        ,    0.9213216...],
           [ 650.        ,    0.9486880...],
           [ 660.        ,    0.9650550...],
           [ 670.        ,    0.9752838...],
           [ 680.        ,    0.9819499...],
           [ 690.        ,    0.9864585...],
           [ 700.        ,    0.9896073...],
           [ 710.        ,    0.9918680...],
           [ 720.        ,    0.9935302...],
           [ 730.        ,    0.9947778...],
           [ 740.        ,    0.9957312...],
           [ 750.        ,    0.9964714...],
           [ 760.        ,    0.9970543...],
           [ 770.        ,    0.9975190...],
           [ 780.        ,    0.9978936...]],
          SpragueInterpolator,
          {},
          Extrapolator,
          {'method': 'Constant', 'left': None, 'right': None})
```

Return type None

`__init__`() → None

Return type None

Methods

`RGB_to_coefficients`(RGB)	Look up a given *RGB* colourspace array and return corresponding coefficients.
`RGB_to_sd`(RGB[, shape])	Look up a given *RGB* colourspace array and return the corresponding spectral distribution.
`__init__`()	
`generate`(colourspace[, cmfs, illuminant, ...])	Generate the lookup table data for given *RGB* colourspace, colour matching functions, illuminant and given size.
`read`(path)	Load a lookup table from a **.coeff* file.
`write`(path)	Write the lookup table to a **.coeff* file.

Attributes

coefficients	Getter property for the *Jakob and Hanika (2019)* interpolator coefficients.
interpolator	Getter property for the *Jakob and Hanika (2019)* interpolator.
lightness_scale	Getter property for the *Jakob and Hanika (2019)* interpolator lightness scale.
size	Getter property for the *Jakob and Hanika (2019)* interpolator size, i.e. the samples count on one side of the 3D table.

Ancillary Objects

colour.recovery

sd_Jakob2019(coefficients[, shape])	Return a spectral distribution following the spectral model given by *Jakob and Hanika (2019)*.
find_coefficients_Jakob2019(XYZ[, cmfs, ...])	Compute the coefficients for *Jakob and Hanika (2019)* reflectance spectral model.

colour.recovery.sd_Jakob2019

colour.recovery.**sd_Jakob2019**(*coefficients: ArrayLike*, *shape: colour.colorimetry.spectrum.SpectralShape = SPECTRAL_SHAPE_JAKOB2019*) → *colour.colorimetry.spectrum.SpectralDistribution*

Return a spectral distribution following the spectral model given by *Jakob and Hanika (2019)*.

Parameters

- **coefficients** (ArrayLike) – Dimensionless coefficients for *Jakob and Hanika (2019)* reflectance spectral model.
- **shape** (colour.colorimetry.spectrum.SpectralShape) – Shape used by the spectral distribution.

Returns *Jakob and Hanika (2019)* spectral distribution.

Return type colour.SpectralDistribution

References

[JH19]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_Jakob2019([-9e-05, 8.5e-02, -20], SpectralShape(400, 700, 20))
...
...
SpectralDistribution([[ 400.        ,    0.3143046...],
                      [ 420.        ,    0.4133320...],
                      [ 440.        ,    0.4880034...],
                      [ 460.        ,    0.5279562...],
```

(continues on next page)

(continued from previous page)

```
            [ 480.         ,    0.5319346...],
            [ 500.         ,    0.5      ...],
            [ 520.         ,    0.4326202...],
            [ 540.         ,    0.3373544...],
            [ 560.         ,    0.2353056...],
            [ 580.         ,    0.1507665...],
            [ 600.         ,    0.0931332...],
            [ 620.         ,    0.0577434...],
            [ 640.         ,    0.0367011...],
            [ 660.         ,    0.0240879...],
            [ 680.         ,    0.0163316...],
            [ 700.         ,    0.0114118...]],
           SpragueInterpolator,
           {},
           Extrapolator,
           {'method': 'Constant', 'left': None, 'right': None})
```

colour.recovery.find_coefficients_Jakob2019

colour.recovery.**find_coefficients_Jakob2019**(*XYZ: ArrayLike, cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None, illuminant: colour.colorimetry.spectrum.SpectralDistribution | None = None, coefficients_0: ArrayLike = zeros(3), max_error: float = JND_CIE1976 / 100, dimensionalise: bool = True*) → Tuple[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ceae260d0>, float]

Compute the coefficients for *Jakob and Hanika (2019)* reflectance spectral model.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values to find the coefficients for.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Standard Illuminant D65*.
- **coefficients_0** (ArrayLike) – Starting coefficients for the solver.
- **max_error** (float) – Maximal acceptable error. Set higher to save computational time. If *None*, the solver will keep going until it is very close to the minimum. The default is ACCEPTABLE_DELTA_E.
- **dimensionalise** (bool) – If *True*, returned coefficients are dimensionful and will not work correctly if fed back as coefficients_0. The default is *True*.

Returns Tuple of computed coefficients that best fit the given colour and ΔE_{76} between the target colour and the colour corresponding to the computed coefficients.

Return type tuple

References

[JH19]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> find_coefficients_Jakob2019(XYZ)
(array([  1.3723791...e-04,  -1.3514399...e-01,   3.0838973...e+01]), 0.0141941...)
```

Mallett and Yuksel (2019)

`colour.recovery`

`RGB_to_sd_Mallett2019`(RGB[, basis_functions])	Recover the spectral distribution of given *RGB* colourspace array using *Mallett and Yuksel (2019)* method.

colour.recovery.RGB_to_sd_Mallett2019

`colour.recovery.`**`RGB_to_sd_Mallett2019`**(*RGB: ArrayLike, basis_functions:* colour.colorimetry.spectrum.MultiSpectralDistributions = *MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019*) → *colour.colorimetry.spectrum.SpectralDistribution*

Recover the spectral distribution of given *RGB* colourspace array using *Mallett and Yuksel (2019)* method.

Parameters

- **`RGB`** (`ArrayLike`) – *RGB* colourspace array.
- **`basis_functions`** (`colour.colorimetry.spectrum.MultiSpectralDistributions`) – Basis functions for the method. The default is to use the built-in *sRGB* basis functions, i.e. `colour.recovery.MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019`.

Returns Recovered reflectance.

Return type `colour.SpectralDistribution`

References

[MY19]

Notes

- In-addition to the *BT.709* primaries used by the *sRGB* colourspace, [MY19] tried *BT.2020*, *P3 D65*, *Adobe RGB 1998*, *NTSC (1987)*, *Pal/Secam*, *ProPhoto RGB*, and *Adobe Wide Gamut RGB* primaries, every one of which encompasses a larger (albeit not-always-enveloping) set of *CIE L*a*b** colours than BT.709. Of these, only *Pal/Secam* produces a feasible basis, which is relatively unsurprising since it is very similar to *BT.709*, whereas the others are significantly larger.

Examples

```
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS, XYZ_to_sRGB
>>> from colour.colorimetry import sd_to_XYZ_integration
>>> from colour.recovery import SPECTRAL_SHAPE_sRGB_MALLETT2019
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> RGB = XYZ_to_sRGB(XYZ, apply_cctf_encoding=False)
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SPECTRAL_SHAPE_sRGB_MALLETT2019)
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> sd = RGB_to_sd_Mallett2019(RGB)
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 380.        ,    0.1735531...],
                      [ 385.        ,    0.1720357...],
                      [ 390.        ,    0.1677721...],
                      [ 395.        ,    0.1576605...],
                      [ 400.        ,    0.1372829...],
                      [ 405.        ,    0.1170849...],
                      [ 410.        ,    0.0895694...],
                      [ 415.        ,    0.0706232...],
                      [ 420.        ,    0.0585765...],
                      [ 425.        ,    0.0523959...],
                      [ 430.        ,    0.0497598...],
                      [ 435.        ,    0.0476057...],
                      [ 440.        ,    0.0465079...],
                      [ 445.        ,    0.0460337...],
                      [ 450.        ,    0.0455839...],
                      [ 455.        ,    0.0452872...],
                      [ 460.        ,    0.0450981...],
                      [ 465.        ,    0.0448895...],
                      [ 470.        ,    0.0449257...],
                      [ 475.        ,    0.0448987...],
                      [ 480.        ,    0.0446834...],
                      [ 485.        ,    0.0441372...],
                      [ 490.        ,    0.0417137...],
                      [ 495.        ,    0.0373832...],
                      [ 500.        ,    0.0357657...],
                      [ 505.        ,    0.0348263...],
                      [ 510.        ,    0.0341953...],
                      [ 515.        ,    0.0337683...],
                      [ 520.        ,    0.0334979...],
                      [ 525.        ,    0.0332991...],
                      [ 530.        ,    0.0331909...],
                      [ 535.        ,    0.0332181...],
                      [ 540.        ,    0.0333387...],
                      [ 545.        ,    0.0334970...],
                      [ 550.        ,    0.0337381...],
                      [ 555.        ,    0.0341847...],
                      [ 560.        ,    0.0346447...],
                      [ 565.        ,    0.0353993...],
                      [ 570.        ,    0.0367367...],
```

(continues on next page)

(continued from previous page)

```
                      [ 575.        ,    0.0392007...],
                      [ 580.        ,    0.0445902...],
                      [ 585.        ,    0.0625633...],
                      [ 590.        ,    0.2965381...],
                      [ 595.        ,    0.4215576...],
                      [ 600.        ,    0.4347139...],
                      [ 605.        ,    0.4385134...],
                      [ 610.        ,    0.4385184...],
                      [ 615.        ,    0.4385249...],
                      [ 620.        ,    0.4374694...],
                      [ 625.        ,    0.4384672...],
                      [ 630.        ,    0.4368251...],
                      [ 635.        ,    0.4340867...],
                      [ 640.        ,    0.4303219...],
                      [ 645.        ,    0.4243257...],
                      [ 650.        ,    0.4159482...],
                      [ 655.        ,    0.4057443...],
                      [ 660.        ,    0.3919874...],
                      [ 665.        ,    0.3742784...],
                      [ 670.        ,    0.3518421...],
                      [ 675.        ,    0.3240127...],
                      [ 680.        ,    0.2955145...],
                      [ 685.        ,    0.2625658...],
                      [ 690.        ,    0.2343423...],
                      [ 695.        ,    0.2174830...],
                      [ 700.        ,    0.2060461...],
                      [ 705.        ,    0.1977437...],
                      [ 710.        ,    0.1916846...],
                      [ 715.        ,    0.1861020...],
                      [ 720.        ,    0.1823908...],
                      [ 725.        ,    0.1807923...],
                      [ 730.        ,    0.1795571...],
                      [ 735.        ,    0.1785623...],
                      [ 740.        ,    0.1775758...],
                      [ 745.        ,    0.1771614...],
                      [ 750.        ,    0.1767431...],
                      [ 755.        ,    0.1764319...],
                      [ 760.        ,    0.1762597...],
                      [ 765.        ,    0.1762209...],
                      [ 770.        ,    0.1761803...],
                      [ 775.        ,    0.1761195...],
                      [ 780.        ,    0.1760763...]],
                     SpragueInterpolator,
                     {},
                     Extrapolator,
                     {'method': 'Constant', 'left': None, 'right': None})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
...
array([ 0.2065436...,  0.1219996...,  0.0513764...])
```

Ancillary Objects

`colour.recovery`

MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019	the base object for multi spectral computations.
SPECTRAL_SHAPE_sRGB_MALLETT2019	Shape for *Mallett and Yuksel (2019) sRGB* colourspace basis functions: (380, 780, 5).
spectral_primary_decomposition_Mallett2019(...)	Perform the spectral primary decomposition as described in *Mallett and Yuksel (2019)* for given *RGB* colourspace.

colour.recovery.MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019

colour.recovery.**MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019 = MultiSpectralDistributions(name='Basis Functions - sRGB - Mallett 2019', ...)**

the base object for multi spectral computations. It is used to model colour matching functions, display primaries, camera sensitivities, etc. . .

The multi-spectral distributions will be initialised according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable. Extrapolation is performed according to *CIE 167:2005* recommendation.

Important: Specific documentation about getting, setting, indexing and slicing the multi-spectral power distributions values is available in the *Spectral Representation and Continuous Signal* section.

Parameters

- **data** – Data to be stored in the multi-spectral distributions.
- **domain** – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths property.
- **labels** – Names to use for the colour.SpectralDistribution class instances.
- **extrapolator** – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_kwargs** – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **interpolator** – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **name** – Multi-spectral distributions name.
- **display_labels** – Multi-spectral distributions labels for figures, default to colour.MultiSpectralDistributions.labels property value.

Warning: The *Cubic Spline* method might produce unexpected results with exceptionally noisy or non-uniformly spaced data.

Attributes

- display_name
- display_labels
- wavelengths
- values
- shape

Methods

- __init__()
- interpolate()
- extrapolate()
- align()
- trim()
- normalise()
- to_sds()

References

[CIET13805a], [CIET13805d], [CIET14804c]

Examples

Instantiating the multi-spectral distributions with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900),
... }
>>> labels = ("x_bar", "y_bar", "z_bar")
>>> with numpy_print_options(suppress=True):
...     MultiSpectralDistributions(data, labels=labels)
...
...
MultiSpectral...([[ 500.     ,    0.0049 ,    0.323  ,    0.272  ],
             ...  [ 510.     ,    0.0093 ,    0.503  ,    0.1582 ],
             ...  [ 520.     ,    0.06327,    0.71   ,    0.07825],
             ...  [ 530.     ,    0.1655 ,    0.862  ,    0.04216],
             ...  [ 540.     ,    0.2904 ,    0.954  ,    0.0203 ],
             ...  [ 550.     ,    0.43345,    0.99495,    0.00875],
             ...  [ 560.     ,    0.5945 ,    0.995  ,    0.0039 ]],
             ... [...'x_bar', ...'y_bar', ...'z_bar'],
             ... SpragueInterpolator,
```

(continues on next page)

(continued from previous page)

```
            ... {},
            ... Extrapolator,
            ... {'method': 'Constant', 'left': None, 'right': None})
```

Instantiating a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> with numpy_print_options(suppress=True):
...     MultiSpectralDistributions(data, labels=labels)
...
...
MultiSpectral...([[ 500.     ,    0.0049 ,    0.323  ,    0.272  ],
            ...  [ 510.     ,    0.0093 ,    0.503  ,    0.1582 ],
            ...  [ 511.     ,    0.00314,    0.31416,    0.03142],
            ...  [ 520.     ,    0.06327,    0.71   ,    0.07825],
            ...  [ 530.     ,    0.1655 ,    0.862  ,    0.04216],
            ...  [ 540.     ,    0.2904 ,    0.954  ,    0.0203 ],
            ...  [ 550.     ,    0.43345,    0.99495,    0.00875],
            ...  [ 560.     ,    0.5945 ,    0.995  ,    0.0039 ]],
            ... [...'x_bar', ...'y_bar', ...'z_bar'],
            ... CubicSplineInterpolator,
            ... {},
            ... Extrapolator,
            ... {'method': 'Constant', 'left': None, 'right': None})
```

Instantiation with a *Pandas DataFrame*:

```
>>> from colour.utilities import is_pandas_installed
>>> if is_pandas_installed():
...     from pandas import DataFrame
...
...     x_bar = [data[key][0] for key in sorted(data.keys())]
...     y_bar = [data[key][1] for key in sorted(data.keys())]
...     z_bar = [data[key][2] for key in sorted(data.keys())]
...     print(
...         MultiSignals(
...             DataFrame(
...                 dict(zip(labels, [x_bar, y_bar, z_bar])), data.keys()
...             )
...         )
...     )
...
[[  5.0000000...e+02   4.9000000...e-03   3.2300000...e-01   2.7200000...e-01]
 [  5.1000000...e+02   9.3000000...e-03   5.0300000...e-01   1.5820000...e-01]
 [  5.2000000...e+02   3.1400000...e-03   3.1416000...e-01   3.1420000...e-02]
 [  5.3000000...e+02   6.3270000...e-02   7.1000000...e-01   7.8250000...e-02]
 [  5.4000000...e+02   1.6550000...e-01   8.6200000...e-01   4.2160000...e-02]
 [  5.5000000...e+02   2.9040000...e-01   9.5400000...e-01   2.0300000...e-02]
 [  5.6000000...e+02   4.3345000...e-01   9.9495000...e-01   8.7500000...e-03]
 [  5.1100000...e+02   5.9450000...e-01   9.9500000...e-01   3.9000000...e-03]]
```

Type Define the multi-spectral distributions

colour.recovery.SPECTRAL_SHAPE_sRGB_MALLETT2019

colour.recovery.**SPECTRAL_SHAPE_sRGB_MALLETT2019 = SpectralShape(380, 780, 5)**

Shape for *Mallett and Yuksel (2019) sRGB* colourspace basis functions: (380, 780, 5).

References

[MY19]

colour.recovery.spectral_primary_decomposition_Mallett2019

colour.recovery.**spectral_primary_decomposition_Mallett2019**(*colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace, cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None, illuminant: colour.colorimetry.spectrum.SpectralDistribution | None = None, metric: Callable = np.linalg.norm, metric_args: tuple = (), optimisation_kwargs: dict | None = None*) → *colour.colorimetry.spectrum.MultiSpectralDistributions*

Perform the spectral primary decomposition as described in *Mallett and Yuksel (2019)* for given *RGB* colourspace.

Parameters

- **colourspace** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace`) – *RGB* colourspace.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | None`) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (`colour.colorimetry.spectrum.SpectralDistribution | None`) – Illuminant spectral distribution, default to *CIE Standard Illuminant D65*.
- **metric** (`Callable`) – Function to be minimised, i.e. the objective function.

  ```
  metric(basis, *metric_args) -> float
  ```

  where `basis` is three reflectances concatenated together, each with a shape matching `shape`.
- **metric_args** (`tuple`) – Additional arguments passed to `metric`.
- **optimisation_kwargs** (`dict | None`) – Parameters for `scipy.optimize.minimize()` definition.

Returns Basis functions for given *RGB* colourspace.

Return type `colour.MultiSpectralDistributions`

References

[MY19]

Notes

- In-addition to the *BT.709* primaries used by the *sRGB* colourspace, [MY19] tried *BT.2020*, *P3 D65*, *Adobe RGB 1998*, *NTSC (1987)*, *Pal/Secam*, *ProPhoto RGB*, and *Adobe Wide Gamut RGB* primaries, every one of which encompasses a larger (albeit not-always-enveloping) set of *CIE L*a*b** colours than BT.709. Of these, only *Pal/Secam* produces a feasible basis, which is relatively unsurprising since it is very similar to *BT.709*, whereas the others are significantly larger.

Examples

```
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS, SpectralShape
>>> from colour.models import RGB_COLOURSPACE_PAL_SECAM
>>> from colour.utilities import numpy_print_options
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> msds = spectral_primary_decomposition_Mallett2019(
...     RGB_COLOURSPACE_PAL_SECAM,
...     cmfs,
...     illuminant,
...     optimisation_kwargs={"options": {"ftol": 1e-5}},
... )
>>> with numpy_print_options(suppress=True):
...     print(msds)
...
[[ 360.          0.3395134...   0.3400214...   0.3204650...]
 [ 370.          0.3355246...   0.3338028...   0.3306724...]
 [ 380.          0.3376707...   0.3185578...   0.3437715...]
 [ 390.          0.3178866...   0.3351754...   0.3469378...]
 [ 400.          0.3045154...   0.3248376...   0.3706469...]
 [ 410.          0.2935652...   0.2919463...   0.4144884...]
 [ 420.          0.1875740...   0.1853729...   0.6270530...]
 [ 430.          0.0167983...   0.054483 ...   0.9287186...]
 [ 440.          0.       ...   0.       ...   1.       ...]
 [ 450.          0.       ...   0.       ...   1.       ...]
 [ 460.          0.       ...   0.       ...   1.       ...]
 [ 470.          0.       ...   0.0458044...   0.9541955...]
 [ 480.          0.       ...   0.2960917...   0.7039082...]
 [ 490.          0.       ...   0.5042592...   0.4957407...]
 [ 500.          0.       ...   0.6655795...   0.3344204...]
 [ 510.          0.       ...   0.8607541...   0.1392458...]
 [ 520.          0.       ...   0.9999998...   0.0000001...]
 [ 530.          0.       ...   1.       ...   0.       ...]
 [ 540.          0.       ...   1.       ...   0.       ...]
 [ 550.          0.       ...   1.       ...   0.       ...]
 [ 560.          0.       ...   0.9924229...   0.       ...]
 [ 570.          0.       ...   0.9970703...   0.0025673...]
 [ 580.          0.0396002...   0.9028231...   0.0575766...]
```

(continues on next page)

(continued from previous page)

```
[ 590.          0.7058973...   0.2941026...   0.        ...]
[ 600.          1.        ...  0.        ...  0.        ...]
[ 610.          1.        ...  0.        ...  0.        ...]
[ 620.          1.        ...  0.        ...  0.        ...]
[ 630.          1.        ...  0.        ...  0.        ...]
[ 640.          0.9835925...   0.0100166...   0.0063908...]
[ 650.          0.7878949...   0.1265097...   0.0855953...]
[ 660.          0.5987994...   0.2051062...   0.1960942...]
[ 670.          0.4724493...   0.2649623...   0.2625883...]
[ 680.          0.3989806...   0.3007488...   0.3002704...]
[ 690.          0.3666586...   0.3164003...   0.3169410...]
[ 700.          0.3497806...   0.3242863...   0.3259329...]
[ 710.          0.3563736...   0.3232441...   0.3203822...]
[ 720.          0.3362624...   0.3326209...   0.3311165...]
[ 730.          0.3245015...   0.3365982...   0.3389002...]
[ 740.          0.3335520...   0.3320670...   0.3343808...]
[ 750.          0.3441287...   0.3291168...   0.3267544...]
[ 760.          0.3343705...   0.3330132...   0.3326162...]
[ 770.          0.3274633...   0.3305704...   0.3419662...]
[ 780.          0.3475263...   0.3262331...   0.3262404...]]
```

Meng, Simon and Hanika (2015)

`colour.recovery`

`XYZ_to_sd_Meng2015`(XYZ[, cmfs, illuminant, ...])	Recover the spectral distribution of given *CIE XYZ* tristimulus values using *Meng et al. (2015)* method.

colour.recovery.XYZ_to_sd_Meng2015

`colour.recovery.`**`XYZ_to_sd_Meng2015`**(*XYZ: ArrayLike, cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None, illuminant: colour.colorimetry.spectrum.SpectralDistribution | None = None, optimisation_kwargs: dict | None = None*) → *colour.colorimetry.spectrum.SpectralDistribution*

Recover the spectral distribution of given *CIE XYZ* tristimulus values using *Meng et al. (2015)* method.

Parameters

- **XYZ** (`ArrayLike`) – *CIE XYZ* tristimulus values to recover the spectral distribution from.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | None`) – Standard observer colour matching functions. The wavelength λ_i range interval of the colour matching functions affects directly the time the computations take. The current default interval of 5 is a good compromise between precision and time spent, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (`colour.colorimetry.spectrum.SpectralDistribution | None`) – Illuminant spectral distribution, default to *CIE Standard Illuminant D65*.
- **optimisation_kwargs** (`dict | None`) – Parameters for `scipy.optimize.minimize()` definition.

Returns Recovered spectral distribution.

Return type colour.SpectralDistribution

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

- The definition used to convert spectrum to *CIE XYZ* tristimulus values is `colour.colorimetry.spectral_to_XYZ_integration()` definition because it processes any measurement interval opposed to `colour.colorimetry.sd_to_XYZ_ASTME308()` definition that handles only measurement interval of 1, 5, 10 or 20nm.

References

[MSHD15]

Examples

```
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd_Meng2015(XYZ, cmfs, illuminant)
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 360.        ,    0.0762005...],
                      [ 370.        ,    0.0761792...],
                      [ 380.        ,    0.0761363...],
                      [ 390.        ,    0.0761194...],
                      [ 400.        ,    0.0762539...],
                      [ 410.        ,    0.0761671...],
                      [ 420.        ,    0.0754649...],
                      [ 430.        ,    0.0731519...],
                      [ 440.        ,    0.0676701...],
                      [ 450.        ,    0.0577800...],
                      [ 460.        ,    0.0441993...],
                      [ 470.        ,    0.0285064...],
                      [ 480.        ,    0.0138728...],
                      [ 490.        ,    0.0033585...],
                      [ 500.        ,    0.       ...],
                      [ 510.        ,    0.       ...],
                      [ 520.        ,    0.       ...],
                      [ 530.        ,    0.       ...],
                      [ 540.        ,    0.0055767...],
                      [ 550.        ,    0.0317581...],
                      [ 560.        ,    0.0754491...],
```

(continues on next page)

(continued from previous page)

```
                                  [ 570.         ,    0.1314115...],
                                  [ 580.         ,    0.1937649...],
                                  [ 590.         ,    0.2559311...],
                                  [ 600.         ,    0.3123173...],
                                  [ 610.         ,    0.3584966...],
                                  [ 620.         ,    0.3927335...],
                                  [ 630.         ,    0.4159458...],
                                  [ 640.         ,    0.4306660...],
                                  [ 650.         ,    0.4391040...],
                                  [ 660.         ,    0.4439497...],
                                  [ 670.         ,    0.4463618...],
                                  [ 680.         ,    0.4474625...],
                                  [ 690.         ,    0.4479868...],
                                  [ 700.         ,    0.4482116...],
                                  [ 710.         ,    0.4482800...],
                                  [ 720.         ,    0.4483472...],
                                  [ 730.         ,    0.4484251...],
                                  [ 740.         ,    0.4484633...],
                                  [ 750.         ,    0.4485071...],
                                  [ 760.         ,    0.4484969...],
                                  [ 770.         ,    0.4484853...],
                                  [ 780.         ,    0.4485134...]],
                                 interpolator=SpragueInterpolator,
                                 interpolator_kwargs={},
                                 extrapolator=Extrapolator,
                                 extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

Otsu, Yamamoto and Hachisuka (2018)

`colour.recovery`

XYZ_to_sd_Otsu2018(XYZ[, cmfs, illuminant, ...])	Recover the spectral distribution of given *CIE XYZ* tristimulus values using *Otsu et al. (2018)* method.

colour.recovery.XYZ_to_sd_Otsu2018

colour.recovery.**XYZ_to_sd_Otsu2018**(*XYZ: ArrayLike*, *cmfs:* colour.colorimetry.spectrum.MultiSpectralDistributions | *None* = *None*, *illuminant:* colour.colorimetry.spectrum.SpectralDistribution | *None* = *None*, *dataset:* colour.recovery.otsu2018.Dataset_Otsu2018 = *DATASET_REFERENCE_OTSU2018*, *clip:* *bool* = *True*) → *colour.colorimetry.spectrum.SpectralDistribution*

Recover the spectral distribution of given *CIE XYZ* tristimulus values using *Otsu et al. (2018)* method.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values to recover the spectral distribution from.

- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Standard Illuminant D65*.
- **dataset** (colour.recovery.otsu2018.Dataset_Otsu2018) – Dataset to use for reconstruction. The default is to use the published data.
- **clip** (bool) – If *True*, the default, values below zero and above unity in the recovered spectral distributions will be clipped. This ensures that the returned reflectance is physical and conserves energy, but will cause noticeable colour differences in case of very saturated colours.

Returns Recovered spectral distribution. Its shape is always that of the colour.recovery.SPECTRAL_SHAPE_OTSU2018 class instance.

Return type colour.SpectralDistribution

Raises ValueError – If the dataset shape is undefined.

References

[OYH18]

Examples

```
>>> from colour import (
...     CCS_ILLUMINANTS,
...     SDS_ILLUMINANTS,
...     MSDS_CMFS,
...     XYZ_to_sRGB,
... )
>>> from colour.colorimetry import sd_to_XYZ_integration
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SPECTRAL_SHAPE_OTSU2018)
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd_Otsu2018(XYZ, cmfs, illuminant)
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 380.        ,    0.0601939...],
                      [ 390.        ,    0.0568063...],
                      [ 400.        ,    0.0517429...],
                      [ 410.        ,    0.0495841...],
                      [ 420.        ,    0.0502007...],
                      [ 430.        ,    0.0506489...],
                      [ 440.        ,    0.0510020...],
                      [ 450.        ,    0.0493782...],
                      [ 460.        ,    0.0468046...],
                      [ 470.        ,    0.0437132...],
                      [ 480.        ,    0.0416957...],
                      [ 490.        ,    0.0403783...],
```

(continues on next page)

(continued from previous page)

```
                [ 500.        ,    0.0405197...],
                [ 510.        ,    0.0406031...],
                [ 520.        ,    0.0416912...],
                [ 530.        ,    0.0430956...],
                [ 540.        ,    0.0444474...],
                [ 550.        ,    0.0459336...],
                [ 560.        ,    0.0507631...],
                [ 570.        ,    0.0628967...],
                [ 580.        ,    0.0844661...],
                [ 590.        ,    0.1334277...],
                [ 600.        ,    0.2262428...],
                [ 610.        ,    0.3599330...],
                [ 620.        ,    0.4885571...],
                [ 630.        ,    0.5752546...],
                [ 640.        ,    0.6193023...],
                [ 650.        ,    0.6450744...],
                [ 660.        ,    0.6610548...],
                [ 670.        ,    0.6688673...],
                [ 680.        ,    0.6795426...],
                [ 690.        ,    0.6887933...],
                [ 700.        ,    0.7003469...],
                [ 710.        ,    0.7084128...],
                [ 720.        ,    0.7154674...],
                [ 730.        ,    0.7234334...]],
               SpragueInterpolator,
               {},
               Extrapolator,
               {'method': 'Constant', 'left': None, 'right': None})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065494...,  0.1219712...,  0.0514002...])
```

Ancillary Objects

`colour.recovery`

Dataset_Otsu2018([shape, basis_functions, ...])	Store all the information needed for the *Otsu et al. (2018)* spectral upsampling method.
Tree_Otsu2018(*args, **kwargs)	A sub-class of `colour.recovery.otsu2018.Node` class representing the root node of a tree containing information shared with all the nodes, such as the standard observer colour matching functions and the illuminant, if any is used.

colour.recovery.Dataset_Otsu2018

class `colour.recovery.`**`Dataset_Otsu2018`**(*shape: SpectralShape | None = None, basis_functions: NDArrayFloat | None = None, means: NDArrayFloat | None = None, selector_array: NDArrayFloat | None = None*)

Store all the information needed for the *Otsu et al. (2018)* spectral upsampling method.

Datasets can be either generated and converted as a `colour.recovery.Dataset_Otsu2018` class instance using the `colour.recovery.Tree_Otsu2018.to_dataset()` method or alternatively, loaded from disk with the `colour.recovery.Dataset_Otsu2018.read()` method.

Parameters

- **shape** (`SpectralShape | None`) – Shape of the spectral data.

- **basis_functions** (NDArrayFloat | None) – Three basis functions for every cluster.
- **means** (NDArrayFloat | None) – Mean for every cluster.
- **selector_array** (NDArrayFloat | None) – Array describing how to select the appropriate cluster. See colour.recovery.Dataset_Otsu2018.select() method for details.

Return type None

Attributes

- shape
- basis_functions
- means
- selector_array

Methods

- __init__()
- select()
- cluster()
- read()
- write()

References

[OYH18]

Examples

```
>>> import os
>>> import colour
>>> from colour.characterisation import SDS_COLOURCHECKERS
>>> from colour.colorimetry import sds_and_msds_to_msds
>>> reflectances = sds_and_msds_to_msds(
...     SDS_COLOURCHECKERS["ColorChecker N Ohta"].values()
... )
>>> node_tree = Tree_Otsu2018(reflectances)
>>> node_tree.optimise(iterations=2, print_callable=lambda x: x)
>>> dataset = node_tree.to_dataset()
>>> path = os.path.join(
...     colour.__path__[0],
...     "recovery",
...     "tests",
...     "resources",
...     "ColorChecker_Otsu2018.npz",
... )
>>> dataset.write(path)
>>> dataset = Dataset_Otsu2018()
>>> dataset.read(path)
```

__init__(*shape: SpectralShape | None = None, basis_functions: NDArrayFloat | None = None, means: NDArrayFloat | None = None, selector_array: NDArrayFloat | None = None*) → None

Parameters

- **shape** (SpectralShape | None) –
- **basis_functions** (NDArrayFloat | None) –
- **means** (NDArrayFloat | None) –
- **selector_array** (NDArrayFloat | None) –

Return type None

Methods

`__init__`([shape, basis_functions, means, ...])	
`cluster`(xy)	Return the basis functions and dataset mean for the given *CIE xy* coordinates.
`read`(path)	Read and loads a dataset from an *.npz* file.
`select`(xy)	Return the cluster index appropriate for the given *CIE xy* coordinates.
`write`(path)	Write the dataset to an *.npz* file at given path.

Attributes

`basis_functions`	Getter property for the basis functions of the *Otsu et al. (2018)* dataset.
`means`	Getter property for means of the *Otsu et al. (2018)* dataset.
`selector_array`	Getter property for the selector array of the *Otsu et al. (2018)* dataset.
`shape`	Getter property for the shape used by the *Otsu et al. (2018)* dataset.

colour.recovery.Tree_Otsu2018

class `colour.recovery.`**Tree_Otsu2018**(**args: Any, **kwargs: Any*)

A sub-class of `colour.recovery.otsu2018.Node` class representing the root node of a tree containing information shared with all the nodes, such as the standard observer colour matching functions and the illuminant, if any is used.

Global operations involving the entire tree, such as optimisation and conversion to dataset, are implemented in this sub-class.

Parameters

- **reflectances** (MultiSpectralDistributions) – Reference reflectances of the *n* reference colours to use for optimisation.
- **cmfs** (MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Standard Illuminant D65*.

Return type None

Attributes

- reflectances
- cmfs
- illuminant

Methods

- __init__()
- __str__()
- optimise()
- to_dataset()

References

[OYH18]

Examples

```
>>> import os
>>> import colour
>>> from colour import MSDS_CMFS, SDS_COLOURCHECKERS, SDS_ILLUMINANTS
>>> from colour.colorimetry import sds_and_msds_to_msds
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS["D65"].copy().align(cmfs.shape)
>>> reflectances = sds_and_msds_to_msds(
...     SDS_COLOURCHECKERS["ColorChecker N Ohta"].values()
... )
>>> node_tree = Tree_Otsu2018(reflectances, cmfs, illuminant)
>>> node_tree.optimise(iterations=2, print_callable=lambda x: x)
>>> dataset = node_tree.to_dataset()
>>> path = os.path.join(
...     colour.__path__[0],
...     "recovery",
...     "tests",
...     "resources",
...     "ColorChecker_Otsu2018.npz",
... )
>>> dataset.write(path)
>>> dataset = Dataset_Otsu2018()
>>> dataset.read(path)
>>> sd = XYZ_to_sd_Otsu2018(XYZ, cmfs, illuminant, dataset)
>>> with numpy_print_options(suppress=True):
...     sd
```

(continues on next page)

(continued from previous page)

```
...
SpectralDistribution([[ 360.        ,    0.0651341...],
                      [ 370.        ,    0.0651341...],
                      [ 380.        ,    0.0651341...],
                      [ 390.        ,    0.0749684...],
                      [ 400.        ,    0.0815578...],
                      [ 410.        ,    0.0776439...],
                      [ 420.        ,    0.0721897...],
                      [ 430.        ,    0.0649064...],
                      [ 440.        ,    0.0567185...],
                      [ 450.        ,    0.0484685...],
                      [ 460.        ,    0.0409768...],
                      [ 470.        ,    0.0358964...],
                      [ 480.        ,    0.0307857...],
                      [ 490.        ,    0.0270148...],
                      [ 500.        ,    0.0273773...],
                      [ 510.        ,    0.0303157...],
                      [ 520.        ,    0.0331285...],
                      [ 530.        ,    0.0363027...],
                      [ 540.        ,    0.0425987...],
                      [ 550.        ,    0.0513442...],
                      [ 560.        ,    0.0579256...],
                      [ 570.        ,    0.0653850...],
                      [ 580.        ,    0.0929522...],
                      [ 590.        ,    0.1600326...],
                      [ 600.        ,    0.2586159...],
                      [ 610.        ,    0.3701242...],
                      [ 620.        ,    0.4702243...],
                      [ 630.        ,    0.5396261...],
                      [ 640.        ,    0.5737561...],
                      [ 650.        ,    0.590848 ...],
                      [ 660.        ,    0.5935371...],
                      [ 670.        ,    0.5923295...],
                      [ 680.        ,    0.5956326...],
                      [ 690.        ,    0.5982513...],
                      [ 700.        ,    0.6017904...],
                      [ 710.        ,    0.6016419...],
                      [ 720.        ,    0.5996892...],
                      [ 730.        ,    0.6000018...],
                      [ 740.        ,    0.5964443...],
                      [ 750.        ,    0.5868181...],
                      [ 760.        ,    0.5860973...],
                      [ 770.        ,    0.5614878...],
                      [ 780.        ,    0.5289331...]],
                     SpragueInterpolator,
                     {},
                     Extrapolator,
                     {'method': 'Constant', 'left': None, 'right': None})
```

Return a new instance of the `colour.utilities.Node` class.

Parameters

- **args** – Arguments.
- **kwargs** – Keywords arguments.
- **reflectances** (`MultiSpectralDistributions`) –

- **cmfs** (MultiSpectralDistributions | None) –
- **illuminant** (SpectralDistribution | None) –

Return type None

__init__(*reflectances: colour.colorimetry.spectrum.MultiSpectralDistributions*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*, *illuminant: colour.colorimetry.spectrum.SpectralDistribution | None = None*) → None

Parameters

- **reflectances** (colour.colorimetry.spectrum.MultiSpectralDistributions) –
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) –
- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution | None) –

Return type None

Methods

__init__(reflectances[, cmfs, illuminant])	
branch_reconstruction_error()	Compute the reconstruction error for all the leaves data connected to the node or its children, i.e. the reconstruction errors summation for all the leaves in the branch.
is_inner()	Return whether the node is an inner node.
is_leaf()	Return whether the node is a leaf node.
is_root()	Return whether the node is a root node.
leaf_reconstruction_error()	Return the reconstruction error of the node data.
minimise(minimum_cluster_size)	Find the best partition for the node that minimises the leaf reconstruction error.
optimise([iterations, minimum_cluster_size, ...])	Optimise the tree by repeatedly performing optimal partitioning of the nodes, creating a tree that minimises the total reconstruction error.
render([tab_level])	Render the current node and its children as a string.
split(children, axis)	Convert the leaf node into an inner node using given children and partition axis.
to_dataset()	Create a colour.recovery.Dataset_Otsu2018 class instance based on data stored in the tree.
walk([ascendants])	Return a generator used to walk into colour.utilities.Node trees.

Attributes

children	Getter and setter property for the node children.
cmfs	Getter property for the standard observer colour matching functions.
data	Getter property for the node data.
id	Getter property for the node id.
illuminant	Getter property for the illuminant.
leaves	Getter property for the node leaves.
name	Getter and setter property for the name.
parent	Getter and setter property for the node parent.
partition_axis	Getter property for the node partition axis.
reflectances	Getter property for the reference reflectances.
root	Getter property for the node tree.
row	Getter property for the node row for the selector array.
siblings	Getter property for the node siblings.

Smits (1999)

`colour.recovery`

RGB_to_sd_Smits1999(RGB)	Recover the spectral distribution of given *RGB* colourspace array using *Smits (1999)* method.
SDS_SMITS1999	*Smits (1999)* spectral distributions.

colour.recovery.RGB_to_sd_Smits1999

colour.recovery.**RGB_to_sd_Smits1999**(*RGB: ArrayLike*) → *colour.colorimetry.spectrum.SpectralDistribution*

Recover the spectral distribution of given *RGB* colourspace array using *Smits (1999)* method.

Parameters **RGB** (ArrayLike) – *RGB* colourspace array to recover the spectral distribution from.

Returns Recovered spectral distribution.

Return type colour.SpectralDistribution

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Smi99]

Examples

```
>>> import numpy as np
>>> from colour import MSDS_CMFS, SDS_ILLUMINANTS, SpectralShape
>>> from colour.colorimetry import sd_to_XYZ_integration
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> RGB = XYZ_to_RGB_Smits1999(XYZ)
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS["E"].copy().align(cmfs.shape)
>>> sd = RGB_to_sd_Smits1999(RGB)
>>> with numpy_print_options(suppress=True):
...     sd
...
SpectralDistribution([[ 380.        ,    0.0787830...],
                      [ 417.7778    ,    0.0622018...],
                      [ 455.5556    ,    0.0446206...],
                      [ 493.3333    ,    0.0352220...],
                      [ 531.1111    ,    0.0324149...],
                      [ 568.8889    ,    0.0330105...],
                      [ 606.6667    ,    0.3207115...],
                      [ 644.4444    ,    0.3836164...],
                      [ 682.2222    ,    0.3836164...],
                      [ 720.        ,    0.3835649...]],
                     LinearInterpolator,
                     {},
                     Extrapolator,
                     {'method': 'Constant', 'left': None, 'right': None})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.1894770...,  0.1126470...,  0.0474420...])
```

colour.recovery.SDS_SMITS1999

colour.recovery.**SDS_SMITS1999 = CanonicalMapping({'white': ..., 'cyan': ..., 'magenta': ..., 'yellow': ..., 'red': ..., 'green': ..., 'blue': ...})**

Smits (1999) spectral distributions.

References

[Smi99]

Camera RGB Sensitivities Recovery

Jiang, Liu, Gu and Süsstrunk (2013)

`colour.recovery`

`RGB_to_sd_camera_sensitivity_Jiang2013`(RGB, ...)	Recover a single camera *RGB* sensitivity for given camera *RGB* values using *Jiang et al. (2013)* method.
`RGB_to_msds_camera_sensitivities_Jiang2013`(...)	Recover the camera *RGB* sensitivities for given camera *RGB* values using *Jiang et al. (2013)* method.

colour.recovery.RGB_to_sd_camera_sensitivity_Jiang2013

colour.recovery.**RGB_to_sd_camera_sensitivity_Jiang2013**(*RGB: ArrayLike*, *illuminant:* colour.colorimetry.spectrum.SpectralDistribution, *reflectances:* colour.colorimetry.spectrum.MultiSpectralDistributions, *eigen_w: ArrayLike*, *shape:* colour.colorimetry.spectrum.SpectralShape | *None* = *None*) → *colour.colorimetry.spectrum.SpectralDistribution*

Recover a single camera *RGB* sensitivity for given camera *RGB* values using *Jiang et al. (2013)* method.

Parameters

- **RGB** (`ArrayLike`) – Camera *RGB* values corresponding with `reflectances`.
- **illuminant** (`colour.colorimetry.spectrum.SpectralDistribution`) – Illuminant spectral distribution used to produce the camera *RGB* values.
- **reflectances** (`colour.colorimetry.spectrum.MultiSpectralDistributions`) – Reflectance spectral distributions used to produce the camera *RGB* values.
- **eigen_w** (`ArrayLike`) – Eigen-vectors v for the particular camera *RGB* sensitivity being recovered.
- **shape** (`colour.colorimetry.spectrum.SpectralShape | None`) – Spectral shape of the recovered camera *RGB* sensitivity, `illuminant` and `reflectances` will be aligned to it if passed, otherwise, `illuminant` shape is used.

Returns Recovered camera *RGB* sensitivities.

Return type `colour.RGB_CameraSensitivities`

Examples

```
>>> from colour.colorimetry import (
...     SDS_ILLUMINANTS,
...     msds_to_XYZ,
...     sds_and_msds_to_msds,
... )
>>> from colour.characterisation import (
...     MSDS_CAMERA_SENSITIVITIES,
...     SDS_COLOURCHECKERS,
... )
>>> from colour.recovery import SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017
>>> illuminant = SDS_ILLUMINANTS["D65"]
>>> sensitivities = MSDS_CAMERA_SENSITIVITIES["Nikon 5100 (NPL)"]
>>> reflectances = [
...     sd.copy().align(SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017)
...     for sd in SDS_COLOURCHECKERS["BabelColor Average"].values()
... ]
>>> reflectances = sds_and_msds_to_msds(reflectances)
>>> R, G, B = tsplit(
...     msds_to_XYZ(
...         reflectances,
...         method="Integration",
...         cmfs=sensitivities,
...         illuminant=illuminant,
...         k=1,
...         shape=SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017,
...     )
... )
>>> R_w, G_w, B_w = tsplit(np.moveaxis(BASIS_FUNCTIONS_DYER2017, 0, 1))
>>> RGB_to_sd_camera_sensitivity_Jiang2013(
...     R,
...     illuminant,
...     reflectances,
...     R_w,
...     SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017,
... )
SpectralDistribution([[  4.00000000e+02,   7.3976716...e-04],
                      [  4.10000000e+02,  -8.7040243...e-04],
                      [  4.20000000e+02,   4.6893657...e-03],
                      [  4.30000000e+02,   7.7522012...e-03],
                      [  4.40000000e+02,   6.9238417...e-03],
                      [  4.50000000e+02,   5.3089422...e-03],
                      [  4.60000000e+02,   4.4780109...e-03],
                      [  4.70000000e+02,   4.6386816...e-03],
                      [  4.80000000e+02,   5.1897663...e-03],
                      [  4.90000000e+02,   4.3906620...e-03],
                      [  5.00000000e+02,   4.2189259...e-03],
                      [  5.10000000e+02,   5.4270976...e-03],
                      [  5.20000000e+02,   9.6722601...e-03],
                      [  5.30000000e+02,   1.4272520...e-02],
                      [  5.40000000e+02,   7.9609053...e-03],
                      [  5.50000000e+02,   4.5917460...e-03],
                      [  5.60000000e+02,   5.2723695...e-03],
                      [  5.70000000e+02,   1.0479224...e-02],
                      [  5.80000000e+02,   5.3101298...e-02],
                      [  5.90000000e+02,   9.8185490...e-02],
```

(continues on next page)

(continued from previous page)

```
                   [  6.00000000e+02,   9.9775094...e-02],
                   [  6.10000000e+02,   8.3935824...e-02],
                   [  6.20000000e+02,   6.9216733...e-02],
                   [  6.30000000e+02,   5.6902763...e-02],
                   [  6.40000000e+02,   4.2810635...e-02],
                   [  6.50000000e+02,   3.0064003...e-02],
                   [  6.60000000e+02,   2.3093789...e-02],
                   [  6.70000000e+02,   1.3756855...e-02],
                   [  6.80000000e+02,   4.1785101...e-03],
                   [  6.90000000e+02,  -3.8014848...e-04],
                   [  7.00000000e+02,  -5.7544253...e-04]],
                  SpragueInterpolator,
                  {},
                  Extrapolator,
                  {'method': 'Constant', 'left': None, 'right': None})
```

colour.recovery.RGB_to_msds_camera_sensitivities_Jiang2013

`colour.recovery.`**`RGB_to_msds_camera_sensitivities_Jiang2013`**(*RGB: ArrayLike*, *illuminant: colour.colorimetry.spectrum.SpectralDistribution*, *reflectances: colour.colorimetry.spectrum.MultiSpectralDistributions*, *basis_functions=BASIS_FUNCTIONS_DYER2017*, *shape: colour.colorimetry.spectrum.SpectralShape | None = None*) → *colour.colorimetry.spectrum.MultiSpectralDistributions*

Recover the camera *RGB* sensitivities for given camera *RGB* values using *Jiang et al. (2013)* method.

Parameters

- **RGB** (`ArrayLike`) – Camera *RGB* values corresponding with `reflectances`.
- **illuminant** (`colour.colorimetry.spectrum.SpectralDistribution`) – Illuminant spectral distribution used to produce the camera *RGB* values.
- **reflectances** (`colour.colorimetry.spectrum.MultiSpectralDistributions`) – Reflectance spectral distributions used to produce the camera *RGB* values.
- **basis_functions** – Basis functions for the method. The default is to use the built-in *sRGB* basis functions, i.e. `colour.recovery.BASIS_FUNCTIONS_DYER2017`.
- **shape** (`colour.colorimetry.spectrum.SpectralShape | None`) – Spectral shape of the recovered camera *RGB* sensitivities, `illuminant` and `reflectances` will be aligned to it if passed, otherwise, `illuminant` shape is used.

Returns Recovered camera *RGB* sensitivities.

Return type `colour.RGB_CameraSensitivities`

Examples

```
>>> from colour.colorimetry import (
...     SDS_ILLUMINANTS,
...     msds_to_XYZ,
...     sds_and_msds_to_msds,
... )
>>> from colour.characterisation import (
...     MSDS_CAMERA_SENSITIVITIES,
...     SDS_COLOURCHECKERS,
... )
>>> from colour.recovery import SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017
>>> illuminant = SDS_ILLUMINANTS["D65"]
>>> sensitivities = MSDS_CAMERA_SENSITIVITIES["Nikon 5100 (NPL)"]
>>> reflectances = [
...     sd.copy().align(SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017)
...     for sd in SDS_COLOURCHECKERS["BabelColor Average"].values()
... ]
>>> reflectances = sds_and_msds_to_msds(reflectances)
>>> RGB = msds_to_XYZ(
...     reflectances,
...     method="Integration",
...     cmfs=sensitivities,
...     illuminant=illuminant,
...     k=1,
...     shape=SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017,
... )
>>> RGB_to_msds_camera_sensitivities_Jiang2013(
...     RGB,
...     illuminant,
...     reflectances,
...     BASIS_FUNCTIONS_DYER2017,
...     SPECTRAL_SHAPE_BASIS_FUNCTIONS_DYER2017,
... ).values
array([[  7.2281577...e-03,   9.2250648...e-03,  -9.8836897...e-03],
       [ -8.5045760...e-03,   1.1277748...e-02,   3.8624865...e-03],
       [  4.5819113...e-02,   7.1552094...e-02,   4.0406829...e-01],
       [  7.5745635...e-02,   1.1530030...e-01,   7.1177452...e-01],
       [  6.7651854...e-02,   1.5311354...e-01,   8.5161378...e-01],
       [  5.1872905...e-02,   1.8828774...e-01,   9.3658053...e-01],
       [  4.3753995...e-02,   2.6093723...e-01,   9.7049828...e-01],
       [  4.5323885...e-02,   3.7531459...e-01,   9.5883525...e-01],
       [  5.0708454...e-02,   4.4750685...e-01,   8.8451412...e-01],
       [  4.2900523...e-02,   4.5047800...e-01,   7.5069924...e-01],
       [  4.1222513...e-02,   6.1672868...e-01,   5.5327277...e-01],
       [  5.3027385...e-02,   7.8015416...e-01,   3.8368507...e-01],
       [  9.4506252...e-02,   9.1751657...e-01,   2.4143664...e-01],
       [  1.3945472...e-01,   1.0000000...e+00,   1.5616071...e-01],
       [  7.7784852...e-02,   9.2719372...e-01,   1.0462050...e-01],
       [  4.4865285...e-02,   8.5627976...e-01,   6.5035086...e-02],
       [  5.1515558...e-02,   7.5193757...e-01,   3.3979292...e-02],
       [  1.0239098...e-01,   6.2562412...e-01,   2.0583993...e-02],
       [  5.1884509...e-01,   4.9264953...e-01,   1.4571020...e-02],
       [  9.5935619...e-01,   3.4322427...e-01,   1.0656116...e-02],
       [  9.7488799...e-01,   2.0857245...e-01,   6.8892462...e-03],
       [  8.2012477...e-01,   1.1178699...e-01,   4.3808407...e-03],
       [  6.7630666...e-01,   6.5977834...e-02,   4.0420907...e-03],
```

(continues on next page)

(continued from previous page)

```
       [  5.5598866...e-01,   4.4719007...e-02,   4.2502316...e-03],
       [  4.1829651...e-01,   3.3471790...e-02,   4.6139542...e-03],
       [  2.9375101...e-01,   2.4044889...e-02,   4.7376860...e-03],
       [  2.2564606...e-01,   1.8870707...e-02,   4.6336440...e-03],
       [  1.3441624...e-01,   1.0702974...e-02,   3.4919622...e-03],
       [  4.0827617...e-02,   5.5529047...e-03,   1.3990786...e-03],
       [ -3.7143757...e-03,   2.5093564...e-03,   3.9765262...e-04],
       [ -5.6225656...e-03,   1.5643397...e-03,   5.8472693...e-04]])
```

Ancillary Objects

colour.recovery

PCA_Jiang2013(msds_camera_sensitivities[, ...])	Perform the *Principal Component Analysis* (PCA) on given camera *RGB* sensitivities.

colour.recovery.PCA_Jiang2013

colour.recovery.**PCA_Jiang2013**(*msds_camera_sensitivities: Mapping[str, MultiSpectralDistributions], eigen_w_v_count: int | None = None, additional_data: bool = False*) → Tuple[Tuple[NDArrayFloat, NDArrayFloat, NDArrayFloat], Tuple[NDArrayFloat, NDArrayFloat, NDArrayFloat]] | Tuple[NDArrayFloat, NDArrayFloat, NDArrayFloat]

Perform the *Principal Component Analysis* (PCA) on given camera *RGB* sensitivities.

Parameters

- **msds_camera_sensitivities** (Mapping[str, MultiSpectralDistributions]) – Camera *RGB* sensitivities.
- **eigen_w_v_count** (int | None) – Eigen-values w and eigen-vectors v count.
- **additional_data** (bool) – Whether to return both the eigen-values w and eigen-vectors v.

Returns Tuple of camera *RGB* sensitivities eigen-values w and eigen-vectors v or tuple of camera *RGB* sensitivities eigen-vectors v.

Return type tuple

Examples

```
>>> from colour.colorimetry import SpectralShape
>>> from colour.characterisation import MSDS_CAMERA_SENSITIVITIES
>>> shape = SpectralShape(400, 700, 10)
>>> camera_sensitivities = {
...     camera: msds.copy().align(shape)
...     for camera, msds in MSDS_CAMERA_SENSITIVITIES.items()
... }
>>> np.array(PCA_Jiang2013(camera_sensitivities)).shape
(3, 31, 31)
```

Colour Temperature

Correlated Colour Temperature

`colour`

uv_to_CCT(uv[, method])	Return the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using given method.
UV_TO_CCT_METHODS	Supported *CIE UCS* colourspace *uv* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.
CCT_to_uv(CCT_D_uv[, method])	Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.
CCT_TO_UV_METHODS	Supported correlated colour temperature T_{cp} to *CIE UCS* colourspace *uv* chromaticity coordinates computation methods.
xy_to_CCT(xy[, method])	Return the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using given method.
XY_TO_CCT_METHODS	Supported *CIE xy* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.
CCT_to_xy(CCT[, method])	Return the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.
CCT_TO_XY_METHODS	Supported correlated colour temperature T_{cp} to *CIE xy* chromaticity coordinates computation methods.

colour.uv_to_CCT

colour.**uv_to_CCT**(*uv: ArrayLike, method: Union[Literal['Krystek 1985', 'Ohno 2013', 'Planck 1900', 'Robertson 1968'], str] = 'Ohno 2013', **kwargs: Any*) → NDArrayFloat

Return the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using given method.

Parameters

- **uv** (ArrayLike) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **method** (Union[Literal['Krystek 1985', 'Ohno 2013', 'Planck 1900', 'Robertson 1968'], str]) – Computation method.
- **cmfs** – {colour.temperature.uv_to_CCT_Ohno2013(), colour.temperature.uv_to_CCT_Planck1900()}, Standard observer colour matching functions.
- **count** – {colour.temperature.uv_to_CCT_Ohno2013()}, Temperatures count in the planckian tables.
- **end** – {colour.temperature.uv_to_CCT_Ohno2013()}, Temperature range end in kelvins.
- **iterations** – {colour.temperature.uv_to_CCT_Ohno2013()}, Number of planckian tables to generate.
- **optimisation_kwargs** – {colour.temperature.uv_to_CCT_Krystek1985()}, Parameters for scipy.optimize.minimize() definition.

- **start** – {colour.temperature.uv_to_CCT_Ohno2013()}, Temperature range start in kelvins.
- **kwargs** (Any) –

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type numpy.ndarray

References

[AdobeSystems13b], [AdobeSystems13a], [CIET14804e], [Kry85], [Ohn14], [WS00m]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.1978, 0.3122])
>>> uv_to_CCT(uv)
array([  6.5074747...e+03,   3.2233463...e-03])
```

colour.UV_TO_CCT_METHODS

colour.**UV_TO_CCT_METHODS = CanonicalMapping({'Krystek 1985': ..., 'Ohno 2013': ..., 'Planck 1900': ..., 'Robertson 1968': ..., 'ohno2013': ..., 'robertson1968': ...})**

Supported *CIE UCS* colourspace *uv* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

[AdobeSystems13b], [AdobeSystems13a], [CIET14804e], [Kry85], [Ohn14], [WS00m]

Aliases:

- 'ohno2013': 'Ohno 2013'
- 'robertson1968': 'Robertson 1968'

colour.CCT_to_uv

colour.**CCT_to_uv**(*CCT_D_uv: ArrayLike, method: Union[Literal['Krystek 1985', 'Ohno 2013', 'Planck 1900', 'Robertson 1968'], str] = 'Ohno 2013', **kwargs: Any*) → NDArrayFloat

Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.

Parameters

- **CCT_D_uv** (ArrayLike) – Correlated colour temperature T_{cp}, Δ_{uv}.
- **method** (Union[Literal['Krystek 1985', 'Ohno 2013', 'Planck 1900', 'Robertson 1968'], str]) – Computation method.
- **cmfs** – {colour.temperature.CCT_to_uv_Ohno2013(), colour.temperature.CCT_to_uv_Planck1900()}, Standard observer colour matching functions.
- **kwargs** (Any) –

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type numpy.ndarray

References

[AdobeSystems13b], [AdobeSystems13a], [CIET14804e], [Kry85], [Ohn14], [WS00m]

Examples

```
>>> import numpy as np
>>> CCT_D_uv = np.array([6507.47380460, 0.00322335])
>>> CCT_to_uv(CCT_D_uv)
array([ 0.1977999...,  0.3121999...])
```

colour.CCT_TO_UV_METHODS

colour.**CCT_TO_UV_METHODS = CanonicalMapping({'Krystek 1985': ..., 'Ohno 2013': ..., 'Planck 1900': ..., 'Robertson 1968': ..., 'ohno2013': ..., 'robertson1968': ...})**

Supported correlated colour temperature T_{cp} to *CIE UCS* colourspace *uv* chromaticity coordinates computation methods.

References

[AdobeSystems13b], [AdobeSystems13a], [CIET14804e], [Kry85], [Ohn14], [WS00m]

Aliases:

- 'ohno2013': 'Ohno 2013'
- 'robertson1968': 'Robertson 1968'

colour.xy_to_CCT

colour.**xy_to_CCT**(*xy: ArrayLike, method: Union[Literal['CIE Illuminant D Series', 'Kang 2002', 'Hernandez 1999', 'McCamy 1992'], str] = 'CIE Illuminant D Series'*) → NDArrayFloat

Return the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using given method.

Parameters

- **xy** (ArrayLike) – *CIE xy* chromaticity coordinates.
- **method** (Union[Literal['CIE Illuminant D Series', 'Kang 2002', 'Hernandez 1999', 'McCamy 1992'], str]) – Computation method.
- **optimisation_kwargs** – {colour.temperature.xy_to_CCT_CIE_D(), colour.temperature.xy_to_CCT_Kang2002()}, Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}.

Return type numpy.ndarray

References

[HernandezAndresLR99], [KMH+02], [Wikipedia01a], [Wikipedia01b], [WS00k]

Examples

```
>>> import numpy as np
>>> xy_to_CCT(np.array([0.31270, 0.32900]))
6508.1175148...
>>> xy_to_CCT(np.array([0.31270, 0.32900]), "Hernandez 1999")
...
6500.7420431...
```

colour.XY_TO_CCT_METHODS

colour.**XY_TO_CCT_METHODS = CanonicalMapping({'CIE Illuminant D Series': ..., 'Hernandez 1999': ..., 'Kang 2002': ..., 'McCamy 1992': ..., 'daylight': ..., 'kang2002': ..., 'mccamy1992': ..., 'hernandez1999': ...})**

Supported *CIE xy* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

[HernandezAndresLR99], [KMH+02], [Wikipedia01a], [Wikipedia01b], [WS00k]

Aliases:

- 'daylight': 'CIE Illuminant D Series'
- 'kang2002': 'Kang 2002'
- 'mccamy1992': 'McCamy 1992'
- 'hernandez1999': 'Hernandez 1999'

colour.CCT_to_xy

colour.**CCT_to_xy**(*CCT: ArrayLike, method: Union[Literal['CIE Illuminant D Series', 'Kang 2002', 'Hernandez 1999', 'McCamy 1992'], str] = 'CIE Illuminant D Series'*) → NDArrayFloat

Return the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.

Parameters

- **CCT** (ArrayLike) – Correlated colour temperature T_{cp}.
- **method** (Union[Literal['CIE Illuminant D Series', 'Kang 2002', 'Hernandez 1999', 'McCamy 1992'], str]) – Computation method.
- **optimisation_kwargs** – {colour.temperature.CCT_to_xy_Hernandez1999(), colour.temperature.CCT_to_xy_McCamy1992()}, Parameters for scipy.optimize.minimize() definition.

Returns *CIE xy* chromaticity coordinates.

Return type numpy.ndarray

References

[HernandezAndresLR99], [KMH+02], [Wikipedia01a], [Wikipedia01b], [WS00k]

Examples

```
>>> CCT_to_xy(6504.38938305)
array([ 0.3127077...,  0.3291128...])
>>> CCT_to_xy(6504.38938305, "Kang 2002")
...
array([ 0.313426 ...,  0.3235959...])
```

colour.CCT_TO_XY_METHODS

colour.**CCT_TO_XY_METHODS = CanonicalMapping({'CIE Illuminant D Series': ..., 'Hernandez 1999': ..., 'Kang 2002': ..., 'McCamy 1992': ..., 'daylight': ..., 'kang2002': ..., 'mccamy1992': ..., 'hernandez1999': ...})**

Supported correlated colour temperature T_{cp} to *CIE xy* chromaticity coordinates computation methods.

References

[HernandezAndresLR99], [KMH+02], [Wikipedia01a], [Wikipedia01b], [WS00k]

Aliases:

- 'daylight': 'CIE Illuminant D Series'
- 'kang2002': 'Kang 2002'
- 'mccamy1992': 'McCamy 1992'
- 'hernandez1999': 'Hernandez 1999'

Robertson (1968)

colour.temperature

mired_to_CCT(mired)	Convert given micro reciprocal degree to correlated colour temperature T_{cp}.
CCT_to_mired(CCT)	Convert given correlated colour temperature T_{cp} to micro reciprocal degree (mired).
uv_to_CCT_Robertson1968(uv)	Return the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Roberston (1968)* method.
CCT_to_uv_Robertson1968(CCT_D_uv)	Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} and Δ_{uv} using *Roberston (1968)* method.

colour.temperature.mired_to_CCT

colour.temperature.**mired_to_CCT**(*mired: ArrayLike*) → NDArrayFloat

Convert given micro reciprocal degree to correlated colour temperature T_{cp}.

Parameters **mired** (ArrayLike) – Micro reciprocal degree (mired).

Returns Correlated colour temperature T_{cp}.

Return type numpy.ndarray

Examples

```
>>> CCT_to_mired(153.84615384615384)
6500.0
```

colour.temperature.CCT_to_mired

colour.temperature.**CCT_to_mired**(*CCT: ArrayLike*) → NDArrayFloat

Convert given correlated colour temperature T_{cp} to micro reciprocal degree (mired).

Parameters **CCT** (ArrayLike) – Correlated colour temperature T_{cp}.

Returns Micro reciprocal degree (mired).

Return type numpy.ndarray

Examples

```
>>> CCT_to_mired(6500)
153.8461538...
```

colour.temperature.uv_to_CCT_Robertson1968

colour.temperature.**uv_to_CCT_Robertson1968**(*uv: ArrayLike*) → NDArrayFloat

Return the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Roberston (1968)* method.

Parameters **uv** (ArrayLike) – *CIE UCS* colourspace *uv* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type numpy.ndarray

References

[AdobeSystems13b], [WS00m]

Examples

```
>>> uv = np.array([0.193741375998230, 0.315221043940594])
>>> uv_to_CCT_Robertson1968(uv)
array([  6.5000162...e+03,   8.3333289...e-03])
```

colour.temperature.CCT_to_uv_Robertson1968

colour.temperature.**CCT_to_uv_Robertson1968**(*CCT_D_uv: ArrayLike*) → NDArrayFloat

Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} and Δ_{uv} using *Roberston (1968)* method.

Parameters **CCT_D_uv** (ArrayLike) – Correlated colour temperature T_{cp}, Δ_{uv}.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type numpy.ndarray

References

[AdobeSystems13a], [WS00m]

Examples

```
>>> CCT_D_uv = np.array([6500.0081378199056, 0.008333331244225])
>>> CCT_to_uv_Robertson1968(CCT_D_uv)
array([ 0.1937413...,  0.3152210...])
```

Krystek (1985)

colour.temperature

uv_to_CCT_Krystek1985(uv[, optimisation_kwargs])	Return the correlated colour temperature T_{cp} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Krystek (1985)* method.
CCT_to_uv_Krystek1985(CCT)	Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using *Krystek (1985)* method.

colour.temperature.uv_to_CCT_Krystek1985

colour.temperature.**uv_to_CCT_Krystek1985**(*uv: ArrayLike*, *optimisation_kwargs: dict | None = None*) → NDArrayFloat

Return the correlated colour temperature T_{cp} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Krystek (1985)* method.

Parameters

- **uv** (ArrayLike) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **optimisation_kwargs** (dict | None) – Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}.

Return type numpy.ndarray

> **Warning:** *Krystek (1985)* does not give an analytical inverse transformation to compute the correlated colour temperature T_{cp} from given *CIE UCS* colourspace *uv* chromaticity coordinates, the current implementation relies on optimisation using `scipy.optimize.minimize()` definition and thus has reduced precision and poor performance.

Notes

- *Krystek (1985)* method computations are valid for correlated colour temperature T_{cp} normalised to domain [1000, 15000].

References

[Kry85]

Examples

```
>>> uv_to_CCT_Krystek1985(np.array([0.20047203, 0.31029290]))
...
6504.3894290...
```

colour.temperature.CCT_to_uv_Krystek1985

colour.temperature.**CCT_to_uv_Krystek1985**(*CCT: ArrayLike*) → NDArrayFloat

Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using *Krystek (1985)* method.

Parameters **CCT** (ArrayLike) – Correlated colour temperature T_{cp}.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type numpy.ndarray

Notes

- *Krystek (1985)* method computations are valid for correlated colour temperature T_{cp} normalised to domain [1000, 15000].

References

[Kry85]

Examples

```
>>> CCT_to_uv_Krystek1985(6504.38938305)
array([ 0.2004720...,  0.3102929...])
```

Ohno (2013)

`colour.temperature`

`uv_to_CCT_Ohno2013`(uv[, cmfs, start, end, ...])	Return the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates, colour matching functions and temperature range using *Ohno (2013)* method.
`CCT_to_uv_Ohno2013`(CCT_D_uv[, cmfs])	Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp}, Δ_{uv} and colour matching functions using *Ohno (2013)* method.
`XYZ_to_CCT_Ohno2013`(XYZ[, cmfs, start, end, ...])	Return the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE XYZ* tristimulus values, colour matching functions and temperature range using *Ohno (2013)* method.
`CCT_to_XYZ_Ohno2013`(CCT_D_uv[, cmfs])	Return the *CIE XYZ* tristimulus values from given correlated colour temperature T_{cp}, Δ_{uv} and colour matching functions using *Ohno (2013)* method.

colour.temperature.uv_to_CCT_Ohno2013

`colour.temperature.`**`uv_to_CCT_Ohno2013`**(*uv: ArrayLike, cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None, start: float | None = None, end: float | None = None, spacing: float | None = None*) → NDArrayFloat

Return 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.

Parameters

- **uv** (`ArrayLike`) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | None`) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **start** (`float | None`) – Temperature range start in kelvin degrees, default to 1000.
- **end** (`float | None`) – Temperature range end in kelvin degrees, default to 100000.
- **spacing** (`float | None`) – Spacing between values of the underlying planckian table expressed as a multiplier. Default to 1.001. The closer to 1.0, the higher the precision of the returned colour temperature T_{cp} and Δ_{uv}. 1.01 provides a good balance between performance and accuracy. `spacing` value must be greater than 1.

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type numpy.ndarray

References

[Ohn14]

Examples

```
>>> from colour import MSDS_CMFS, SPECTRAL_SHAPE_DEFAULT
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SPECTRAL_SHAPE_DEFAULT)
... )
>>> uv = np.array([0.1978, 0.3122])
>>> uv_to_CCT_Ohno2013(uv, cmfs)
array([  6.50747...e+03,   3.22334...e-03])
```

colour.temperature.CCT_to_uv_Ohno2013

colour.temperature.**CCT_to_uv_Ohno2013**(*CCT_D_uv: ArrayLike, cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*) → NDArrayFloat

Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp}, Δ_{uv} and colour matching functions using *Ohno (2013)* method.

Parameters

- **CCT_D_uv** (ArrayLike) – Correlated colour temperature T_{cp}, Δ_{uv}.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type numpy.ndarray

References

[Ohn14]

Examples

```
>>> from colour import MSDS_CMFS, SPECTRAL_SHAPE_DEFAULT
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SPECTRAL_SHAPE_DEFAULT)
... )
>>> CCT_D_uv = np.array([6507.4342201047066, 0.003223690901513])
>>> CCT_to_uv_Ohno2013(CCT_D_uv, cmfs)
array([ 0.1977999...,  0.3122004...])
```

colour.temperature.XYZ_to_CCT_Ohno2013

colour.temperature.**XYZ_to_CCT_Ohno2013**(*XYZ: ArrayLike*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*, *start: float | None = None*, *end: float | None = None*, *spacing: float | None = None*)

Return the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE XYZ* tristimulus values, colour matching functions and temperature range using *Ohno (2013)* method.

Parameters

- **XYZ** (ArrayLike) – *XYZ* colourspace *uv* chromaticity coordinates.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **start** (float | None) – Temperature range start in kelvin degrees, default to 1000.
- **end** (float | None) – Temperature range end in kelvin degrees, default to 100000.
- **spacing** (float | None) – Spacing between values of the underlying planckian table expressed as a multiplier. Default to 1.001. The closer to 1.0, the higher the precision of the returned colour temperature T_{cp} and Δ_{uv}. 1.01 provides a good balance between performance and accuracy. spacing value must be greater than 1.

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type numpy.ndarray

References

[Ohn14]

Examples

```
>>> from colour import MSDS_CMFS, SPECTRAL_SHAPE_DEFAULT
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SPECTRAL_SHAPE_DEFAULT)
... )
>>> XYZ = np.array([0.95035049, 1.0, 1.08935705])
>>> XYZ_to_CCT_Ohno2013(XYZ, cmfs)
array([  6.5074399...e+03,   3.2236914...e-03])
```

colour.temperature.CCT_to_XYZ_Ohno2013

colour.temperature.**CCT_to_XYZ_Ohno2013**(*CCT_D_uv: ArrayLike, cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*)

Return the *CIE XYZ* tristimulus values from given correlated colour temperature T_{cp}, Δ_{uv} and colour matching functions using *Ohno (2013)* method.

Parameters

- **CCT_D_uv** (ArrayLike) – Correlated colour temperature T_{cp}, Δ_{uv}.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type numpy.ndarray

Examples

```
>>> from colour import MSDS_CMFS, SPECTRAL_SHAPE_DEFAULT
>>> cmfs = (
...     MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
...     .copy()
...     .align(SPECTRAL_SHAPE_DEFAULT)
... )
>>> CCT_D_uv = np.array([6507.4342201047066, 0.003223690901513])
>>> CCT_to_XYZ_Ohno2013(CCT_D_uv, cmfs)
array([ 0.9503504...,  1.        ,  1.0893570...])
```

Planck (1900)

colour.temperature

uv_to_CCT_Planck1900(uv[, cmfs, ...])	Return the correlated colour temperature T_{cp} of a blackbody from given *CIE UCS* colourspace *uv* chromaticity coordinates and colour matching functions.
CCT_to_uv_Planck1900(CCT[, cmfs])	Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} and colour matching functions using the spectral radiance of a blackbody at the given thermodynamic temperature.

colour.temperature.uv_to_CCT_Planck1900

`colour.temperature.uv_to_CCT_Planck1900`(*uv: ArrayLike*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*, *optimisation_kwargs: dict | None = None*) → NDArrayFloat

Return the correlated colour temperature T_{cp} of a blackbody from given *CIE UCS* colourspace *uv* chromaticity coordinates and colour matching functions.

Parameters

- **uv** (ArrayLike) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **optimisation_kwargs** (dict | None) – Parameters for `scipy.optimize.minimize()` definition.

Returns Correlated colour temperature T_{cp}.

Return type `numpy.ndarray`

Warning: The current implementation relies on optimisation using `scipy.optimize.minimize()` definition and thus has reduced precision and poor performance.

References

[CIET14804e]

Examples

```
>>> uv_to_CCT_Planck1900(np.array([0.20042808, 0.31033343]))
...
6504.0000617...
```

colour.temperature.CCT_to_uv_Planck1900

`colour.temperature.CCT_to_uv_Planck1900`(*CCT: ArrayLike*, *cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None*) → NDArrayFloat

Return the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} and colour matching functions using the spectral radiance of a blackbody at the given thermodynamic temperature.

Parameters

- **CCT** (ArrayLike) – Colour temperature T_{cp}.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type `numpy.ndarray`

References

[CIET14804e]

Examples

```
>>> CCT_to_uv_Planck1900(6504)
array([ 0.2004280...,  0.3103334...])
```

McCamy (1992)

`colour.temperature`

`xy_to_CCT_McCamy1992`(xy)	Return the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *McCamy (1992)* method.
`CCT_to_xy_McCamy1992`(CCT[, optimisation_kwargs])	Return the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *McCamy (1992)* method.

colour.temperature.xy_to_CCT_McCamy1992

`colour.temperature.`**`xy_to_CCT_McCamy1992`**(*xy: ArrayLike*) → NDArrayFloat

Return the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *McCamy (1992)* method.

Parameters **xy** (ArrayLike) – *CIE xy* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}.

Return type `numpy.ndarray`

References

[Wikipedia01a]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.31270, 0.32900])
>>> xy_to_CCT_McCamy1992(xy)
6505.0805913...
```

colour.temperature.CCT_to_xy_McCamy1992

colour.temperature.**CCT_to_xy_McCamy1992**(*CCT: ArrayLike*, *optimisation_kwargs: dict | None = None*) → NDArrayFloat

Return the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *McCamy (1992)* method.

Parameters

- **CCT** (ArrayLike) – Correlated colour temperature T_{cp}.
- **optimisation_kwargs** (dict | None) – Parameters for scipy.optimize.minimize() definition.

Returns *CIE xy* chromaticity coordinates.

Return type numpy.ndarray

Warning: *McCamy (1992)* method for computing *CIE xy* chromaticity coordinates from given correlated colour temperature is not a bijective function and might produce unexpected results. It is given for consistency with other correlated colour temperature computation methods but should be avoided for practical applications. The current implementation relies on optimisation using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[Wikipedia01a]

Examples

```
>>> CCT_to_xy_McCamy1992(6505.0805913074782)
array([ 0.3127...,  0.329...])
```

Hernandez-Andres, Lee and Romero (1999)

colour.temperature

xy_to_CCT_Hernandez1999(xy)	Return the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Hernandez-Andres et al. (1999)* method.
CCT_to_xy_Hernandez1999(CCT[, ...])	Return the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Hernandez-Andres et al. (1999)* method.

colour.temperature.xy_to_CCT_Hernandez1999

colour.temperature.**xy_to_CCT_Hernandez1999**(*xy: ArrayLike*) → NDArrayFloat

Return the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Hernandez-Andres et al. (1999)* method.

Parameters **xy** (ArrayLike) – *CIE xy* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}.

Return type numpy.ndarray

References

[HernandezAndresLR99]

Examples

```
>>> xy = np.array([0.31270, 0.32900])
>>> xy_to_CCT_Hernandez1999(xy)
6500.7420431...
```

colour.temperature.CCT_to_xy_Hernandez1999

colour.temperature.**CCT_to_xy_Hernandez1999**(*CCT: ArrayLike*, *optimisation_kwargs: dict | None = None*) → NDArrayFloat

Return the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Hernandez-Andres et al. (1999)* method.

Parameters

- **CCT** (ArrayLike) – Correlated colour temperature T_{cp}.
- **optimisation_kwargs** (dict | None) – Parameters for scipy.optimize.minimize() definition.

Returns *CIE xy* chromaticity coordinates.

Return type numpy.ndarray

> **Warning:** *Hernandez-Andres et al. (1999)* method for computing *CIE xy* chromaticity coordinates from given correlated colour temperature is not a bijective function and might produce unexpected results. It is given for consistency with other correlated colour temperature computation methods but should be avoided for practical applications. The current implementation relies on optimisation using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[HernandezAndresLR99]

Examples

```
>>> CCT_to_xy_Hernandez1999(6500.7420431786531)
array([ 0.3127...,  0.329...])
```

Kang, Moon, Hong, Lee, Cho and Kim (2002)

colour.temperature

xy_to_CCT_Kang2002(xy[, optimisation_kwargs])	Return the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Kang et al. (2002)* method.
CCT_to_xy_Kang2002(CCT)	Return the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Kang et al. (2002)* method.

colour.temperature.xy_to_CCT_Kang2002

colour.temperature.**xy_to_CCT_Kang2002**(*xy: ArrayLike*, *optimisation_kwargs: dict | None = None*) → NDArrayFloat

Return the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Kang et al. (2002)* method.

Parameters

- **xy** (ArrayLike) – *CIE xy* chromaticity coordinates.
- **optimisation_kwargs** (dict | None) – Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}.

Return type numpy.ndarray

Warning: *Kang et al. (2002)* does not give an analytical inverse transformation to compute the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates, the current implementation relies on optimisation using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[KMH+02]

Examples

```
>>> xy_to_CCT_Kang2002(np.array([0.31342600, 0.32359597]))
...
6504.3893128...
```

colour.temperature.CCT_to_xy_Kang2002

colour.temperature.**CCT_to_xy_Kang2002**(*CCT: ArrayLike*) → NDArrayFloat

Return the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Kang et al. (2002)* method.

Parameters **CCT** (ArrayLike) – Correlated colour temperature T_{cp}.

Returns *CIE xy* chromaticity coordinates.

Return type numpy.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

xy_to_CCT_CIE_D(xy[, optimisation_kwargs])	Return the correlated colour temperature T_{cp} of a *CIE Illuminant D Series* from its *CIE xy* chromaticity coordinates.
CCT_to_xy_CIE_D(CCT)	Return the *CIE xy* chromaticity coordinates of a *CIE Illuminant D Series* from its correlated colour temperature T_{cp}.

colour.temperature.xy_to_CCT_CIE_D

colour.temperature.**xy_to_CCT_CIE_D**(*xy: ArrayLike*, *optimisation_kwargs: dict | None = None*) → NDArrayFloat

Return the correlated colour temperature T_{cp} of a *CIE Illuminant D Series* from its *CIE xy* chromaticity coordinates.

Parameters

- **xy** (ArrayLike) – *CIE xy* chromaticity coordinates.
- **optimisation_kwargs** (dict | None) – Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}.

Return type numpy.ndarray

> **Warning:** The *CIE Illuminant D Series* method does not give an analytical inverse transformation to compute the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates, the current implementation relies on optimisation using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[WS00k]

Examples

```
>>> xy_to_CCT_CIE_D(np.array([0.31270775, 0.32911283]))
...
6504.3895840...
```

colour.temperature.CCT_to_xy_CIE_D

colour.temperature.**CCT_to_xy_CIE_D**(*CCT: ArrayLike*) → NDArrayFloat

Return the *CIE xy* chromaticity coordinates of a *CIE Illuminant D Series* from its correlated colour temperature T_{cp}.

Parameters **CCT** (ArrayLike) – Correlated colour temperature T_{cp}.

Returns *CIE xy* chromaticity coordinates.

Return type numpy.ndarray

Raises **ValueError** – If the correlated colour temperature is not in appropriate domain.

References

[WS00k]

Examples

```
>>> CCT_to_xy_CIE_D(6504.38938305)
array([ 0.3127077...,  0.3291128...])
```

Utilities

Callback Management

colour

colour.utilities

Callback(name, function)	Define a callback.
MixinCallback()	A mixin providing support for callbacks.

colour.utilities.Callback

class colour.utilities.**Callback**(*name: str*, *function: Callable*)

Bases: object

Define a callback.

Parameters

- **name** (str) – Callback name.
- **function** (Callable) – Callback callable.

Return type None

__eq__(*other*)

Return self==value.

__hash__ = None

__init__(*name: str*, *function: Callable*) → None

Parameters

- **name** (str) –
- **function** (Callable) –

Return type None

__repr__()

Return repr(self).

__weakref__

list of weak references to the object (if defined)

colour.utilities.MixinCallback

class colour.utilities.**MixinCallback**

Bases: object

A mixin providing support for callbacks.

Attributes

- callbacks
- __setattr__

Methods

- register_callback()
- unregister_callback()

Examples

```
>>> class WithCallback(MixinCallback):
...     def __init__(self):
...         super().__init__()
...         self.attribute_a = "a"
...
>>> with_callback = WithCallback()
>>> def _on_attribute_a_changed(self, name: str, value: str) -> str:
...     return value.upper()
>>> with_callback.register_callback(
...     "attribute_a", "on_attribute_a_changed", _on_attribute_a_changed
... )
>>> with_callback.attribute_a = "a"
>>> with_callback.attribute_a
'A'
```

Return type None

`__init__`() → None

Return type None

property callbacks: collections.defaultdict[str, List[colour.utilities.callback.Callback]]

Getter property for the callbacks.

Returns Callbacks.

Return type `defaultdict`

`__setattr__`(*name: str*, *value: Any*) → None

Set given value to the attribute with given name.

Parameters

- **attribute** – Attribute to set the value of.
- **value** (Any) – Value to set the attribute with.
- **name** (str) –

Return type None

`register_callback`(*attribute: str*, *name: str*, *function: Callable*) → None

Register the callback with given name for given attribute.

Parameters

- **attribute** (str) – Attribute to register the callback for.
- **name** (str) – Callback name.
- **function** (Callable) – Callback callable.

Return type None

Examples

```
>>> class WithCallback(MixinCallback):
...     def __init__(self):
...         super().__init__()
...         self.attribute_a = "a"
...
>>> with_callback = WithCallback()
>>> with_callback.register_callback(
...     "attribute_a", "callback", lambda *args: None
... )
>>> with_callback.callbacks
defaultdict(<class 'list'>, {'attribute_a': [Callback(name='callback', function=
↪<function <lambda> at 0x...>)]})
```

unregister_callback(*attribute: str*, *name: str*) → None

Unregister the callback with given name for given attribute.

Parameters

- **attribute** (str) – Attribute to unregister the callback for.
- **name** (str) – Callback name.

Return type None

Examples

```
>>> class WithCallback(MixinCallback):
...     def __init__(self):
...         super().__init__()
...         self.attribute_a = "a"
...
>>> with_callback = WithCallback()
>>> with_callback.register_callback(
...     "attribute_a", "callback", lambda s, n, v: v
... )
>>> with_callback.callbacks
defaultdict(<class 'list'>, {'attribute_a': [Callback(name='callback', function=
↪<function <lambda> at 0x...>)]})
>>> with_callback.unregister_callback("attribute_a", "callback")
>>> with_callback.callbacks
defaultdict(<class 'list'>, {})
```

__weakref__

list of weak references to the object (if defined)

Common

colour

domain_range_scale(scale)	Define context manager and decorator temporarily setting *Colour* domain-range scale.
get_domain_range_scale()	Return the current *Colour* domain-range scale.
set_domain_range_scale([scale])	Set the current *Colour* domain-range scale.

colour.domain_range_scale

class colour.**domain_range_scale**(*scale: Union[Literal['ignore', 'reference', 'Ignore', 'Reference', '1', '100'], str]*)

Define context manager and decorator temporarily setting *Colour* domain-range scale.

The following scales are available:

- **'Reference'**, the default *Colour* domain-range scale which varies depending on the referenced algorithm, e.g. [0, 1], [0, 10], [0, 100], [0, 255], etc...
- **'1'**, a domain-range scale normalised to [0, 1], it is important to acknowledge that this is a soft normalisation and it is possible to use negative out of gamut values or high dynamic range data exceeding 1.

Parameters **scale** (`Literal['ignore', 'reference', 'Ignore', 'Reference', '1', '100'] | str`) – *Colour* domain-range scale to set.

Return type None

> **Warning:**
>
> - The **'Ignore'** and **'100'** domain-range scales are for internal usage only!

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale("1"):
...     to_domain_1(1)
...
array(1.0)
>>> with domain_range_scale("Reference"):
...     from_range_1(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale("1"):
...     to_domain_1(1)
...
array(1.0)
>>> with domain_range_scale("1"):
...     from_range_1(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale("100"):
...     to_domain_1(1)
...
array(0.01)
>>> with domain_range_scale("100"):
...     from_range_1(1)
...
array(100.0)
```

__init__(*scale: Union[Literal['ignore', 'reference', 'Ignore', 'Reference', '1', '100'], str]*) → None

Parameters **scale** (`Union[Literal['ignore', 'reference', 'Ignore', 'Reference', '1', '100'], str]`) –

Return type None

Methods

`__init__`(scale)

colour.get_domain_range_scale

`colour.`**`get_domain_range_scale`**`()` → Union[Literal['ignore', 'reference', '1', '100'], str]

Return the current *Colour* domain-range scale. The following scales are available:

- **'Reference'**, the default *Colour* domain-range scale which varies depending on the referenced algorithm, e.g. [0, 1], [0, 10], [0, 100], [0, 255], etc...
- **'1'**, a domain-range scale normalised to [0, 1], it is important to acknowledge that this is a soft normalisation and it is possible to use negative out of gamut values or high dynamic range data exceeding 1.

Returns *Colour* domain-range scale.

Return type `str`

> **Warning:**
>
> - The **'Ignore'** and **'100'** domain-range scales are for internal usage only!

colour.set_domain_range_scale

`colour.`**`set_domain_range_scale`**(*scale: Union[Literal['ignore', 'reference', 'Ignore', 'Reference', '1', '100'], str] = 'reference'*)

Set the current *Colour* domain-range scale. The following scales are available:

- **'Reference'**, the default *Colour* domain-range scale which varies depending on the referenced algorithm, e.g. [0, 1], [0, 10], [0, 100], [0, 255], etc...
- **'1'**, a domain-range scale normalised to [0, 1], it is important to acknowledge that this is a soft normalisation and it is possible to use negative out of gamut values or high dynamic range data exceeding 1.

Parameters **scale** (`Union[Literal['ignore', 'reference', 'Ignore', 'Reference', '1', '100'], str]`) – *Colour* domain-range scale to set.

> **Warning:**
>
> - The **'Ignore'** and **'100'** domain-range scales are for internal usage only!

colour.utilities

CacheRegistry()	A registry for mapping-based caches.

colour.utilities.CacheRegistry

class colour.utilities.**CacheRegistry**

Bases: object

A registry for mapping-based caches.

Attributes

- registry

Methods

- `__init__()`
- `__str__()`
- register_cache()
- unregister_cache()
- clear_cache()
- clear_all_caches()

Examples

```
>>> cache_registry = CacheRegistry()
>>> cache_a = cache_registry.register_cache("Cache A")
>>> cache_a["Foo"] = "Bar"
>>> cache_b = cache_registry.register_cache("Cache B")
>>> cache_b["John"] = "Doe"
>>> cache_b["Luke"] = "Skywalker"
>>> print(cache_registry)
{'Cache A': '1 item(s)', 'Cache B': '2 item(s)'}
>>> cache_registry.clear_cache("Cache A")
>>> print(cache_registry)
{'Cache A': '0 item(s)', 'Cache B': '2 item(s)'}
>>> cache_registry.unregister_cache("Cache B")
>>> print(cache_registry)
{'Cache A': '0 item(s)'}
>>> print(cache_b)
{}
```

Return type None

`__init__`() → None

Return type None

property registry: dict

Getter property for the cache registry.

Returns Cache registry.

Return type dict

__str__() → str

Return a formatted string representation of the cache registry.

Returns Formatted string representation.

Return type str

register_cache(*name: str*) → dict

Register a new cache with given name in the registry.

Parameters **name** (str) – Cache name for the registry.

Returns Registered cache.

Return type dict

Examples

```
>>> cache_registry = CacheRegistry()
>>> cache_a = cache_registry.register_cache("Cache A")
>>> cache_a["Foo"] = "Bar"
>>> cache_b = cache_registry.register_cache("Cache B")
>>> cache_b["John"] = "Doe"
>>> cache_b["Luke"] = "Skywalker"
>>> print(cache_registry)
{'Cache A': '1 item(s)', 'Cache B': '2 item(s)'}
```

unregister_cache(*name: str*)

Unregister cache with given name in the registry.

Parameters **name** (str) – Cache name in the registry.

Notes

- The cache is cleared before being unregistered.

Examples

```
>>> cache_registry = CacheRegistry()
>>> cache_a = cache_registry.register_cache("Cache A")
>>> cache_a["Foo"] = "Bar"
>>> cache_b = cache_registry.register_cache("Cache B")
>>> cache_b["John"] = "Doe"
>>> cache_b["Luke"] = "Skywalker"
>>> print(cache_registry)
{'Cache A': '1 item(s)', 'Cache B': '2 item(s)'}
>>> cache_registry.unregister_cache("Cache B")
>>> print(cache_registry)
{'Cache A': '1 item(s)'}
>>> print(cache_b)
{}
```

clear_cache(*name: str*)

Clear the cache with given name.

Parameters **name** (str) – Cache name in the registry.

Examples

```
>>> cache_registry = CacheRegistry()
>>> cache_a = cache_registry.register_cache("Cache A")
>>> cache_a["Foo"] = "Bar"
>>> print(cache_registry)
{'Cache A': '1 item(s)'}
>>> cache_registry.clear_cache("Cache A")
>>> print(cache_registry)
{'Cache A': '0 item(s)'}
```

clear_all_caches()

Clear all the caches in the registry.

Examples

```
>>> cache_registry = CacheRegistry()
>>> cache_a = cache_registry.register_cache("Cache A")
>>> cache_a["Foo"] = "Bar"
>>> cache_b = cache_registry.register_cache("Cache B")
>>> cache_b["John"] = "Doe"
>>> cache_b["Luke"] = "Skywalker"
>>> print(cache_registry)
{'Cache A': '1 item(s)', 'Cache B': '2 item(s)'}
>>> cache_registry.clear_all_caches()
>>> print(cache_registry)
{'Cache A': '0 item(s)', 'Cache B': '0 item(s)'}
```

__weakref__

list of weak references to the object (if defined)

CACHE_REGISTRY	A registry for mapping-based caches.
handle_numpy_errors(**kwargs)	Decorate a function to handle *Numpy* errors.
ignore_numpy_errors(function)	Wrap given function wrapper.
raise_numpy_errors(function)	Wrap given function wrapper.
print_numpy_errors(function)	Wrap given function wrapper.
warn_numpy_errors(function)	Wrap given function wrapper.
ignore_python_warnings(function)	Decorate a function to ignore *Python* warnings.
attest(condition[, message])	Provide the *assert* statement functionality without being disabled by optimised Python execution.
batch(sequence[, k])	Return a batch generator from given sequence.
disable_multiprocessing()	Define a context manager and decorator to temporarily disabling *Colour* multiprocessing state.
multiprocessing_pool(*args, **kwargs)	Define a context manager providing a multiprocessing pool.
is_ctlrender_installed([raise_exception])	Return whether *ctlrender* is installed and available.

continues on next page

Table 4 – continued from previous page

is_graphviz_installed([raise_exception])	Return whether *Graphviz* is installed and available.
is_matplotlib_installed([raise_exception])	Return whether *Matplotlib* is installed and available.
is_networkx_installed([raise_exception])	Return whether *NetworkX* is installed and available.
is_opencolorio_installed([raise_exception])	Return whether *OpenColorIO* is installed and available.
is_openimageio_installed([raise_exception])	Return whether *OpenImageIO* is installed and available.
is_pandas_installed([raise_exception])	Return whether *Pandas* is installed and available.
is_tqdm_installed([raise_exception])	Return whether *tqdm* is installed and available.
is_trimesh_installed([raise_exception])	Return whether *Trimesh* is installed and available.
is_xxhash_installed([raise_exception])	Return whether *xxhash* is installed and available.
required(*requirements)	Decorate a function to check whether various ancillary package requirements are satisfied.
is_iterable(a)	Return whether given variable a is iterable.
is_string(a)	Return whether given variable a is a str-like variable.
is_numeric(a)	Return whether given variable a is a Real-like variable.
is_integer(a)	Return whether given variable a is an numpy.integer-like variable under given threshold.
is_sibling(element, mapping)	Return whether given element type is present in given mapping types.
filter_kwargs(function, **kwargs)	Filter keyword arguments incompatible with the given function signature.
filter_mapping(mapping, names)	Filter given mapping with given names.
first_item(a)	Return the first item of given iterable.
copy_definition(definition[, name])	Copy a definition using the same code, globals, defaults, closure, and name.
validate_method(method, valid_methods[, message])	Validate whether given method exists in the given valid methods and returns the method lower cased.
optional(value, default)	Handle optional argument value by providing a default value.
slugify(object_[, allow_unicode])	Generate a *SEO* friendly and human-readable slug from given object.
int_digest(args[, seed])	Generate an integer digest for given argument using *xxhash* if available or falling back to hash() if not.

colour.utilities.CACHE_REGISTRY

colour.utilities.**CACHE_REGISTRY = <colour.utilities.common.CacheRegistry object>**

A registry for mapping-based caches.

Attributes

- registry

Methods

- `__init__()`
- `__str__()`
- `register_cache()`
- `unregister_cache()`
- `clear_cache()`
- `clear_all_caches()`

Examples

```
>>> cache_registry = CacheRegistry()
>>> cache_a = cache_registry.register_cache("Cache A")
>>> cache_a["Foo"] = "Bar"
>>> cache_b = cache_registry.register_cache("Cache B")
>>> cache_b["John"] = "Doe"
>>> cache_b["Luke"] = "Skywalker"
>>> print(cache_registry)
{'Cache A': '1 item(s)', 'Cache B': '2 item(s)'}
>>> cache_registry.clear_cache("Cache A")
>>> print(cache_registry)
{'Cache A': '0 item(s)', 'Cache B': '2 item(s)'}
>>> cache_registry.unregister_cache("Cache B")
>>> print(cache_registry)
{'Cache A': '0 item(s)'}
>>> print(cache_b)
{}
```

colour.utilities.handle_numpy_errors

`colour.utilities.`**`handle_numpy_errors`**`(**kwargs: Any)` → Callable

Decorate a function to handle *Numpy* errors.

Parameters **kwargs** (Any) – Keywords arguments.

Return type Callable

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: Callable*) → Callable

Wrap given function wrapper.

Parameters **function** (Callable) –

Return type *Callable*

colour.utilities.raise_numpy_errors

colour.utilities.**raise_numpy_errors**(*function: Callable*) → Callable

Wrap given function wrapper.

Parameters **function** (Callable) –

Return type *Callable*

colour.utilities.print_numpy_errors

colour.utilities.**print_numpy_errors**(*function: Callable*) → Callable

Wrap given function wrapper.

Parameters **function** (Callable) –

Return type *Callable*

colour.utilities.warn_numpy_errors

colour.utilities.**warn_numpy_errors**(*function: Callable*) → Callable

Wrap given function wrapper.

Parameters **function** (Callable) –

Return type *Callable*

colour.utilities.ignore_python_warnings

colour.utilities.**ignore_python_warnings**(*function: Callable*) → Callable

Decorate a function to ignore *Python* warnings.

Parameters **function** (Callable) – Function to decorate.

Return type Callable

Examples

```
>>> @ignore_python_warnings
... def f():
...     warnings.warn("This is an ignored warning!")
...
>>> f()
```

colour.utilities.attest

`colour.utilities.`**`attest`**(*condition: bool | DTypeBoolean, message: str = ''*)

Provide the *assert* statement functionality without being disabled by optimised Python execution.

Parameters

- **condition** (`bool | DTypeBoolean`) – Condition to attest/assert.
- **message** (`str`) – Message to display when the assertion fails.

colour.utilities.batch

`colour.utilities.`**`batch`**(*sequence:* collections.abc.Sequence, *k: Union[int, Literal[3]] = 3*) → *collections.abc.Generator*

Return a batch generator from given sequence.

Parameters

- **sequence** (`collections.abc.Sequence`) – Sequence to create batches from.
- **k** (`Union[int, Literal[3]]`) – Batch size.

Yields *Generator* – Batch generator.

Return type *collections.abc.Generator*

Examples

```
>>> batch(tuple(range(10)), 3)
<generator object batch at 0x...>
```

colour.utilities.disable_multiprocessing

class `colour.utilities.`**`disable_multiprocessing`**

Define a context manager and decorator to temporarily disabling *Colour* multiprocessing state.

`__init__`()

Methods

`__init__`()	

colour.utilities.multiprocessing_pool

colour.utilities.**multiprocessing_pool**(**args: Any*, ***kwargs: Any*) → *collections.abc.Generator*

Define a context manager providing a multiprocessing pool.

Parameters

- **args** (Any) – Arguments.
- **kwargs** (Any) – Keywords arguments.

Yields *Generator* – Multiprocessing pool.

Return type *collections.abc.Generator*

Examples

```
>>> from functools import partial
>>> def _add(a, b):
...     return a + b
...
>>> with multiprocessing_pool() as pool:
...     pool.map(partial(_add, b=2), range(10))
...
...
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

colour.utilities.is_ctlrender_installed

colour.utilities.**is_ctlrender_installed**(*raise_exception: bool = False*) → bool

Return whether *ctlrender* is installed and available.

Parameters **raise_exception** (bool) – Whether to raise an exception if *ctlrender* is unavailable.

Returns Whether *ctlrender* is installed.

Return type bool

Raises **ImportError** – If *ctlrender* is not installed.

colour.utilities.is_graphviz_installed

colour.utilities.**is_graphviz_installed**(*raise_exception: bool = False*) → bool

Return whether *Graphviz* is installed and available.

Parameters **raise_exception** (bool) – Whether to raise an exception if *Graphviz* is unavailable.

Returns Whether *Graphviz* is installed.

Return type bool

Raises **ImportError** – If *Graphviz* is not installed.

colour.utilities.is_matplotlib_installed

colour.utilities.**is_matplotlib_installed**(*raise_exception: bool = False*) → bool

Return whether *Matplotlib* is installed and available.

Parameters **raise_exception** (bool) – Whether to raise an exception if *Matplotlib* is unavailable.

Returns Whether *Matplotlib* is installed.

Return type bool

Raises **ImportError** – If *Matplotlib* is not installed.

colour.utilities.is_networkx_installed

colour.utilities.**is_networkx_installed**(*raise_exception: bool = False*) → bool

Return whether *NetworkX* is installed and available.

Parameters **raise_exception** (bool) – Whether to raise an exception if *NetworkX* is unavailable.

Returns Whether *NetworkX* is installed.

Return type bool

Raises **ImportError** – If *NetworkX* is not installed.

colour.utilities.is_opencolorio_installed

colour.utilities.**is_opencolorio_installed**(*raise_exception: bool = False*) → bool

Return whether *OpenColorIO* is installed and available.

Parameters **raise_exception** (bool) – Whether to raise an exception if *OpenColorIO* is unavailable.

Returns Whether *OpenColorIO* is installed.

Return type bool

Raises **ImportError** – If *OpenColorIO* is not installed.

colour.utilities.is_openimageio_installed

colour.utilities.**is_openimageio_installed**(*raise_exception: bool = False*) → bool

Return whether *OpenImageIO* is installed and available.

Parameters **raise_exception** (bool) – Whether to raise an exception if *OpenImageIO* is unavailable.

Returns Whether *OpenImageIO* is installed.

Return type bool

Raises **ImportError** – If *OpenImageIO* is not installed.

colour.utilities.is_pandas_installed

`colour.utilities.is_pandas_installed(raise_exception: bool = False) → bool`

Return whether *Pandas* is installed and available.

Parameters `raise_exception` (bool) – Whether to raise an exception if *Pandas* is unavailable.

Returns Whether *Pandas* is installed.

Return type bool

Raises `ImportError` – If *Pandas* is not installed.

colour.utilities.is_tqdm_installed

`colour.utilities.is_tqdm_installed(raise_exception: bool = False) → bool`

Return whether *tqdm* is installed and available.

Parameters `raise_exception` (bool) – Whether to raise an exception if *tqdm* is unavailable.

Returns Whether *tqdm* is installed.

Return type bool

Raises `ImportError` – If *tqdm* is not installed.

colour.utilities.is_trimesh_installed

`colour.utilities.is_trimesh_installed(raise_exception: bool = False) → bool`

Return whether *Trimesh* is installed and available.

Parameters `raise_exception` (bool) – Whether to raise an exception if *Trimesh* is unavailable.

Returns Whether *Trimesh* is installed.

Return type bool

Raises `ImportError` – If *Trimesh* is not installed.

colour.utilities.is_xxhash_installed

`colour.utilities.is_xxhash_installed(raise_exception: bool = False) → bool`

Return whether *xxhash* is installed and available.

Parameters `raise_exception` (bool) – Whether to raise an exception if *xxhash* is unavailable.

Returns Whether *xxhash* is installed.

Return type bool

Raises `ImportError` – If *xxhash* is not installed.

colour.utilities.required

colour.utilities.**required**(**requirements: Literal['ctlrender', 'Graphviz', 'Matplotlib', 'NetworkX', 'OpenColorIO', 'OpenImageIO', 'Pandas', 'tqdm', 'trimesh', 'xxhash']*) → Callable

Decorate a function to check whether various ancillary package requirements are satisfied.

Parameters **requirements** (`Literal['ctlrender', 'Graphviz', 'Matplotlib', 'NetworkX', 'OpenColorIO', 'OpenImageIO', 'Pandas', 'tqdm', 'trimesh', 'xxhash']`) – Requirements to check whether they are satisfied.

Return type Callable

colour.utilities.is_iterable

colour.utilities.**is_iterable**(*a: Any*) → bool

Return whether given variable a is iterable.

Parameters **a** (`Any`) – Variable a to check the iterability.

Returns Whether variable a is iterable.

Return type `bool`

Examples

```
>>> is_iterable([1, 2, 3])
True
>>> is_iterable(1)
False
```

colour.utilities.is_string

colour.utilities.**is_string**(*a: Any*) → bool

Return whether given variable a is a `str`-like variable.

Parameters **a** (`Any`) – Variable a to test.

Returns Whether variable a is a `str`-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: Any*) → bool

Return whether given variable a is a `Real`-like variable.

Parameters **a** (`Any`) – Variable a to test.

Returns Whether variable a is a `Real`-like variable.

Return type `bool`

Examples

```
>>> is_numeric(1)
True
>>> is_numeric((1,))
False
```

colour.utilities.is_integer

colour.utilities.**is_integer**(*a: Any*) → bool

Return whether given variable a is an `numpy.integer`-like variable under given threshold.

Parameters **a** (`Any`) – Variable a to test.

Returns Whether variable a is an `numpy.integer`-like variable.

Return type `bool`

Notes

- The determination threshold is defined by the `colour.algebra.common.INTEGER_THRESHOLD` attribute.

Examples

```
>>> is_integer(1)
True
>>> is_integer(1.01)
False
```

colour.utilities.is_sibling

colour.utilities.**is_sibling**(*element: Any*, *mapping: collections.abc.Mapping*) → bool

Return whether given element type is present in given mapping types.

Parameters

- **element** (`Any`) – Element to check whether its type is present in the mapping types.
- **mapping** (`collections.abc.Mapping`) – Mapping types.

Returns Whether given element type is present in given mapping types.

Return type `bool`

colour.utilities.filter_kwargs

colour.utilities.**filter_kwargs**(*function: Callable*, ***kwargs: Any*) → dict

Filter keyword arguments incompatible with the given function signature.

Parameters

- **function** (Callable) – Callable to filter the incompatible keyword arguments.
- **kwargs** (Any) – Keywords arguments.

Returns Filtered keyword arguments.

Return type dict

Examples

```
>>> def fn_a(a):
...     return a
...
>>> def fn_b(a, b=0):
...     return a, b
...
>>> def fn_c(a, b=0, c=0):
...     return a, b, c
...
>>> fn_a(1, **filter_kwargs(fn_a, b=2, c=3))
1
>>> fn_b(1, **filter_kwargs(fn_b, b=2, c=3))
(1, 2)
>>> fn_c(1, **filter_kwargs(fn_c, b=2, c=3))
(1, 2, 3)
```

colour.utilities.filter_mapping

colour.utilities.**filter_mapping**(*mapping: collections.abc.Mapping*, *names: str | collections.abc.Sequence[str]*) → dict

Filter given mapping with given names.

Parameters

- **mapping** (collections.abc.Mapping) – Mapping to filter.
- **names** (str | collections.abc.Sequence[str]) – Name for given mapping elements or a list of names.

Returns Filtered mapping elements.

Return type dict

Notes

- If the mapping passed is a colour.utilities.CanonicalMapping class instance, then the lower, slugified and canonical keys are also used for matching.
- To honour the filterers ordering, the return value is a dict class instance.

Examples

```
>>> class Element:
...     pass
...
>>> mapping = {
...     "Element A": Element(),
...     "Element B": Element(),
...     "Element C": Element(),
...     "Not Element C": Element(),
... }
>>> filter_mapping(mapping, "Element A")
{'Element A': <colour.utilities.common.Element object at 0x...>}
```

colour.utilities.first_item

colour.utilities.**first_item**(*a:* collections.abc.Iterable) → *Any*

Return the first item of given iterable.

Parameters **a** (collections.abc.Iterable) – Iterable to get the first item from.

Return type object

Raises **StopIteration** – If the iterable is empty.

Examples

```
>>> a = range(10)
>>> first_item(a)
0
```

colour.utilities.copy_definition

colour.utilities.**copy_definition**(*definition:* Callable, *name:* str | None = *None*) → Callable

Copy a definition using the same code, globals, defaults, closure, and name.

Parameters

- **definition** (Callable) – Definition to be copied.
- **name** (str | None) – Optional definition copy name.

Returns Definition copy.

Return type Callable

colour.utilities.validate_method

colour.utilities.**validate_method**(*method: str, valid_methods: tuple, message: str = '"{0}" method is invalid, it must be one of {1}!'*) → str

Validate whether given method exists in the given valid methods and returns the method lower cased.

Parameters

- **method** (str) – Method to validate.
- **valid_methods** (tuple) – Valid methods.
- **message** (str) – Message for the exception.

Returns Method lower cased.

Return type str

Raises ValueError – If the method does not exist.

Examples

```
>>> validate_method("Valid", ("Valid", "Yes", "Ok"))
'valid'
```

colour.utilities.optional

colour.utilities.**optional**(*value: Optional[colour.utilities.common.T], default: colour.utilities.common.T*) → colour.utilities.common.T

Handle optional argument value by providing a default value.

Parameters

- **value** (Optional[colour.utilities.common.T]) – Optional argument value.
- **default** (colour.utilities.common.T) – Default argument value if value is *None*.

Returns Argument value.

Return type T

Examples

```
>>> optional("Foo", "Bar")
'Foo'
>>> optional(None, "Bar")
'Bar'
```

colour.utilities.slugify

colour.utilities.**slugify**(*object_: Any*, *allow_unicode: bool = False*) → str

Generate a *SEO* friendly and human-readable slug from given object.

Convert to ASCII if `allow_unicode` is *False*. Convert spaces or repeated dashes to single dashes. Remove characters that aren't alphanumerics, underscores, or hyphens. Convert to lowercase. Also strip leading and trailing whitespace, dashes, and underscores.

Parameters

- **object** – Object to convert to a slug.
- **allow_unicode** (bool) – Whether to allow unicode characters in the generated slug.
- **object_** (Any) –

Returns Generated slug.

Return type str

References

[DjangoSFoundation22]

Examples

```
>>> slugify(
...     " Jack & Jill like numbers 1,2,3 and 4 and silly characters ?%.$!/"
... )
'jack-jill-like-numbers-123-and-4-and-silly-characters'
```

colour.utilities.int_digest

colour.utilities.**int_digest**(*args: str | bytes | bytearray | memoryview | array.ArrayType[int]*, *seed: int = 0*) → int

Generate an integer digest for given argument using *xxhash* if available or falling back to `hash()` if not.

Parameters

- **args** (str | bytes | bytearray | memoryview | array.ArrayType[int]) – Argument to generate the int digest of.
- **seed** (int) – Seed used to alter result predictably.

Returns Integer digest.

Return type int

Array

colour.utilities

MixinDataclassFields()	A mixin providing fields introspection for the dataclass-like class fields.
MixinDataclassIterable()	A mixin providing iteration capabilities over the dataclass-like class fields.
MixinDataclassArray()	A mixin providing conversion methods for dataclass-like class conversion to numpy.ndarray class.
MixinDataclassArithmetic()	A mixin providing mathematical operations for dataclass-like class.

colour.utilities.MixinDataclassFields

class colour.utilities.**MixinDataclassFields**

Bases: object

A mixin providing fields introspection for the dataclass-like class fields.

Attributes

- fields

property fields: tuple

Getter property for the fields of the dataclass-like class.

Returns Tuple of dataclass-like class fields.

Return type tuple

__weakref__

list of weak references to the object (if defined)

colour.utilities.MixinDataclassIterable

class colour.utilities.**MixinDataclassIterable**

Bases: colour.utilities.array.MixinDataclassFields

A mixin providing iteration capabilities over the dataclass-like class fields.

Attributes

- keys
- values
- items

Methods

- `__iter__()`

Notes

- The `colour.utilities.MixinDataclassIterable` class inherits the methods from the following class:
  - `colour.utilities.MixinDataclassFields`

property keys: tuple

Getter property for the `dataclass`-like class keys, i.e. the field names.

Returns `dataclass`-like class keys.

Return type `tuple`

property values: tuple

Getter property for the `dataclass`-like class values, i.e. the field values.

Returns `dataclass`-like class values.

Return type `tuple`

property items: tuple

Getter property for the `dataclass`-like class items, i.e. the field names and values.

Returns `dataclass`-like class items.

Return type `tuple`

`__iter__()` → *collections.abc.Generator*

Return a generator for the `dataclass`-like class fields.

Yields *Generator* – `dataclass`-like class field generator.

Return type *collections.abc.Generator*

colour.utilities.MixinDataclassArray

class colour.utilities.**MixinDataclassArray**

Bases: `colour.utilities.array.MixinDataclassIterable`

A mixin providing conversion methods for `dataclass`-like class conversion to `numpy.ndarray` class.

Methods

- `__array__()`

Notes

- The `colour.utilities.MixinDataclassArray` class inherits the methods from the following classes:
  - `colour.utilities.MixinDataclassIterable`
  - `colour.utilities.MixinDataclassFields`

__array__(*dtype: Type[DTypeReal] | None = None*) → NDArray

Implement support for `dataclass`-like class conversion to `numpy.ndarray` class.

A field set to *None* will be filled with *np.nan* according to the shape of the first field not set with *None*.

Parameters **dtype** (`Type[DTypeReal] | None`) – `numpy.dtype` to use for conversion to *np.ndarray*, default to the `numpy.dtype` defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.

Returns `dataclass`-like class converted to `numpy.ndarray`.

Return type `numpy.ndarray`

colour.utilities.MixinDataclassArithmetic

class colour.utilities.**MixinDataclassArithmetic**

Bases: `colour.utilities.array.MixinDataclassArray`

A mixin providing mathematical operations for `dataclass`-like class.

Methods

- `__iadd__()`
- `__add__()`
- `__isub__()`
- `__sub__()`
- `__imul__()`
- `__mul__()`
- `__idiv__()`
- `__div__()`
- `__ipow__()`
- `__pow__()`
- `arithmetical_operation()`

Notes

- The `colour.utilities.MixinDataclassArithmetic` class inherits the methods from the following classes:
  - `colour.utilities.MixinDataclassArray`
  - `colour.utilities.MixinDataclassIterable`
  - `colour.utilities.MixinDataclassFields`

__add__(*a: Any*) → Dataclass

Implement support for addition.

Parameters **a** (`Any`) – Variable a to add.

Returns Variable added `dataclass`-like class.

Return type `dataclass`

__iadd__(*a: Any*) → Dataclass

Implement support for in-place addition.

Parameters **a** (Any) – Variable a to add in-place.

Returns In-place variable added `dataclass`-like class.

Return type `dataclass`

__sub__(*a: Any*) → Dataclass

Implement support for subtraction.

Parameters **a** (Any) – Variable a to subtract.

Returns Variable subtracted `dataclass`-like class.

Return type `dataclass`

__isub__(*a: Any*) → Dataclass

Implement support for in-place subtraction.

Parameters **a** (Any) – Variable a to subtract in-place.

Returns In-place variable subtracted `dataclass`-like class.

Return type `dataclass`

__mul__(*a: Any*) → Dataclass

Implement support for multiplication.

Parameters **a** (Any) – Variable a to multiply by.

Returns Variable multiplied `dataclass`-like class.

Return type `dataclass`

__imul__(*a: Any*) → Dataclass

Implement support for in-place multiplication.

Parameters **a** (Any) – Variable a to multiply by in-place.

Returns In-place variable multiplied `dataclass`-like class.

Return type `dataclass`

__div__(*a: Any*) → Dataclass

Implement support for division.

Parameters **a** (Any) – Variable a to divide by.

Returns Variable divided `dataclass`-like class.

Return type `dataclass`

__idiv__(*a: Any*) → Dataclass

Implement support for in-place division.

Parameters **a** (Any) – Variable a to divide by in-place.

Returns In-place variable divided `dataclass`-like class.

Return type `dataclass`

__itruediv__(*a: Any*) → Dataclass

Implement support for in-place division.

Parameters **a** (Any) – Variable a to divide by in-place.

Returns In-place variable divided `dataclass`-like class.

Return type `dataclass`

__truediv__(*a: Any*) → Dataclass

Implement support for division.

Parameters **a** (Any) – Variable a to divide by.

Returns Variable divided `dataclass`-like class.

Return type `dataclass`

__pow__(*a: Any*) → Dataclass

Implement support for exponentiation.

Parameters **a** (Any) – Variable a to exponentiate by.

Returns Variable exponentiated `dataclass`-like class.

Return type `dataclass`

__ipow__(*a: Any*) → Dataclass

Implement support for in-place exponentiation.

Parameters **a** (Any) – Variable a to exponentiate by in-place.

Returns In-place variable exponentiated `dataclass`-like class.

Return type `dataclass`

arithmetical_operation(*a: Any*, *operation: str*, *in_place: bool = False*) → Dataclass

Perform given arithmetical operation with a operand on the `dataclass`-like class.

Parameters

- **a** (Any) – Operand.
- **operation** (str) – Operation to perform.
- **in_place** (bool) – Operation happens in place.

Returns `dataclass`-like class with arithmetical operation performed.

Return type `dataclass`

as_array(a[, dtype])	Convert given variable a to numpy.ndarray using given numpy.dtype.
as_int(a[, dtype])	Attempt to convert given variable a to numpy.integer using given numpy.dtype.
as_float(a[, dtype])	Attempt to convert given variable a to numpy.floating using given numpy.dtype.
as_int_array(a[, dtype])	Convert given variable a to numpy.ndarray using given numpy.dtype.
as_float_array(a[, dtype])	Convert given variable a to numpy.ndarray using given numpy.dtype.
as_int_scalar(a[, dtype])	Convert given a variable to numpy.integer using given numpy.dtype.
as_float_scalar(a[, dtype])	Convert given a variable to numpy.floating using given numpy.dtype.
set_default_int_dtype([dtype])	Set *Colour* default numpy.integer precision by setting colour.constant.DEFAULT_INT_DTYPE attribute with given numpy.dtype wherever the attribute is imported.
set_default_float_dtype([dtype])	Set *Colour* default numpy.floating precision by setting colour.constant.DEFAULT_FLOAT_DTYPE attribute with given numpy.dtype wherever the attribute is imported.
to_domain_1(a[, scale_factor, dtype])	Scale given array a to domain '**1**'.

continues on next page

Table 5 – continued from previous page

to_domain_10(a[, scale_factor, dtype])	Scale given array a to domain '**10**', used by *Munsell Renotation System*.
to_domain_100(a[, scale_factor, dtype])	Scale given array a to domain '**100**'.
to_domain_degrees(a[, scale_factor, dtype])	Scale given array a to degrees domain.
to_domain_int(a[, bit_depth, dtype])	Scale given array a to int domain.
from_range_1(a[, scale_factor, dtype])	Scale given array a from range '**1**'.
from_range_10(a[, scale_factor, dtype])	Scale given array a from range '**10**', used by *Munsell Renotation System*.
from_range_100(a[, scale_factor, dtype])	Scale given array a from range '**100**'.
from_range_degrees(a[, scale_factor, dtype])	Scale given array a from degrees range.
from_range_int(a[, bit_depth, dtype])	Scale given array a from int range.
is_ndarray_copy_enabled()	Return whether *Colour* numpy.ndarray copy is enabled: Various API objects return a copy of their internal numpy.ndarray for safety purposes but this can be a slow operation impacting performance.
set_ndarray_copy_enable(enable)	Set *Colour* numpy.ndarray copy enabled state.
ndarray_copy_enable(enable)	Define a context manager and decorator temporarily setting *Colour* numpy.ndarray copy enabled state.
ndarray_copy(a)	Return a numpy.ndarray copy if the relevant *Colour* state is enabled: Various API objects return a copy of their internal numpy.ndarray for safety purposes but this can be a slow operation impacting performance.
closest_indexes(a, b)	Return the array a closest element indexes to the reference array b elements.
closest(a, b)	Return the closest array a elements to the reference array b elements.
interval(distribution[, unique])	Return the interval size of given distribution.
is_uniform(distribution)	Return whether given distribution is uniform.
has_only_nan(a)	Return whether given array a contains only NaN values.
in_array(a, b[, tolerance])	Return whether each element of the array a is also present in the array b within given tolerance.
tstack(a[, dtype])	Stack given array of arrays a along the last axis (tail) to produce a stacked array.
tsplit(a[, dtype])	Split given stacked array a along the last axis (tail) to produce an array of arrays.
row_as_diagonal(a)	Return the rows of given array a as diagonal matrices.
orient(a[, orientation])	Orient given array a according to given orientation.
centroid(a)	Return the centroid indexes of given array a.
fill_nan(a[, method, default])	Fill given array a NaN values according to given method.
ndarray_write(a)	Define a context manager setting given array a writeable to operate one and then read-only.
zeros(shape[, dtype, order])	Wrap np.zeros() definition to create an array with the active numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.
ones(shape[, dtype, order])	Wrap np.ones() definition to create an array with the active numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.

continues on next page

Table 5 – continued from previous page

full(shape, fill_value[, dtype, order])	Wrap np.full() definition to create an array with the active type defined by the:attr:*colour.constant.DEFAULT_FLOAT_DTYPE* attribute.
index_along_last_axis(a, indexes)	Reduce the dimension of array a by one, by using an array of indexes to pick elements off the last axis.
format_array_as_row(a[, decimals, separator])	Format given array a as a row.

colour.utilities.as_array

colour.utilities.**as_array**(*a: ArrayLike, dtype: Type[DType] | None = None*) → NDArray

Convert given variable a to numpy.ndarray using given numpy.dtype.

Parameters

- **a** (ArrayLike) – Variable to convert.
- **dtype** (Type[DType] | None) – numpy.dtype to use for conversion, default to the numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.

Returns Variable a converted to numpy.ndarray.

Return type numpy.ndarray

Examples

```
>>> as_array([1, 2, 3])
array([1, 2, 3]...)
>>> as_array([1, 2, 3], dtype=DEFAULT_FLOAT_DTYPE)
array([ 1.,  2.,  3.])
```

colour.utilities.as_int

colour.utilities.**as_int**(*a: ArrayLike, dtype: Type[DTypeInt] | None = None*) → NDArrayInt

Attempt to convert given variable a to numpy.integer using given numpy.dtype. If variable a is not a scalar or 0-dimensional, it is converted to numpy.ndarray.

Parameters

- **a** (ArrayLike) – Variable to convert.
- **dtype** (Type[DTypeInt] | None) – numpy.dtype to use for conversion, default to the numpy.dtype defined by the colour.constant.DEFAULT_INT_DTYPE attribute.

Returns Variable a converted to numpy.integer.

Return type numpy.ndarray

Examples

```
>>> as_int(np.array(1))
1
>>> as_int(np.array([1]))
array([1])
>>> as_int(np.arange(10))
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]...)
```

colour.utilities.as_float

colour.utilities.**as_float**(*a: ArrayLike, dtype: Type[DTypeFloat] | None = None*) → NDArrayFloat

Attempt to convert given variable a to numpy.floating using given numpy.dtype. If variable a is not a scalar or 0-dimensional, it is converted to numpy.ndarray.

Parameters

- **a** (ArrayLike) – Variable to convert.
- **dtype** (Type[DTypeFloat] | None) – numpy.dtype to use for conversion, default to the numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.

Returns Variable a converted to numpy.floating.

Return type numpy.ndarray

Examples

```
>>> as_float(np.array(1))
1.0
>>> as_float(np.array([1]))
array([ 1.])
>>> as_float(np.arange(10))
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

colour.utilities.as_int_array

colour.utilities.**as_int_array**(*a: ArrayLike, dtype: Type[DTypeInt] | None = None*) → NDArrayInt

Convert given variable a to numpy.ndarray using given numpy.dtype.

Parameters

- **a** (ArrayLike) – Variable to convert.
- **dtype** (Type[DTypeInt] | None) – numpy.dtype to use for conversion, default to the numpy.dtype defined by the colour.constant.DEFAULT_INT_DTYPE attribute.

Returns Variable a converted to numpy.ndarray.

Return type numpy.ndarray

Examples

```
>>> as_int_array([1.0, 2.0, 3.0])
array([1, 2, 3]...)
```

colour.utilities.as_float_array

colour.utilities.**as_float_array**(*a: ArrayLike*, *dtype: Type[DTypeFloat] | None = None*) → NDArrayFloat

Convert given variable a to `numpy.ndarray` using given `numpy.dtype`.

Parameters

- **a** (`ArrayLike`) – Variable to convert.
- **dtype** (`Type[DTypeFloat] | None`) – `numpy.dtype` to use for conversion, default to the `numpy.dtype` defined by the `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.

Returns Variable a converted to `numpy.ndarray`.

Return type `numpy.ndarray`

Examples

```
>>> as_float_array([1, 2, 3])
array([ 1.,  2.,  3.])
```

colour.utilities.as_int_scalar

colour.utilities.**as_int_scalar**(*a: ArrayLike*, *dtype: Type[DTypeInt] | None = None*) → int

Convert given a variable to `numpy.integer` using given `numpy.dtype`.

Parameters

- **a** (`ArrayLike`) – Variable to convert.
- **dtype** (`Type[DTypeInt] | None`) – `numpy.dtype` to use for conversion, default to the `numpy.dtype` defined by the `colour.constant.DEFAULT_INT_DTYPE` attribute.

Returns a variable converted to `numpy.integer`.

Return type `int`

Warning:

- The return type is effectively annotated as `int` and not `numpy.integer`.

Examples

```
>>> as_int_scalar(np.array(1))
1
```

colour.utilities.as_float_scalar

colour.utilities.**as_float_scalar**(*a: ArrayLike*, *dtype: Type[DTypeFloat] | None = None*) → float

Convert given a variable to numpy.floating using given numpy.dtype.

Parameters

- **a** (ArrayLike) – Variable to convert.
- **dtype** (Type[DTypeFloat] | None) – numpy.dtype to use for conversion, default to the numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.

Returns a variable converted to numpy.floating.

Return type float

Warning:

- The return type is effectively annotated as float and not numpy.floating.

Examples

```
>>> as_float_scalar(np.array(1))
1.0
```

colour.utilities.set_default_int_dtype

colour.utilities.**set_default_int_dtype**(*dtype: typing.Type[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ceefa45d0>] = DEFAULT_INT_DTYPE*) → None

Set *Colour* default numpy.integer precision by setting colour.constant.DEFAULT_INT_DTYPE attribute with given numpy.dtype wherever the attribute is imported.

Parameters dtype (Type[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ced2d5c90>]) – numpy.dtype to set colour.constant.DEFAULT_INT_DTYPE with.

Return type None

Notes

- It is possible to define the int precision at import time by setting the *COLOUR_SCIENCE__DEFAULT_INT_DTYPE* environment variable, for example *set COLOUR_SCIENCE__DEFAULT_INT_DTYPE=int32*.

Warning: This definition is mostly given for consistency purposes with colour.utilities.set_default_float_dtype() definition but contrary to the latter, changing *integer* precision will almost certainly completely break *Colour*. With great power comes great responsibility.

Examples

```
>>> as_int_array(np.ones(3)).dtype
dtype('int64')
>>> set_default_int_dtype(np.int32)
>>> as_int_array(np.ones(3)).dtype
dtype('int32')
>>> set_default_int_dtype(np.int64)
>>> as_int_array(np.ones(3)).dtype
dtype('int64')
```

colour.utilities.set_default_float_dtype

colour.utilities.**set_default_float_dtype**(*dtype: typing.Type[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ceb10ae10>] = DEFAULT_FLOAT_DTYPE*) → None

Set *Colour* default `numpy.floating` precision by setting `colour.constant.DEFAULT_FLOAT_DTYPE` attribute with given `numpy.dtype` wherever the attribute is imported.

Parameters dtype (`Type[<sphinx.util.inspect.TypeAliasForwardRef object at 0x7f6ceb10ae10>]`) – `numpy.dtype` to set `colour.constant.DEFAULT_FLOAT_DTYPE` with.

Return type None

> **Warning:** Changing *float* precision might result in various *Colour* functionality breaking entirely: https://github.com/numpy/numpy/issues/6860. With great power comes great responsibility.

Notes

- It is possible to define the *float* precision at import time by setting the *COLOUR_SCIENCE__DEFAULT_FLOAT_DTYPE* environment variable, for example *set COLOUR_SCIENCE__DEFAULT_FLOAT_DTYPE=float32*.
- Some definition returning a single-scalar ndarray might not honour the given *float* precision: https://github.com/numpy/numpy/issues/16353

Examples

```
>>> as_float_array(np.ones(3)).dtype
dtype('float64')
>>> set_default_float_dtype(np.float16)
>>> as_float_array(np.ones(3)).dtype
dtype('float16')
>>> set_default_float_dtype(np.float64)
>>> as_float_array(np.ones(3)).dtype
dtype('float64')
```

colour.utilities.to_domain_1

colour.utilities.**to_domain_1**(*a: ArrayLike*, *scale_factor: ArrayLike = 100*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a to domain **'1'**. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'** or **'1'**, the definition is almost entirely by-passed and will conveniently convert array a to `np.ndarray`.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is divided by `scale_factor`, typically 100.

Parameters

- **a** (`ArrayLike`) – Array a to scale to domain **'1'**.
- **scale_factor** (`ArrayLike`) – Scale factor, usually *numeric* but can be a `numpy.ndarray` if some axis need different scaling to be brought to domain **'1'**.
- **dtype** (`Type[DTypeFloat] | None`) – Data type used for the conversion to `np.ndarray`.

Returns Array a scaled to domain **'1'**.

Return type `numpy.ndarray`

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale("Reference"):
...     to_domain_1(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale("1"):
...     to_domain_1(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale("100"):
...     to_domain_1(1)
...
array(0.01)
```

colour.utilities.to_domain_10

colour.utilities.**to_domain_10**(*a: ArrayLike*, *scale_factor: ArrayLike = 10*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a to domain **'10'**, used by *Munsell Renotation System*. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is almost entirely by-passed and will conveniently convert array a to `np.ndarray`.
- If *Colour* domain-range scale is **'1'**, array a is multiplied by `scale_factor`, typically 10.

- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is divided by `scale_factor`, typically 10.

  Parameters

  - **a** (`ArrayLike`) – Array a to scale to domain **'10'**.
  - **scale_factor** (`ArrayLike`) – Scale factor, usually *numeric* but can be a `numpy.ndarray` if some axis need different scaling to be brought to domain **'10'**.
  - **dtype** (`Type[DTypeFloat] | None`) – Data type used for the conversion to `np.ndarray`.

  Returns Array a scaled to domain **'10'**.

  Return type `numpy.ndarray`

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale("Reference"):
...     to_domain_10(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale("1"):
...     to_domain_10(1)
...
array(10.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale("100"):
...     to_domain_10(1)
...
array(0.1)
```

colour.utilities.to_domain_100

`colour.utilities.`**`to_domain_100`**(*a: ArrayLike*, *scale_factor: ArrayLike = 100*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a to domain **'100'**. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'** or **'100'** (currently unsupported private value only used for unit tests), the definition is almost entirely by-passed and will conveniently convert array a to `np.ndarray`.
- If *Colour* domain-range scale is **'1'**, array a is multiplied by `scale_factor`, typically 100.

  Parameters

  - **a** (`ArrayLike`) – Array a to scale to domain **'100'**.
  - **scale_factor** (`ArrayLike`) – Scale factor, usually *numeric* but can be a `numpy.ndarray` if some axis need different scaling to be brought to domain **'100'**.
  - **dtype** (`Type[DTypeFloat] | None`) – Data type used for the conversion to `np.ndarray`.

Returns Array a scaled to domain '**100**'.

Return type numpy.ndarray

Examples

With *Colour* domain-range scale set to '**Reference**':

```
>>> with domain_range_scale("Reference"):
...     to_domain_100(1)
...
array(1.0)
```

With *Colour* domain-range scale set to '**1**':

```
>>> with domain_range_scale("1"):
...     to_domain_100(1)
...
array(100.0)
```

With *Colour* domain-range scale set to '**100**' (unsupported):

```
>>> with domain_range_scale("100"):
...     to_domain_100(1)
...
array(1.0)
```

colour.utilities.to_domain_degrees

colour.utilities.**to_domain_degrees**(*a: ArrayLike*, *scale_factor: ArrayLike = 360*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a to degrees domain. The behaviour is as follows:

- If *Colour* domain-range scale is '**Reference**', the definition is almost entirely by-passed and will conveniently convert array a to `np.ndarray`.
- If *Colour* domain-range scale is '**1**', array a is multiplied by `scale_factor`, typically 360.
- If *Colour* domain-range scale is '**100**' (currently unsupported private value only used for unit tests), array a is multiplied by `scale_factor` / 100, typically 360 / 100.

Parameters

- **a** (ArrayLike) – Array a to scale to degrees domain.
- **scale_factor** (ArrayLike) – Scale factor, usually *numeric* but can be a numpy.ndarray if some axis need different scaling to be brought to degrees domain.
- **dtype** (Type[DTypeFloat] | None) – Data type used for the conversion to `np.ndarray`.

Returns Array a scaled to degrees domain.

Return type numpy.ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale("Reference"):
...     to_domain_degrees(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale("1"):
...     to_domain_degrees(1)
...
array(360.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale("100"):
...     to_domain_degrees(1)
...
array(3.6)
```

colour.utilities.to_domain_int

colour.utilities.**to_domain_int**(*a: ArrayLike*, *bit_depth: ArrayLike = 8*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a to int domain. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is almost entirely by-passed and will conveniently convert array a to `np.ndarray`.
- If *Colour* domain-range scale is **'1'**, array a is multiplied by $2^{bit\_depth} - 1$.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is multiplied by $2^{bit\_depth} - 1$.

Parameters

- **a** (`ArrayLike`) – Array a to scale to int domain.
- **bit_depth** (`ArrayLike`) – Bit-depth, usually *int* but can be a `numpy.ndarray` if some axis need different scaling to be brought to int domain.
- **dtype** (`Type[DTypeFloat] | None`) – Data type used for the conversion to `np.ndarray`.

Returns Array a scaled to int domain.

Return type `numpy.ndarray`

Notes

- To avoid precision issues and rounding, the scaling is performed on *float* numbers.

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale("Reference"):
...     to_domain_int(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale("1"):
...     to_domain_int(1)
...
array(255.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale("100"):
...     to_domain_int(1)
...
array(2.55)
```

colour.utilities.from_range_1

`colour.utilities.`**`from_range_1`**(*a: ArrayLike*, *scale_factor: ArrayLike = 100*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a from range **'1'**. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'** or **'1'**, the definition is entirely by-passed.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is multiplied by `scale_factor`, typically 100.

Parameters

- **a** (`ArrayLike`) – Array a to scale from range **'1'**.
- **scale_factor** (`ArrayLike`) – Scale factor, usually *numeric* but can be a `numpy.ndarray` if some axis need different scaling to be brought from range **'1'**.
- **dtype** (`Type[DTypeFloat] | None`) – Data type used for the conversion to `np.ndarray`.

Returns Array a scaled from range **'1'**.

Return type `numpy.ndarray`

Warning: The scale conversion of variable a happens in-place, i.e. a will be mutated!

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale("Reference"):
...     from_range_1(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale("1"):
...     from_range_1(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale("100"):
...     from_range_1(1)
...
array(100.0)
```

colour.utilities.from_range_10

colour.utilities.**from_range_10**(*a: ArrayLike*, *scale_factor: ArrayLike = 10*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a from range **'10'**, used by *Munsell Renotation System*. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is entirely by-passed.
- If *Colour* domain-range scale is **'1'**, array a is divided by `scale_factor`, typically 10.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is multiplied by `scale_factor`, typically 10.

Parameters

- **a** (ArrayLike) – Array a to scale from range **'10'**.
- **scale_factor** (ArrayLike) – Scale factor, usually *numeric* but can be a `numpy.ndarray` if some axis need different scaling to be brought from range **'10'**.
- **dtype** (Type[DTypeFloat] | None) – Data type used for the conversion to `np.ndarray`.

Returns Array a scaled from range **'10'**.

Return type `numpy.ndarray`

Warning: The scale conversion of variable a happens in-place, i.e. a will be mutated!

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale("Reference"):
...     from_range_10(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale("1"):
...     from_range_10(1)
...
array(0.1)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale("100"):
...     from_range_10(1)
...
array(10.0)
```

colour.utilities.from_range_100

`colour.utilities.`**`from_range_100`**(*a: ArrayLike*, *scale_factor: ArrayLike = 100*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a from range **'100'**. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'** or **'100'** (currently unsupported private value only used for unit tests), the definition is entirely by-passed.
- If *Colour* domain-range scale is **'1'**, array a is divided by `scale_factor`, typically 100.

Parameters

- **a** (`ArrayLike`) – Array a to scale from range **'100'**.
- **`scale_factor`** (`ArrayLike`) – Scale factor, usually *numeric* but can be a `numpy.ndarray` if some axis need different scaling to be brought from range **'100'**.
- **`dtype`** (`Type[DTypeFloat] | None`) – Data type used for the conversion to `np.ndarray`.

Returns Array a scaled from range **'100'**.

Return type `numpy.ndarray`

Warning: The scale conversion of variable a happens in-place, i.e. a will be mutated!

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale("Reference"):
...     from_range_100(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale("1"):
...     from_range_100(1)
...
array(0.01)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale("100"):
...     from_range_100(1)
...
array(1.0)
```

colour.utilities.from_range_degrees

colour.utilities.**from_range_degrees**(*a: ArrayLike*, *scale_factor: ArrayLike = 360*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a from degrees range. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is entirely by-passed.
- If *Colour* domain-range scale is **'1'**, array a is divided by `scale_factor`, typically 360.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is divided by `scale_factor` / 100, typically 360 / 100.

Parameters

- **a** (`ArrayLike`) – Array a to scale from degrees range.
- **scale_factor** (`ArrayLike`) – Scale factor, usually *numeric* but can be a `numpy.ndarray` if some axis need different scaling to be brought from degrees range.
- **dtype** (`Type[DTypeFloat] | None`) – Data type used for the conversion to `np.ndarray`.

Returns Array a scaled from degrees range.

Return type `numpy.ndarray`

Warning: The scale conversion of variable a happens in-place, i.e. a will be mutated!

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale("Reference"):
...     from_range_degrees(1)
...
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale("1"):
...     from_range_degrees(1)
...
array(0.0027777...)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale("100"):
...     from_range_degrees(1)
...
array(0.2777777...)
```

colour.utilities.from_range_int

colour.utilities.**from_range_int**(*a: ArrayLike*, *bit_depth: ArrayLike = 8*, *dtype: Type[DTypeFloat] | None = None*) → NDArray

Scale given array a from int range. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is entirely by-passed.
- If *Colour* domain-range scale is **'1'**, array a is converted to `np.ndarray` and divided by $2^{bit\_depth} - 1$.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is converted to `np.ndarray` and divided by $2^{bit\_depth} - 1$.

Parameters

- **a** (`ArrayLike`) – Array a to scale from int range.
- **bit_depth** (`ArrayLike`) – Bit-depth, usually *int* but can be a `numpy.ndarray` if some axis need different scaling to be brought from int range.
- **dtype** (`Type[DTypeFloat] | None`) – Data type used for the conversion to `np.ndarray`.

Returns Array a scaled from int range.

Return type `numpy.ndarray`

> **Warning:** The scale conversion of variable a happens in-place, i.e. a will be mutated!

Notes

- To avoid precision issues and rounding, the scaling is performed on *float* numbers.

Examples

With *Colour* domain-range scale set to '**Reference**':

```
>>> with domain_range_scale("Reference"):
...     from_range_int(1)
...
array(1.0)
```

With *Colour* domain-range scale set to '**1**':

```
>>> with domain_range_scale("1"):
...     from_range_int(1)
...
array(0.0039215...)
```

With *Colour* domain-range scale set to '**100**' (unsupported):

```
>>> with domain_range_scale("100"):
...     from_range_int(1)
...
array(0.3921568...)
```

colour.utilities.is_ndarray_copy_enabled

colour.utilities.**is_ndarray_copy_enabled**() → bool

Return whether *Colour* `numpy.ndarray` copy is enabled: Various API objects return a copy of their internal `numpy.ndarray` for safety purposes but this can be a slow operation impacting performance.

Returns Whether *Colour* `numpy.ndarray` copy is enabled.

Return type `bool`

Examples

```
>>> with ndarray_copy_enable(False):
...     is_ndarray_copy_enabled()
...
False
>>> with ndarray_copy_enable(True):
...     is_ndarray_copy_enabled()
...
True
```

colour.utilities.set_ndarray_copy_enable

colour.utilities.**set_ndarray_copy_enable**(*enable: bool*)

Set *Colour* numpy.ndarray copy enabled state.

Parameters **enable** (bool) – Whether to enable *Colour* numpy.ndarray copy.

Examples

```
>>> with ndarray_copy_enable(is_ndarray_copy_enabled()):
...     print(is_ndarray_copy_enabled())
...     set_ndarray_copy_enable(False)
...     print(is_ndarray_copy_enabled())
...
True
False
```

colour.utilities.ndarray_copy_enable

class colour.utilities.**ndarray_copy_enable**(*enable: bool*)

Define a context manager and decorator temporarily setting *Colour* numpy.ndarray copy enabled state.

Parameters **enable** (bool) – Whether to enable or disable *Colour* numpy.ndarray copy.

Return type None

__init__(*enable: bool*) → None

Parameters **enable** (bool) –

Return type None

Methods

__init__(enable)

colour.utilities.ndarray_copy

colour.utilities.**ndarray_copy**(*a: numpy.ndarray[Any, numpy.dtype[numpy._typing._array_like._ScalarType_co]]*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Return a numpy.ndarray copy if the relevant *Colour* state is enabled: Various API objects return a copy of their internal numpy.ndarray for safety purposes but this can be a slow operation impacting performance.

Parameters **a** (numpy.ndarray[Any, numpy.dtype[numpy._typing._array_like._ScalarType_co]]) – Array a to return a copy of.

Returns Array a copy according to *Colour* state.

Return type numpy.ndarray

Examples

```
>>> a = np.linspace(0, 1, 10)
>>> id(a) == id(ndarray_copy(a))
False
>>> with ndarray_copy_enable(False):
...     id(a) == id(ndarray_copy(a))
...
True
```

colour.utilities.closest_indexes

colour.utilities.**closest_indexes**(*a: ArrayLike*, *b: ArrayLike*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Return the array a closest element indexes to the reference array b elements.

Parameters

- **a** (ArrayLike) – Array a to search for the closest element indexes.
- **b** (ArrayLike) – Reference array b.

Returns Closest array a element indexes.

Return type numpy.ndarray

Examples

```
>>> a = np.array(
...     [
...         24.31357115,
...         63.62396289,
...         55.71528816,
...         62.70988028,
...         46.84480573,
...         25.40026416,
...     ]
... )
>>> print(closest_indexes(a, 63))
[3]
>>> print(closest_indexes(a, [63, 25]))
[3 5]
```

colour.utilities.closest

colour.utilities.**closest**(*a: ArrayLike*, *b: ArrayLike*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Return the closest array a elements to the reference array b elements.

Parameters

- **a** (ArrayLike) – Array a to search for the closest element.
- **b** (ArrayLike) – Reference array b.

Returns Closest array a elements.

Return type numpy.ndarray

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.interval

`colour.utilities.interval`(*distribution: ArrayLike*, *unique: bool = True*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Return the interval size of given distribution.

Parameters

- **distribution** (`ArrayLike`) – Distribution to retrieve the interval.
- **unique** (`bool`) – Whether to return unique intervals if the distribution is non-uniformly spaced or the complete intervals

Returns Distribution interval.

Return type `numpy.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: ArrayLike) → bool`

Return whether given distribution is uniform.

Parameters **distribution** (ArrayLike) – Distribution to check the uniformity of.

Returns Whether 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.has_only_nan

`colour.utilities.has_only_nan(a: ArrayLike) → bool`

Return whether given array a contains only NaN values.

Parameters **a** (ArrayLike) – Array a to check whether it contains only NaN values.

Returns Whether array a contains only NaN values.

Return type bool

Examples

```
>>> has_only_nan(None)
True
>>> has_only_nan([None, None])
True
>>> has_only_nan([True, None])
False
>>> has_only_nan([0.1, np.nan, 0.3])
False
```

colour.utilities.in_array

colour.utilities.**in_array**(*a: ArrayLike*, *b: ArrayLike*, *tolerance: Real = EPSILON*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Return whether each element of the array a is also present in the array b within given tolerance.

Parameters

- **a** (ArrayLike) – Array a to test the elements from.
- **b** (ArrayLike) – The array b against which to test the elements of array a.
- **tolerance** (Real) – Tolerance value.

Returns A bool array with array a shape describing whether an element of array a is present in array b within given tolerance.

Return type numpy.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: ArrayLike*, *dtype: Type[DTypeBoolean] | Type[DTypeReal] | None = None*) → NDArray

Stack given array of arrays a along the last axis (tail) to produce a stacked array.

It is used to stack an array of arrays produced by the colour.utilities.tsplit() definition.

Parameters

- **a** (ArrayLike) – Array of arrays a to stack along the last axis.
- **dtype** (Type[DTypeBoolean] | Type[DTypeReal] | None) – numpy.dtype to use for initial conversion to numpy.ndarray, default to the numpy.dtype defined by colour.constant.DEFAULT_FLOAT_DTYPE attribute.

Returns Stacked array.

Return type numpy.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: ArrayLike*, *dtype: Type[DTypeBoolean] | Type[DTypeReal] | None = None*) → NDArray

Split given stacked array a along the last axis (tail) to produce an array of arrays.

It is used to split a stacked array produced by the `colour.utilities.tstack()` definition.

Parameters

- **a** (ArrayLike) – Stacked array a to split.
- **dtype** (Type[DTypeBoolean] | Type[DTypeReal] | None) – `numpy.dtype` to use for initial conversion to `numpy.ndarray`, default to the `numpy.dtype` defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.

Returns Array of arrays.

Return type `numpy.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: ArrayLike*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Return the rows of given array a as diagonal matrices.

Parameters **a** (ArrayLike) – Array a to returns the rows of as diagonal matrices.

Returns Array a rows as diagonal matrices.

Return type numpy.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.orient

colour.utilities.**orient**(*a: ArrayLike, orientation: Union[Literal['Ignore', 'Flip', 'Flop', '90 CW', '90 CCW', '180'], str] = 'Ignore'*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Orient given array a according to given orientation.

Parameters

- **a** (ArrayLike) – Array a to orient.
- **orientation** (Union[Literal['Ignore', 'Flip', 'Flop', '90 CW', '90 CCW', '180'], str]) – Orientation to perform.

Returns Oriented array.

Return type numpy.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: ArrayLike*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Return the centroid indexes of given array a.

Parameters **a** (`ArrayLike`) – Array a to returns the centroid indexes of.

Returns Array a centroid indexes.

Return type `numpy.ndarray`

Examples

```
>>> a = np.tile(np.arange(0, 5), (5, 1))
>>> centroid(a)
array([2, 3]...)
```

colour.utilities.fill_nan

`colour.utilities.`**`fill_nan`**(*a: ArrayLike, method: Union[Literal['Interpolation', 'Constant'], str] = 'Interpolation', default: Real = 0*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Fill given array a NaN values according to given method.

Parameters

- **a** (`ArrayLike`) – Array a to fill the NaNs of.
- **method** (`Union[Literal['Interpolation', 'Constant'], str]`) – *Interpolation* method linearly interpolates through the NaN values, *Constant* method replaces NaN values with `default`.
- **default** (`Real`) – Value to use with the *Constant* method.

Returns NaNs filled array a.

Return type numpy.ndarray

Examples

```
>>> a = np.array([0.1, 0.2, np.nan, 0.4, 0.5])
>>> fill_nan(a)
array([ 0.1,  0.2,  0.3,  0.4,  0.5])
>>> fill_nan(a, method="Constant")
array([ 0.1,  0.2,  0. ,  0.4,  0.5])
```

colour.utilities.ndarray_write

colour.utilities.**ndarray_write**(*a: ArrayLike*) → *collections.abc.Generator*

Define a context manager setting given array a writeable to operate one and then read-only.

Parameters **a** (ArrayLike) – Array a to operate on.

Yields *Generator* – Array a operated.

Return type *collections.abc.Generator*

Examples

```
>>> a = np.linspace(0, 1, 10)
>>> a.setflags(write=False)
>>> try:
...     a += 1
... except ValueError:
...     pass
...
>>> with ndarray_write(a):
...     a += 1
...
```

colour.utilities.zeros

colour.utilities.**zeros**(*shape: int | Tuple[int, ...], dtype: Type[DTypeReal] | None = None, order: Literal['C', 'F'] = 'C'*) → NDArray

Wrap np.zeros() definition to create an array with the active numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.

Parameters

- **shape** (int | Tuple[int, ...]) – Shape of the new array, e.g., (2, 3) or 2.
- **dtype** (Type[DTypeReal] | None) – numpy.dtype to use for conversion, default to the numpy.dtype defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.
- **order** (Literal['C', 'F']) – Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

Returns Array of given shape and numpy.dtype, filled with zeros.

Return type numpy.ndarray

Examples

```
>>> zeros(3)
array([ 0.,  0.,  0.])
```

colour.utilities.ones

colour.utilities.**ones**(*shape: int | Tuple[int, ...], dtype: Type[DTypeReal] | None = None, order: Literal['C', 'F'] = 'C'*) → NDArray

Wrap `np.ones()` definition to create an array with the active `numpy.dtype` defined by the `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.

Parameters

- **shape** (`int | Tuple[int, ...]`) – Shape of the new array, e.g., `(2, 3)` or `2`.
- **dtype** (`Type[DTypeReal] | None`) – `numpy.dtype` to use for conversion, default to the `numpy.dtype` defined by the `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.
- **order** (`Literal['C', 'F']`) – Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

Returns Array of given shape and type, filled with ones.

Return type `numpy.ndarray`

Examples

```
>>> ones(3)
array([ 1.,  1.,  1.])
```

colour.utilities.full

colour.utilities.**full**(*shape: int | Tuple[int, ...], fill_value: Real, dtype: Type[DTypeReal] | None = None, order: Literal['C', 'F'] = 'C'*) → NDArray

Wrap `np.full()` definition to create an array with the active type defined by the:attr:*colour.constant.DEFAULT_FLOAT_DTYPE* attribute.

Parameters

- **shape** (`int | Tuple[int, ...]`) – Shape of the new array, e.g., `(2, 3)` or `2`.
- **fill_value** (`Real`) – Fill value.
- **dtype** (`Type[DTypeReal] | None`) – `numpy.dtype` to use for conversion, default to the `numpy.dtype` defined by the `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.
- **order** (`Literal['C', 'F']`) – Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

Returns Array of given shape and `numpy.dtype`, filled with given value.

Return type `numpy.ndarray`

Examples

```
>>> ones(3)
array([ 1.,  1.,  1.])
```

colour.utilities.index_along_last_axis

colour.utilities.**index_along_last_axis**(*a: ArrayLike*, *indexes: ArrayLike*) → numpy.ndarray[*Any*, numpy.dtype[numpy._typing._array_like._ScalarType_co]]

Reduce the dimension of array a by one, by using an array of indexes to pick elements off the last axis.

Parameters

- **a** (ArrayLike) – Array a to be indexed.
- **indexes** (ArrayLike) – *integer* array with the same shape as *a* but with one dimension fewer, containing indices to the last dimension of *a*. All elements must be numbers between *0* and *m* - 1.

Returns Indexed array a.

Return type numpy.ndarray

Raises

- **ValueError** – If the array a and indexes have incompatible shapes.
- **IndexError** – If indexes has elements outside of the allowed range of 0 to *m* - 1 or if it's not an *integer* array.

Examples

```
>>> a = np.array(
...     [
...         [
...             [0.3, 0.5, 6.9],
...             [3.3, 4.4, 1.6],
...             [4.4, 7.5, 2.3],
...             [2.3, 1.6, 7.4],
...         ],
...         [
...             [2.0, 5.9, 2.8],
...             [6.2, 4.9, 8.6],
...             [3.7, 9.7, 7.3],
...             [6.3, 4.3, 3.2],
...         ],
...         [
...             [0.8, 1.9, 0.7],
...             [5.6, 4.0, 1.7],
...             [6.7, 8.2, 1.7],
...             [1.2, 7.1, 1.4],
...         ],
...         [
...             [4.0, 4.8, 8.9],
...             [4.0, 0.3, 6.9],
...             [3.5, 7.1, 4.5],
...             [1.4, 1.9, 1.6],
```

(continues on next page)

(continued from previous page)

```
...         ],
...     ]
... )
>>> indexes = np.array(
...     [[2, 0, 1, 1], [2, 1, 1, 0], [0, 0, 1, 2], [0, 0, 1, 2]]
... )
>>> index_along_last_axis(a, indexes)
array([[ 6.9,  3.3,  7.5,  1.6],
       [ 2.8,  4.9,  9.7,  6.3],
       [ 0.8,  5.6,  8.2,  1.4],
       [ 4. ,  4. ,  7.1,  1.6]])
```

This function can be used to compute the result of `np.min()` along the last axis given the corresponding `np.argmin()` indexes.

```
>>> indexes = np.argmin(a, axis=-1)
>>> np.array_equal(index_along_last_axis(a, indexes), np.min(a, axis=-1))
True
```

In particular, this can be used to manipulate the indexes given by functions like `np.min()` before indexing the array. For example, to get elements directly following the smallest elements:

```
>>> index_along_last_axis(a, (indexes + 1) % 3)
array([[ 0.5,  3.3,  4.4,  7.4],
       [ 5.9,  8.6,  9.7,  6.3],
       [ 0.8,  5.6,  6.7,  7.1],
       [ 4.8,  6.9,  7.1,  1.9]])
```

colour.utilities.format_array_as_row

`colour.utilities.`**`format_array_as_row`**(*a: ArrayLike*, *decimals: int = 7*, *separator: str = ' '*) → str

Format given array a as a row.

Parameters

- **a** (`ArrayLike`) – Array to format.
- **decimals** (`int`) – Decimal count to use when formatting as a row.
- **separator** (`str`) – Separator used to join the array a items.

Returns Array formatted as a row.

Return type `str`

Examples

```
>>> format_array_as_row([1.25, 2.5, 3.75])
'1.2500000 2.5000000 3.7500000'
>>> format_array_as_row([1.25, 2.5, 3.75], 3)
'1.250 2.500 3.750'
>>> format_array_as_row([1.25, 2.5, 3.75], 3, ", ")
'1.250, 2.500, 3.750'
```

Metrics

colour.utilities

metric_mse(a, b[, axis])	Compute the mean squared error (MSE) or mean squared deviation (MSD) between given variables a and b.
metric_psnr(a, b[, max_a, axis])	Compute the peak signal-to-noise ratio (PSNR) between given variables a and b.

colour.utilities.metric_mse

colour.utilities.**metric_mse**(*a: ArrayLike*, *b: ArrayLike*, *axis: Optional[Union[int, Tuple[int]]] = None*) → NDArrayFloat

Compute the mean squared error (MSE) or mean squared deviation (MSD) between given variables a and b.

Parameters

- **a** (ArrayLike) – Variable a.
- **b** (ArrayLike) – Variable b.
- **axis** (Optional[Union[int, Tuple[int]]]) – Axis or axes along which the means are computed. The default is to compute the mean of the flattened array. If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.

Returns Mean squared error (MSE).

Return type numpy.ndarray

References

[Wikipedia03c]

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> b = a * 0.9
>>> metric_mse(a, b)
0.0012714...
```

colour.utilities.metric_psnr

colour.utilities.**metric_psnr**(*a: ArrayLike*, *b: ArrayLike*, *max_a: Real = 1*, *axis: Optional[Union[int, Tuple[int]]] = None*) → NDArrayFloat

Compute the peak signal-to-noise ratio (PSNR) between given variables a and b.

Parameters

- **a** (ArrayLike) – Variable a.
- **b** (ArrayLike) – Variable b.
- **max_a** (Real) – Maximum possible pixel value of the a variable.

- **axis** (Optional[Union[int, Tuple[int]]]) – Axis or axes along which the means are computed. The default is to compute the mean of the flattened array. If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.

Returns Peak signal-to-noise ratio (PSNR).

Return type numpy.ndarray

References

[Wikipedia04a]

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> b = a * 0.9
>>> metric_psnr(a, b)
28.9568515...
```

Data Structures

colour.utilities

CanonicalMapping([data])	Implement a delimiter and case-insensitive dict-like object with support for slugs, i.e. *SEO* friendly and human-readable version of the keys but also canonical keys, i.e. slugified keys without delimiters.
LazyCanonicalMapping([data])	Implement a lazy delimiter and case-insensitive dict-like object inheriting from colour.utilities.CanonicalMapping class.
Lookup	Extend dict type to provide a lookup by value(s).
Node(*args, **kwargs)	Represent a basic node supporting the creation of basic node trees.
Structure(*args, **kwargs)	Define a dict-like object allowing to access key values using dot syntax.

colour.utilities.CanonicalMapping

class colour.utilities.**CanonicalMapping**(*data: collections.abc.Generator | collections.abc.Mapping | None = None, **kwargs: Any*)

Bases: collections.abc.MutableMapping

Implement a delimiter and case-insensitive dict-like object with support for slugs, i.e. *SEO* friendly and human-readable version of the keys but also canonical keys, i.e. slugified keys without delimiters.

The item keys are expected to be str-like objects thus supporting the str.lower() method. Setting items is done by using the given keys. Retrieving or deleting an item and testing whether an item exist is done by transforming the item's key in a sequence as follows:

- *Original Key*
- *Lowercase Key*

- *Slugified Key*
- *Canonical Key*

For example, given the `McCamy 1992` key:

- *Original Key* : `McCamy 1992`
- *Lowercase Key* : `mccamy 1992`
- *Slugified Key* : `mccamy-1992`
- *Canonical Key* : `mccamy1992`

Parameters

- **data** (`Generator | Mapping | None`) – Data to store into the delimiter and case-insensitive `dict`-like object at initialisation.
- **kwargs** (`Any`) – Key / value pairs to store into the mapping at initialisation.

Return type None

Attributes

- `data`

Methods

- `__init__()`
- `__repr__()`
- `__setitem__()`
- `__getitem__()`
- `__delitem__()`
- `__contains__()`
- `__iter__()`
- `__len__()`
- `__eq__()`
- `__ne__()`
- `copy()`
- `lower_keys()`
- `lower_items()`
- `slugified_keys()`
- `slugified_items()`
- `canonical_keys()`
- `canonical_items()`

Examples

```
>>> methods = CanonicalMapping({"McCamy 1992": 1, "Hernandez 1999": 2})
>>> methods["mccamy 1992"]
1
>>> methods["MCCAMY 1992"]
1
>>> methods["mccamy-1992"]
1
>>> methods["mccamy1992"]
1
```

__init__(*data: collections.abc.Generator | collections.abc.Mapping | None = None*, ***kwargs: Any*) → None

Parameters

- **data** (`collections.abc.Generator | collections.abc.Mapping | None`) –
- **kwargs** (`Any`) –

Return type None

property data: dict

Getter property for the delimiter and case-insensitive `dict`-like object data.

Returns Data.

Return type `dict`

__repr__() → str

Return an evaluable string representation of the delimiter and case-insensitive `dict`-like object.

Returns Evaluable string representation.

Return type `str`

__setitem__(*item: str | Any*, *value: Any*)

Set given item with given value in the delimiter and case-insensitive `dict`-like object.

Parameters

- **item** (`str | Any`) – Item to set in the delimiter and case-insensitive `dict`-like object.
- **value** (`Any`) – Value to store in the delimiter and case-insensitive `dict`-like object.

__getitem__(*item: str | Any*) → *Any*

Return the value of given item from the delimiter and case-insensitive `dict`-like object.

Parameters **item** (`str | Any`) – Item to retrieve the value of from the delimiter and case-insensitive `dict`-like object.

Returns Item value.

Return type `object`

Notes

- The item value can be retrieved by using either its lower-case, slugified or canonical variant.

`__delitem__`(*item: str* | Any)

Delete given item from the delimiter and case-insensitive `dict`-like object.

Parameters **`item`** (`str | Any`) – Item to delete from the delimiter and case-insensitive `dict`-like object.

Notes

- The item can be deleted by using either its lower-case, slugified or canonical variant.

`__contains__`(*item: str* | Any) → bool

Return whether the delimiter and case-insensitive `dict`-like object contains given item.

Parameters **`item`** (`str | Any`) – Item to find whether it is in the delimiter and case-insensitive `dict`-like object.

Returns Whether given item is in the delimiter and case-insensitive `dict`-like object.

Return type `bool`

Notes

- The item presence can be checked by using either its lower-case, slugified or canonical variant.

`__iter__`() → *collections.abc.Generator*

Iterate over the items of the delimiter and case-insensitive `dict`-like object.

Yields *Generator* – Item generator.

Return type *collections.abc.Generator*

Notes

- The iterated items are the original items.

`__len__`() → int

Return the items count.

Returns Items count.

Return type `int`

`__eq__`(*other:* Any) → bool

Return whether the delimiter and case-insensitive `dict`-like object is equal to given other object.

Parameters **`other`** (`Any`) – Object to test whether it is equal to the delimiter and case-insensitive `dict`-like object

Returns Whether given object is equal to the delimiter and case-insensitive `dict`-like object.

Return type `bool`

__ne__(*other: Any*) → bool

Return whether the delimiter and case-insensitive `dict`-like object is not equal to given other object.

Parameters **other** (Any) – Object to test whether it is not equal to the delimiter and case-insensitive `dict`-like object

Returns Whether given object is not equal to the delimiter and case-insensitive `dict`-like object.

Return type `bool`

copy() → *colour.utilities.data_structures.CanonicalMapping*

Return a copy of the delimiter and case-insensitive `dict`-like object.

Returns Case-insensitive `dict`-like object copy.

Return type `CanonicalMapping`

> **Warning:**
>
> - The `CanonicalMapping` class copy returned is a *copy* of the object not a *deepcopy*!

lower_keys() → *collections.abc.Generator*

Iterate over the lower-case keys of the delimiter and case-insensitive `dict`-like object.

Yields *Generator* – Item generator.

Return type *collections.abc.Generator*

lower_items() → *collections.abc.Generator*

Iterate over the lower-case items of the delimiter and case-insensitive `dict`-like object.

Yields *Generator* – Item generator.

Return type *collections.abc.Generator*

slugified_keys() → *collections.abc.Generator*

Iterate over the slugified keys of the delimiter and case-insensitive `dict`-like object.

Yields *Generator* – Item generator.

Return type *collections.abc.Generator*

slugified_items() → *collections.abc.Generator*

Iterate over the slugified items of the delimiter and case-insensitive `dict`-like object.

Yields *Generator* – Item generator.

Return type *collections.abc.Generator*

canonical_keys() → *collections.abc.Generator*

Iterate over the canonical keys of the delimiter and case-insensitive `dict`-like object.

Yields *Generator* – Item generator.

Return type *collections.abc.Generator*

canonical_items() → *collections.abc.Generator*

Iterate over the canonical items of the delimiter and case-insensitive `dict`-like object.

Yields *Generator* – Item generator.

Return type *collections.abc.Generator*

__hash__ = None

__weakref__
list of weak references to the object (if defined)

colour.utilities.LazyCanonicalMapping

class colour.utilities.**LazyCanonicalMapping**(*data: collections.abc.Generator | collections.abc.Mapping | None = None*, ***kwargs: Any*)

Bases: `colour.utilities.data_structures.CanonicalMapping`

Implement a lazy delimiter and case-insensitive `dict`-like object inheriting from `colour.utilities.CanonicalMapping` class.

The lazy retrieval is performed as follows: If the value is a callable, then it is evaluated and its return value is stored in place of the current value.

Parameters

- **data** (`Generator | Mapping | None`) – Data to store into the lazy delimiter and case-insensitive `dict`-like object at initialisation.
- **kwargs** (`Any`) – Key / value pairs to store into the mapping at initialisation.

Return type None

Methods

- `__getitem__()`

Examples

```
>>> def callable_a():
...     print(2)
...     return 2
...
>>> methods = LazyCanonicalMapping({"McCamy": 1, "Hernandez": callable_a})
>>> methods["mccamy"]
1
>>> methods["hernandez"]
2
2
```

__getitem__(*item: str | Any*) → *Any*
Return the value of given item from the lazy delimiter and case-insensitive `dict`-like object.

Parameters **item** (`str | Any`) – Item to retrieve the value of from the lazy delimiter and case-insensitive `dict`-like object.

Returns Item value.

Return type `object`

colour.utilities.Lookup

class colour.utilities.**Lookup**

Bases: dict

Extend dict type to provide a lookup by value(s).

Methods

- keys_from_value()
- first_key_from_value()

References

[Mana]

Examples

```
>>> person = Lookup(first_name="John", last_name="Doe", gender="male")
>>> person.first_key_from_value("John")
'first_name'
>>> persons = Lookup(John="Doe", Jane="Doe", Luke="Skywalker")
>>> sorted(persons.keys_from_value("Doe"))
['Jane', 'John']
```

keys_from_value(*value: Any*) → list

Get the keys associated with given value.

Parameters **value** (Any) – Value to find the associated keys.

Returns Keys associated with given value.

Return type list

first_key_from_value(*value: Any*) → *Any*

Get the first key associated with given value.

Parameters **value** (Any) – Value to find the associated first key.

Returns First key associated with given value.

Return type object

__weakref__

list of weak references to the object (if defined)

colour.utilities.Node

class colour.utilities.**Node**(**args: Any*, ***kwargs: Any*)

Bases: object

Represent a basic node supporting the creation of basic node trees.

Parameters

- **name** – Node name.
- **parent** – Parent of the node.
- **children** – Children of the node.

- **data** – The data belonging to this node.
- **args** (Any) –
- **kwargs** (Any) –

Return type Self

Attributes

- name
- parent
- children
- id
- root
- leaves
- siblings
- data

Methods

- __new__()
- __init__()
- __str__()
- __len__()
- is_root()
- is_inner()
- is_leaf()
- walk()
- render()

Examples

```
>>> node_a = Node("Node A")
>>> node_b = Node("Node B", node_a)
>>> node_c = Node("Node C", node_a)
>>> node_d = Node("Node D", node_b)
>>> node_e = Node("Node E", node_b)
>>> node_f = Node("Node F", node_d)
>>> node_g = Node("Node G", node_f)
>>> node_h = Node("Node H", node_g)
>>> [node.name for node in node_a.leaves]
['Node H', 'Node E', 'Node C']
>>> print(node_h.root.name)
Node A
>>> len(node_a)
7
```

Return a new instance of the colour.utilities.Node class.

Parameters

- **args** (Any) – Arguments.
- **kwargs** (Any) – Keywords arguments.

Return type Self

static __new__(*cls*, **args: Any*, ***kwargs: Any*) → Self

Return a new instance of the `colour.utilities.Node` class.

Parameters

- **args** (Any) – Arguments.
- **kwargs** (Any) – Keywords arguments.

Return type *Self*

__init__(*name: str | None = None*, *parent: Optional[Self] = None*, *children: Optional[List[Self]] = None*, *data: Any | None = None*) → None

Parameters

- **name** (`str | None`) –
- **parent** (`Optional[Self]`) –
- **children** (`Optional[List[Self]]`) –
- **data** (`Any | None`) –

Return type None

property name: str

Getter and setter property for the name.

Parameters **value** – Value to set the name with.

Returns Node name.

Return type `str`

property parent: Optional[Self]

Getter and setter property for the node parent.

Parameters **value** – Parent to set the node with.

Returns Node parent.

Return type `Node` or `None`

property children: List[Self]

Getter and setter property for the node children.

Parameters **value** – Children to set the node with.

Returns Node children.

Return type `list`

property id: int

Getter property for the node id.

Returns Node id.

Return type `int`

property root: Self

Getter property for the node tree.

Returns Node root.

Return type Node

property leaves: collections.abc.Generator

Getter property for the node leaves.

Yields *Generator* – Node leaves.

property siblings: collections.abc.Generator

Getter property for the node siblings.

Returns Node siblings.

Return type *Generator*

__weakref__

list of weak references to the object (if defined)

property data: Any

Getter property for the node data.

Returns Node data.

Return type object

__str__() → str

Return a formatted string representation of the node.

Returns Formatted string representation.

Return type class`str`

__len__() → int

Return the number of children of the node.

Returns Number of children of the node.

Return type int

is_root() → bool

Return whether the node is a root node.

Returns Whether the node is a root node.

Return type bool

Examples

```
>>> node_a = Node("Node A")
>>> node_b = Node("Node B", node_a)
>>> node_c = Node("Node C", node_b)
>>> node_a.is_root()
True
>>> node_b.is_root()
False
```

is_inner() → bool

Return whether the node is an inner node.

Returns Whether the node is an inner node.

Return type bool

Examples

```
>>> node_a = Node("Node A")
>>> node_b = Node("Node B", node_a)
>>> node_c = Node("Node C", node_b)
>>> node_a.is_inner()
False
>>> node_b.is_inner()
True
```

is_leaf() → bool

Return whether the node is a leaf node.

Returns Whether the node is a leaf node.

Return type bool

Examples

```
>>> node_a = Node("Node A")
>>> node_b = Node("Node B", node_a)
>>> node_c = Node("Node C", node_b)
>>> node_a.is_leaf()
False
>>> node_c.is_leaf()
True
```

walk(*ascendants: bool = False*) → *collections.abc.Generator*

Return a generator used to walk into colour.utilities.Node trees.

Parameters **ascendants** (bool) – Whether to walk up the node tree.

Yields *Generator* – Node tree walker.

Return type *collections.abc.Generator*

Examples

```
>>> node_a = Node("Node A")
>>> node_b = Node("Node B", node_a)
>>> node_c = Node("Node C", node_a)
>>> node_d = Node("Node D", node_b)
>>> node_e = Node("Node E", node_b)
>>> node_f = Node("Node F", node_d)
>>> node_g = Node("Node G", node_f)
>>> node_h = Node("Node H", node_g)
>>> for node in node_a.walk():
...     print(node.name)
...
Node B
Node D
Node F
Node G
Node H
Node E
Node C
```

render(*tab_level: int = 0*)

Render the current node and its children as a string.

Parameters **tab_level** (int) – Initial indentation level

Returns Rendered node tree.

Return type str

Examples

```
>>> node_a = Node("Node A")
>>> node_b = Node("Node B", node_a)
>>> node_c = Node("Node C", node_a)
>>> print(node_a.render())
|----"Node A"
    |----"Node B"
    |----"Node C"
```

colour.utilities.Structure

class colour.utilities.**Structure**(**args: Any*, ***kwargs: Any*)

Bases: dict

Define a dict-like object allowing to access key values using dot syntax.

Parameters

- **args** (Any) – Arguments.
- **kwargs** (Any) – Key / value pairs.

Return type None

Methods

- `__init__()`
- `__setattr__()`
- `__delattr__()`
- `__dir__()`
- `__getattr__()`
- `__setstate__()`

References

[Rak17]

Examples

```
>>> person = Structure(first_name="John", last_name="Doe", gender="male")
>>> person.first_name
'John'
>>> sorted(person.keys())
['first_name', 'gender', 'last_name']
>>> person["gender"]
'male'
```

`__init__`(**args: Any*, ***kwargs: Any*) → None

Parameters

- **args** (Any) –
- **kwargs** (Any) –

Return type None

`__setattr__`(*name: str*, *value: Any*)

Assign given value to the attribute with given name.

Parameters

- **name** (str) – Name of the attribute to assign the `value` to.
- **value** (Any) – Value to assign to the attribute.

`__delattr__`(*name: str*)

Delete the attribute with given name.

Parameters **name** (str) – Name of the attribute to delete.

`__dir__`() → *collections.abc.Iterable*

Return a list of valid attributes for the `dict`-like object.

Returns List of valid attributes for the `dict`-like object.

Return type `list`

`__getattr__`(*name: str*) → *Any*

Return the value from the attribute with given name.

Parameters **name** (str) – Name of the attribute to get the value from.

Return type `object`

Raises **`AttributeError`** – If the attribute is not defined.

`__setstate__`(*state*)

Set the object state when unpickling.

`__weakref__`

list of weak references to the object (if defined)

Verbose

colour.utilities

message_box(message[, width, padding, ...])	Print a message inside a box.
show_warning(message, category, filename, lineno)	Alternative warnings.showwarning() definition that allows traceback printing.
warning(*args, **kwargs)	Issue a warning.
filter_warnings([colour_runtime_warnings, ...])	Filter *Colour* and also optionally overall Python warnings.
suppress_warnings([colour_runtime_warnings, ...])	Define a context manager filtering *Colour* and also optionally overall Python warnings.
numpy_print_options(*args, **kwargs)	Define a context manager implementing context changes to *Numpy* print behaviour.
describe_environment([runtime_packages, ...])	Describe *Colour* running environment, i.e. interpreter, runtime and development packages.
multiline_str(object_, attributes[, ...])	Return a formatted string representation of the given object.
multiline_repr(object_, attributes[, ...])	Return an (almost) evaluable string representation of the given object.

colour.utilities.message_box

colour.utilities.**message_box**(*message: str*, *width: int = 79*, *padding: int = 3*, *print_callable: Callable = print*)

Print a message inside a box.

Parameters

- **message** (str) – Message to print.
- **width** (int) – Message box width.
- **padding** (int) – Padding on each side of the message.
- **print_callable** (Callable) – Callable used to print the message box.

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.           *
*                                                                         *
===========================================================================
>>> message_box(message, width=60)
============================================================
*                                                          *
*   Lorem ipsum dolor sit amet, consectetur adipiscing     *
*   elit, sed do eiusmod tempor incididunt ut labore et    *
```

(continues on next page)

(continued from previous page)

```
*   dolore magna aliqua.                                    *
*                                                           *
=============================================================
>>> 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.                                                  *
*                                                                         *
===========================================================================
```

colour.utilities.show_warning

colour.utilities.**show_warning**(*message: Warning | str, category: Type[Warning], filename: str, lineno: int, file: TextIO | None = None, line: str | None = None*) → None

Alternative `warnings.showwarning()` definition that allows traceback printing.

This definition is expected to be used by setting the *COLOUR_SCIENCE__COLOUR__SHOW_WARNINGS_WITH_TRAC* environment variable prior to importing *colour*.

Parameters

- **message** (`Warning | str`) – Warning message.
- **category** (`Type[Warning]`) – `Warning` sub-class.
- **filename** (`str`) – File path to read the line at `lineno` from if `line` is None.
- **lineno** (`int`) – Line number to read the line at in `filename` if `line` is None.
- **file** (`TextIO | None`) – `file` object to write the warning to, defaults to `sys.stderr` attribute.
- **line** (`str | None`) – Source code to be included in the warning message.

Return type None

Notes

- Setting the *COLOUR_SCIENCE__COLOUR__SHOW_WARNINGS_WITH_TRACEBACK* environment variable will result in the `warnings.showwarning()` definition to be replaced with the `colour.utilities.show_warning()` definition and thus providing complete traceback from the point where the warning occurred.

colour.utilities.warning

colour.utilities.**warning**(**args: Any, **kwargs: Any*)

Issue a warning.

Parameters

- **args** (`Any`) – Arguments.
- **kwargs** (`Any`) – Keywords arguments.

Examples

```
>>> warning("This is a warning!")
```

colour.utilities.filter_warnings

colour.utilities.**filter_warnings**(*colour_runtime_warnings: Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]] = None, colour_usage_warnings: Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]] = None, colour_warnings: Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]] = None, python_warnings: Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]] = None*)

Filter *Colour* and also optionally overall Python warnings.

The possible values for all the actions, i.e. each argument, are as follows:

- *None* (No action is taken)
- *True* (*ignore*)
- *False* (*default*)
- *error*
- *ignore*
- *always*
- *default*
- *module*
- *once*

Parameters

- **colour_runtime_warnings** (Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]]) – Whether to filter *Colour* runtime warnings according to the action value.
- **colour_usage_warnings** (Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]]) – Whether to filter *Colour* usage warnings according to the action value.
- **colour_warnings** (Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]]) – Whether to filter *Colour* warnings, this also filters *Colour* usage and runtime warnings according to the action value.
- **python_warnings** (Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]]) – Whether to filter *Python* warnings according to the action value.

Examples

Filtering *Colour* runtime warnings:

```
>>> filter_warnings(colour_runtime_warnings=True)
```

Filtering *Colour* usage warnings:

```
>>> filter_warnings(colour_usage_warnings=True)
```

Filtering *Colour* warnings:

```
>>> filter_warnings(colour_warnings=True)
```

Filtering all the *Colour* and also Python warnings:

```
>>> filter_warnings(python_warnings=True)
```

Enabling all the *Colour* and Python warnings:

```
>>> filter_warnings(*[False] * 4)
```

Enabling all the *Colour* and Python warnings using the *default* action:

```
>>> filter_warnings(*["default"] * 4)
```

Setting back the default state:

```
>>> filter_warnings(colour_runtime_warnings=True)
```

colour.utilities.suppress_warnings

colour.utilities.**suppress_warnings**(*colour_runtime_warnings: Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]] = None, colour_usage_warnings: Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]] = None, colour_warnings: Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]] = None, python_warnings: Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]] = None*) → *collections.abc.Generator*

Define a context manager filtering *Colour* and also optionally overall Python warnings.

The possible values for all the actions, i.e. each argument, are as follows:

- *None* (No action is taken)
- *True* (*ignore*)
- *False* (*default*)
- *error*
- *ignore*
- *always*
- *default*
- *module*
- *once*

Parameters

- **colour_runtime_warnings** (Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]]) – Whether to filter *Colour* runtime warnings according to the action value.
- **colour_usage_warnings** (Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]]) – Whether to filter *Colour* usage warnings according to the action value.
- **colour_warnings** (Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]]) – Whether to filter *Colour* warnings, this also filters *Colour* usage and runtime warnings according to the action value.
- **python_warnings** (Optional[Union[bool, Literal['default', 'error', 'ignore', 'always', 'module', 'once']]]) – Whether to filter *Python* warnings according to the action value.

Return type *collections.abc.Generator*

colour.utilities.numpy_print_options

colour.utilities.**numpy_print_options**(**args: Any, **kwargs: Any*) → *collections.abc.Generator*

Define a context manager implementing context changes to *Numpy* print behaviour.

Parameters

- **args** (Any) – Arguments.
- **kwargs** (Any) – Keywords arguments.

Return type *collections.abc.Generator*

Examples

```
>>> np.array([np.pi])
array([ 3.1415926...])
>>> with numpy_print_options(formatter={"float": "{:0.1f}".format}):
...     np.array([np.pi])
array([3.1])
```

colour.utilities.describe_environment

colour.utilities.**describe_environment**(*runtime_packages: bool = True, development_packages: bool = False, extras_packages: bool = False, print_environment: bool = True, **kwargs: Any*) → collections.defaultdict

Describe *Colour* running environment, i.e. interpreter, runtime and development packages.

Parameters

- **runtime_packages** (bool) – Whether to return the runtime packages versions.
- **development_packages** (bool) – Whether to return the development packages versions.
- **extras_packages** (bool) – Whether to return the extras packages versions.
- **print_environment** (bool) – Whether to print the environment.
- **padding** – {colour.utilities.message_box()}, Padding on each side of the message.

- **print_callable** – {colour.utilities.message_box()}, Callable used to print the message box.
- **width** – {colour.utilities.message_box()}, Message box width.
- **kwargs** (Any) –

Returns Environment.

Return type collections.defaultdict

Examples

```
>>> environment = describe_environment(width=75)
===========================================================================
*                                                                         *
*   Interpreter :                                                         *
*       python : 3.8.6 (default, Nov 20 2020, 18:29:40)                   *
*                [Clang 12.0.0 (clang-1200.0.32.27)]                      *
*                                                                         *
*   colour-science.org :                                                  *
*       colour : v0.3.16-3-gd8bac475                                      *
*                                                                         *
*   Runtime :                                                             *
*       imageio : 2.9.0                                                   *
*       matplotlib : 3.3.3                                                *
*       networkx : 2.5                                                    *
*       numpy : 1.19.4                                                    *
*       pandas : 0.25.3                                                   *
*       pygraphviz : 1.6                                                  *
*       scipy : 1.5.4                                                     *
*       tqdm : 4.54.0                                                     *
*                                                                         *
===========================================================================
>>> environment = describe_environment(True, True, True, width=75)
...
===========================================================================
*                                                                         *
*   Interpreter :                                                         *
*       python : 3.8.6 (default, Nov 20 2020, 18:29:40)                   *
*                [Clang 12.0.0 (clang-1200.0.32.27)]                      *
*                                                                         *
*   colour-science.org :                                                  *
*       colour : v0.3.16-3-gd8bac475                                      *
*                                                                         *
*   Runtime :                                                             *
*       imageio : 2.9.0                                                   *
*       matplotlib : 3.3.3                                                *
*       networkx : 2.5                                                    *
*       numpy : 1.19.4                                                    *
*       pandas : 0.25.3                                                   *
*       pygraphviz : 1.6                                                  *
*       scipy : 1.5.4                                                     *
*       tqdm : 4.54.0                                                     *
*                                                                         *
*   Development :                                                         *
*       biblib-simple : 0.1.1                                             *
*       coverage : 5.3                                                    *
```

(continues on next page)

(continued from previous page)

```
*       coveralls : 2.2.0                                                     *
*       flake8 : 3.8.4                                                        *
*       invoke : 1.4.1                                                        *
*       jupyter : 1.0.0                                                       *
*       mock : 4.0.2                                                          *
*       nose : 1.3.7                                                          *
*       pre-commit : 2.1.1                                                    *
*       pytest : 6.1.2                                                        *
*       restructuredtext-lint : 1.3.2                                         *
*       sphinx : 3.1.2                                                        *
*       sphinx_rtd_theme : 0.5.0                                              *
*       sphinxcontrib-bibtex : 1.0.0                                          *
*       toml : 0.10.2                                                         *
*       twine : 3.2.0                                                         *
*       yapf : 0.23.0                                                         *
*                                                                             *
*   Extras :                                                                  *
*       ipywidgets : 7.5.1                                                    *
*       notebook : 6.1.5                                                      *
*                                                                             *
===============================================================================
```

colour.utilities.multiline_str

colour.utilities.**multiline_str**(*object_: Any*, *attributes: List[dict]*, *header_underline: str = '='*, *section_underline: str = '-'*, *separator: str = ' : '*) → str

Return a formatted string representation of the given object.

Parameters

- **object** – Object to format.
- **attributes** (List[dict]) – Attributes to format.
- **header_underline** (str) – Underline character to use for a header.
- **section_underline** (str) – Underline character to use for a section.
- **separator** (str) – Separator to use when formatting the attributes and their values.
- **object_** (Any) –

Returns Formatted string representation.

Return type str

Examples

```
>>> class Data:
...     def __init__(self, a: str, b: int, c: list):
...         self._a = a
...         self._b = b
...         self._c = c
...
...     def __str__(self) -> str:
...         return multiline_str(
...             self,
```

(continues on next page)

(continued from previous page)

```
...             [
...                 {
...                     "formatter": lambda x: (
...                         f"Object - {self.__class__.__name__}"
...                     ),
...                     "header": True,
...                 },
...                 {"line_break": True},
...                 {"label": "Data", "section": True},
...                 {"line_break": True},
...                 {"label": "String", "section": True},
...                 {"name": "_a", "label": 'String "a"'},
...                 {"line_break": True},
...                 {"label": "Integer", "section": True},
...                 {"name": "_b", "label": 'Integer "b"'},
...                 {"line_break": True},
...                 {"label": "List", "section": True},
...                 {
...                     "name": "_c",
...                     "label": 'List "c"',
...                     "formatter": lambda x: "; ".join(x),
...                 },
...             ],
...         )
...
>>> print(Data("Foo", 1, ["John", "Doe"]))
Object - Data
=============

Data
----

String
------
String "a"  : Foo

Integer
-------
Integer "b" : 1

List
----
List "c"    : John; Doe
```

colour.utilities.multiline_repr

colour.utilities.**multiline_repr**(*object_: Any*, *attributes: List[dict]*, *reduce_array_representation: bool = True*) → str

Return an (almost) evaluable string representation of the given object.

Parameters

- **object** – Object to format.
- **attributes** (List[dict]) – Attributes to format.

- **reduce_array_representation** (bool) – Whether to remove the *Numpy array(* and *)* affixes.
- **object_** (Any) –

Returns (Almost) evaluable string representation.

Return type class`str`

Examples

```
>>> class Data:
...     def __init__(self, a: str, b: int, c: list):
...         self._a = a
...         self._b = b
...         self._c = c
...
...     def __repr__(self) -> str:
...         return multiline_repr(
...             self,
...             [
...                 {"name": "_a"},
...                 {"name": "_b"},
...                 {
...                     "name": "_c",
...                     "formatter": lambda x: repr(x)
...                     .replace("[", "(")
...                     .replace("]", ")"),
...                 },
...             ],
...         )
...
>>> Data("Foo", 1, ["John", "Doe"])
Data('Foo',
     1,
     ('John', 'Doe'))
```

Ancillary Objects

colour.utilities

ColourWarning	Define the base class of *Colour* warnings.
ColourUsageWarning	Define the base class of *Colour* usage warnings.
ColourRuntimeWarning	Define the base class of *Colour* runtime warnings.

colour.utilities.ColourWarning

class colour.utilities.**ColourWarning**

Bases: Warning

Define the base class of *Colour* warnings.

It is a subclass of the Warning class.

__weakref__

list of weak references to the object (if defined)

colour.utilities.ColourUsageWarning

class colour.utilities.**ColourUsageWarning**

Bases: Warning

Define the base class of *Colour* usage warnings.

It is a subclass of the colour.utilities.ColourWarning class.

__weakref__

list of weak references to the object (if defined)

colour.utilities.ColourRuntimeWarning

class colour.utilities.**ColourRuntimeWarning**

Bases: Warning

Define the base class of *Colour* runtime warnings.

It is a subclass of the colour.utilities.ColourWarning class.

__weakref__

list of weak references to the object (if defined)

Colour Volume

Optimal Colour Stimuli - MacAdam Limits

colour

is_within_macadam_limits(xyY[, illuminant, ...])	Return whether given *CIE xyY* colourspace array is within MacAdam limits of given illuminant.
OPTIMAL_COLOUR_STIMULI_ILLUMINANTS	Illuminants *Optimal Colour Stimuli*.

colour.is_within_macadam_limits

colour.**is_within_macadam_limits**(*xyY: ArrayLike, illuminant: Union[Literal['A', 'C', 'D65'], str] = 'D65', tolerance: float = 100 * EPSILON*) → NDArrayFloat

Return whether given *CIE xyY* colourspace array is within MacAdam limits of given illuminant.

Parameters

- **xyY** (ArrayLike) – *CIE xyY* colourspace array.
- **illuminant** (Union[Literal['A', 'C', 'D65'], str]) – Illuminant name.
- **tolerance** (float) – Tolerance allowed in the inside-triangle check.

Returns Whether given *CIE xyY* colourspace array is within MacAdam limits.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

Examples

```
>>> is_within_macadam_limits(np.array([0.3205, 0.4131, 0.51]), "A")
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.51], [0.0005, 0.0031, 0.001]])
>>> is_within_macadam_limits(a, "A")
array([ True, False], dtype=bool)
```

colour.OPTIMAL_COLOUR_STIMULI_ILLUMINANTS

colour.**OPTIMAL_COLOUR_STIMULI_ILLUMINANTS = CanonicalMapping({'A': ..., 'C': ..., 'D65': ...})**

Illuminants *Optimal Colour Stimuli*.

References

[Wikipedia04b]

Mesh Volume

colour

is_within_mesh_volume(points, mesh[, tolerance])	Return whether given points are within given mesh volume using Delaunay triangulation.

colour.is_within_mesh_volume

colour.**is_within_mesh_volume**(*points: ArrayLike, mesh: ArrayLike, tolerance: float = 100 * EPSILON*) → NDArrayFloat

Return whether given points are within given mesh volume using Delaunay triangulation.

Parameters

- **points** (ArrayLike) – Points to check if they are within mesh volume.
- **mesh** (ArrayLike) – Points of the volume used to generate the Delaunay triangulation.
- **tolerance** (float) – Tolerance allowed in the inside-triangle check.

Returns Whether given points are within given mesh volume.

Return type numpy.ndarray

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])	Return whether given *CIE XYZ* tristimulus values are within Pointer's Gamut volume.

colour.is_within_pointer_gamut

`colour.`**`is_within_pointer_gamut`**(*XYZ: ArrayLike*, *tolerance: float = 100 * EPSILON*) → NDArrayFloat

Return whether given *CIE XYZ* tristimulus values are within Pointer's Gamut volume.

Parameters

- **XYZ** (`ArrayLike`) – *CIE XYZ* tristimulus values.
- **tolerance** (`float`) – Tolerance allowed in the inside-triangle check.

Returns Wether given *CIE XYZ* tristimulus values are within Pointer's Gamut volume.

Return type `numpy.ndarray`

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> is_within_pointer_gamut(np.array([0.3205, 0.4131, 0.5100]))
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.5100], [0.0005, 0.0031, 0.0010]])
>>> is_within_pointer_gamut(a)
array([ True, False], dtype=bool)
```

RGB Volume

colour

RGB_colourspace_limits(colourspace)	Compute given *RGB* colourspace volume limits in *CIE L*a*b** colourspace.
RGB_colourspace_pointer_gamut_coverage_MonteCarlo(...)	Return given *RGB* colourspace percentage coverage of Pointer's Gamut volume using *Monte Carlo* method.
RGB_colourspace_visible_spectrum_coverage_MonteCarlo(...)	Return given *RGB* colourspace percentage coverage of visible spectrum volume using *Monte Carlo* method.
RGB_colourspace_volume_MonteCarlo(colourspace)	Perform given *RGB* colourspace volume computation using *Monte Carlo* method and multiprocessing.
RGB_colourspace_volume_coverage_MonteCarlo(...)	Return given *RGB* colourspace percentage coverage of an arbitrary volume.

colour.RGB_colourspace_limits

colour.**RGB_colourspace_limits**(*colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace*) → NDArrayFloat

Compute given *RGB* colourspace volume limits in *CIE L*a*b** colourspace.

Parameters colourspace (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace to compute the volume of.

Returns *RGB* colourspace volume limits.

Return type numpy.ndarray

Notes

The limits are computed for the given *RGB* colourspace illuminant. This is important to account for, if the intent is to compare various *RGB* colourspaces together. In this instance, they must be chromatically adapted to the same illuminant before-hand. See colour.RGB_Colourspace.chromatically_adapt() method for more information.

Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB as sRGB
>>> RGB_colourspace_limits(sRGB)
array([[   0.        ...,  100.        ...],
       [ -86.182855 ...,   98.2563272...],
       [-107.8503557...,   94.4894974...]])
```

colour.RGB_colourspace_pointer_gamut_coverage_MonteCarlo

colour.**RGB_colourspace_pointer_gamut_coverage_MonteCarlo**(*colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace, samples: int = 1000000, random_generator: Callable = random_triplet_generator, random_state: numpy.random.mtrand.RandomState | None = None*) → float

Return given *RGB* colourspace percentage coverage of Pointer's Gamut volume using *Monte Carlo* method.

Parameters

- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace to compute the *Pointer's Gamut* coverage percentage.
- **samples** (int) – Samples count.
- **random_generator** (Callable) – Random triplet generator providing the random samples.
- **random_state** (numpy.random.mtrand.RandomState | None) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns Percentage coverage of *Pointer's Gamut* volume.

Return type float

Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_pointer_gamut_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng
... )
81...
```

colour.RGB_colourspace_visible_spectrum_coverage_MonteCarlo

colour.**RGB_colourspace_visible_spectrum_coverage_MonteCarlo**(*colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspac samples: int = 1000000, random_generator: Callable = random_triplet_generator, random_state: numpy.random.mtrand.RandomState | None = None*) → float

Return given *RGB* colourspace percentage coverage of visible spectrum volume using *Monte Carlo* method.

Parameters

- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace to compute the visible spectrum coverage percentage.
- **samples** (int) – Samples count.

- **random_generator** (`Callable`) – Random triplet generator providing the random samples.
- **random_state** (`numpy.random.mtrand.RandomState | None`) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns Percentage coverage of visible spectrum volume.

Return type `float`

Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_visible_spectrum_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng
... )
46...
```

colour.RGB_colourspace_volume_MonteCarlo

`colour.`**`RGB_colourspace_volume_MonteCarlo`**(*colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace, samples: int = 1000000, limits: ArrayLike = np.array([[0, 100], [- 150, 150], [- 150, 150]]), illuminant_Lab: ArrayLike = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65'], chromatic_adaptation_transform: Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]] = 'CAT02', random_generator: Callable = random_triplet_generator, random_state: numpy.random.mtrand.RandomState | None = None*) → float

Perform given *RGB* colourspace volume computation using *Monte Carlo* method and multiprocessing.

Parameters

- **of.** (`colourspace RGB colourspace to compute the volume`) –
- **count.** (`samples Samples`) –
- **volume.** (`limits CIE L*a*b* colourspace`) –
- **coordinates.** (`illuminant_Lab CIE L*a*b* colourspace illuminant chromaticity`) –
- **method.** (`chromatic_adaptation_transform Chromatic adaptation`) –
- **the** (`random_generator Random triplet generator providing the random samples within`) – *CIE L*a*b** colourspace volume.
- **random** (`random_state Mersenne Twister pseudo-random number generator to use in the`) – number generator.
- **colourspace** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace`) –
- **samples** (`int`) –
- **limits** (`ArrayLike`) –

- **illuminant_Lab** (ArrayLike) –
- **chromatic_adaptation_transform** (Optional[Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]]) –
- **random_generator** (Callable) –
- **random_state** (numpy.random.mtrand.RandomState | None) –

Returns *RGB* colourspace volume.

Return type float

Notes

- The doctest is assuming that np.random.RandomState() definition will return the same sequence no matter which *OS* or *Python* version is used. There is however no formal promise about the *prng* sequence reproducibility of either *Python* or *Numpy* implementations: Laurent. (2012). Reproducibility of python pseudo-random numbers across systems and versions? Retrieved January 20, 2015, from http://stackoverflow.com/questions/8786084/reproducibility-of-python-pseudo-random-numbers-across-systems-and-versions

Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB as sRGB
>>> from colour.utilities import disable_multiprocessing
>>> prng = np.random.RandomState(2)
>>> with disable_multiprocessing():
...     RGB_colourspace_volume_MonteCarlo(sRGB, 10e3, random_state=prng)
...
...
8...
```

colour.RGB_colourspace_volume_coverage_MonteCarlo

colour.**RGB_colourspace_volume_coverage_MonteCarlo**(*colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace, coverage_sampler: Callable, samples: int = 1000000, random_generator: Callable = random_triplet_generator, random_state: numpy.random.mtrand.RandomState | None = None*) → float

Return given *RGB* colourspace percentage coverage of an arbitrary volume.

Parameters

- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace to compute the volume coverage percentage.
- **coverage_sampler** (Callable) – Python object responsible for checking the volume coverage.
- **samples** (int) – Samples count.
- **random_generator** (Callable) – Random triplet generator providing the random samples.

- **random_state** (numpy.random.mtrand.RandomState | None) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns Percentage coverage of volume.

Return type float

Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_volume_coverage_MonteCarlo(
...     sRGB, is_within_pointer_gamut, 10e3, random_state=prng
... )
...
81...
```

Ro?sch-MacAdam Colour solid - Visible Spectrum

colour

is_within_visible_spectrum(XYZ[, cmfs, ...])	Return whether given *CIE XYZ* tristimulus values are within the visible spectrum volume, i.e. *Ro?sch-MacAdam* colour solid, for given colour matching functions and illuminant.

colour.is_within_visible_spectrum

colour.**is_within_visible_spectrum**(*XYZ: ArrayLike, cmfs:* colour.colorimetry.spectrum.MultiSpectralDistributions | *None* = *None, illuminant:* colour.colorimetry.spectrum.SpectralDistribution | *None* = *None, tolerance: float* = *100 * EPSILON, **kwargs:* Any) → NDArrayFloat

Return whether given *CIE XYZ* tristimulus values are within the visible spectrum volume, i.e. *Ro?sch-MacAdam* colour solid, for given colour matching functions and illuminant.

Parameters

- **XYZ** (ArrayLike) – *CIE XYZ* tristimulus values.
- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **tolerance** (float) – Tolerance allowed in the inside-triangle check.
- **kwargs** (Any) – {colour.msds_to_XYZ()}, See the documentation of the previously listed definition.

Returns Are *CIE XYZ* tristimulus values within the visible spectrum volume, i.e. *Ro?sch-MacAdam* colour solid.

Return type numpy.ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> is_within_visible_spectrum(np.array([0.3205, 0.4131, 0.51]))
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.51], [-0.0005, 0.0031, 0.001]])
>>> is_within_visible_spectrum(a)
array([ True, False], dtype=bool)
```

Ancillary Objects

colour.volume

generate_pulse_waves(bins[, pulse_order, ...])	Generate the pulse waves of given number of bins necessary to totally stimulate the colour matching functions and produce the *Ro?sch-MacAdam* colour solid.
XYZ_outer_surface([cmfs, illuminant, ...])	Generate the *Ro?sch-MacAdam* colour solid, i.e. *CIE XYZ* colourspace outer surface, for given colour matching functions using multi-spectral conversion of pulse waves to *CIE XYZ* tristimulus values.
solid_RoschMacAdam([cmfs, illuminant, ...])	Generate the *Ro?sch-MacAdam* colour solid, i.e. *CIE XYZ* colourspace outer surface, for given colour matching functions using multi-spectral conversion of pulse waves to *CIE XYZ* tristimulus values.

colour.volume.generate_pulse_waves

colour.volume.**generate_pulse_waves**(*bins: int, pulse_order: Union[Literal['Bins', 'Pulse Wave Width'], str] = 'Bins', filter_jagged_pulses: bool = False*) → NDArrayFloat

Generate the pulse waves of given number of bins necessary to totally stimulate the colour matching functions and produce the *Ro?sch-MacAdam* colour solid.

Assuming 5 bins, a first set of SPDs would be as follows:

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

The second one:

```
1 1 0 0 0
0 1 1 0 0
0 0 1 1 0
0 0 0 1 1
1 0 0 0 1
```

The third:

1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1

Etc. . .

Parameters

- **bins** (int) – Number of bins of the pulse waves.
- **pulse_order** (Union[Literal['Bins', 'Pulse Wave Width'], str]) – Method for ordering the pulse waves. *Bins* is the default order, with *Pulse Wave Width* ordering, instead of iterating over the pulse wave widths first, iteration occurs over the bins, producing blocks of pulse waves with increasing width.
- **filter_jagged_pulses** (bool) – Whether to filter jagged pulses. When pulse_order is set to *Pulse Wave Width*, the pulses are ordered by increasing width. Because of the discrete nature of the underlying signal, the resulting pulses will be jagged. For example assuming 5 bins, the center block with the two extreme values added would be as follows:

```
0 0 0 0 0
0 0 1 0 0
0 0 1 1 0 <--
0 1 1 1 0
0 1 1 1 1 <--
1 1 1 1 1
```

  Setting the filter_jagged_pulses parameter to *True* will result in the removal of the two marked pulse waves above thus avoiding jagged lines when plotting and having to resort to excessive bins values.

Returns Pulse waves.

Return type numpy.ndarray

References

[Lin15], [Man18], [MartinezVerduPC+07]

Examples

```
>>> generate_pulse_waves(5)
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.],
       [ 1.,  1.,  0.,  0.,  0.],
       [ 0.,  1.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  1.,  0.],
       [ 0.,  0.,  0.,  1.,  1.],
       [ 1.,  0.,  0.,  0.,  1.],
       [ 1.,  1.,  1.,  0.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  0.,  1.,  1.,  1.],
       [ 1.,  0.,  0.,  1.,  1.],
       [ 1.,  1.,  0.,  0.,  1.],
       [ 1.,  1.,  1.,  1.,  0.],
```

(continues on next page)

(continued from previous page)

```
       [ 0.,  1.,  1.,  1.,  1.],
       [ 1.,  0.,  1.,  1.,  1.],
       [ 1.,  1.,  0.,  1.,  1.],
       [ 1.,  1.,  1.,  0.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])
>>> generate_pulse_waves(5, "Pulse Wave Width")
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  0.,  0.,  1.],
       [ 1.,  1.,  1.,  0.,  1.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  1.,  1.,  0.,  0.],
       [ 1.,  1.,  1.,  0.,  0.],
       [ 1.,  1.,  1.,  1.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  1.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  1.,  1.,  1.,  1.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  1.,  1.,  1.],
       [ 1.,  0.,  1.,  1.,  1.],
       [ 0.,  0.,  0.,  0.,  1.],
       [ 1.,  0.,  0.,  0.,  1.],
       [ 1.,  0.,  0.,  1.,  1.],
       [ 1.,  1.,  0.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])
>>> generate_pulse_waves(5, "Pulse Wave Width", True)
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  1.,  0.,  0.,  1.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  1.,  1.,  1.],
       [ 0.,  0.,  0.,  0.,  1.],
       [ 1.,  0.,  0.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])
```

colour.volume.XYZ_outer_surface

colour.volume.**XYZ_outer_surface**(*cmfs:* colour.colorimetry.spectrum.MultiSpectralDistributions | *None = None, illuminant:* colour.colorimetry.spectrum.SpectralDistribution | *None = None, point_order: Union[Literal['Bins', 'Pulse Wave Width'], str] = 'Bins', filter_jagged_points: bool = False, **kwargs: Any*) → NDArrayFloat

Generate the *Ro?sch-MacAdam* colour solid, i.e. *CIE XYZ* colourspace outer surface, for given colour matching functions using multi-spectral conversion of pulse waves to *CIE XYZ* tristimulus values.

Parameters

- **cmfs** (`colour.colorimetry.spectrum.MultiSpectralDistributions | None`) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree*

Standard Observer.

- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **point_order** (Union[Literal['Bins', 'Pulse Wave Width'], str]) – Method for ordering the underlying pulse waves used to generate the *Ro?sch-MacAdam* colour solid. *Bins* is the default order, with *Pulse Wave Width* ordering, instead of iterating over the pulse wave widths first, iteration occurs over the bins, producing blocks of pulse waves with increasing width.
- **filter_jagged_points** (bool) – Whether to filter the underlying jagged pulses. When point_order is set to *Pulse Wave Width*, the pulses are ordered by increasing width. Because of the discrete nature of the underlying signal, the resulting pulses will be jagged. For example assuming 5 bins, the center block with the two extreme values added would be as follows:

```
0 0 0 0 0
0 0 1 0 0
0 0 1 1 0 <--
0 1 1 1 0
0 1 1 1 1 <--
1 1 1 1 1
```

  Setting the filter_jagged_points parameter to *True* will result in the removal of the two marked pulse waves above thus avoiding jagged lines when plotting and having to resort to excessive bins values.

- **kwargs** (Any) – {colour.msds_to_XYZ()}, See the documentation of the previously listed definition.

Returns *Ro?sch-MacAdam* colour solid, *CIE XYZ* outer surface tristimulus values.

Return type numpy.ndarray

References

[Lin15], [Man18], [MartinezVerduPC+07]

Examples

```
>>> from colour import MSDS_CMFS, SPECTRAL_SHAPE_DEFAULT
>>> shape = SpectralShape(
...     SPECTRAL_SHAPE_DEFAULT.start, SPECTRAL_SHAPE_DEFAULT.end, 84
... )
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> XYZ_outer_surface(cmfs.copy().align(shape))
array([[  0.0000000...e+00,   0.0000000...e+00,   0.0000000...e+00],
       [  9.6361381...e-05,   2.9056776...e-06,   4.4961226...e-04],
       [  2.5910529...e-01,   2.1031298...e-02,   1.3207468...e+00],
       [  1.0561021...e-01,   6.2038243...e-01,   3.5423571...e-02],
       [  7.2647980...e-01,   3.5460869...e-01,   2.1005149...e-04],
       [  1.0971874...e-02,   3.9635453...e-03,   0.0000000...e+00],
       [  3.0792572...e-05,   1.1119762...e-05,   0.0000000...e+00],
       [  2.5920165...e-01,   2.1034203...e-02,   1.3211965...e+00],
       [  3.6471551...e-01,   6.4141373...e-01,   1.3561704...e+00],
       [  8.3209002...e-01,   9.7499113...e-01,   3.5633622...e-02],
       [  7.3745167...e-01,   3.5857224...e-01,   2.1005149...e-04],
       [  1.1002667...e-02,   3.9746651...e-03,   0.0000000...e+00],
```

(continues on next page)

(continued from previous page)

```
        [  1.2715395...e-04,   1.4025439...e-05,   4.4961226...e-04],
        [  3.6481187...e-01,   6.4141663...e-01,   1.3566200...e+00],
        [  1.0911953...e+00,   9.9602242...e-01,   1.3563805...e+00],
        [  8.4306189...e-01,   9.7895467...e-01,   3.5633622...e-02],
        [  7.3748247...e-01,   3.5858336...e-01,   2.1005149...e-04],
        [  1.1099028...e-02,   3.9775708...e-03,   4.4961226...e-04],
        [  2.5923244...e-01,   2.1045323...e-02,   1.3211965...e+00],
        [  1.0912916...e+00,   9.9602533...e-01,   1.3568301...e+00],
        [  1.1021671...e+00,   9.9998597...e-01,   1.3563805...e+00],
        [  8.4309268...e-01,   9.7896579...e-01,   3.5633622...e-02],
        [  7.3757883...e-01,   3.5858626...e-01,   6.5966375...e-04],
        [  2.7020432...e-01,   2.5008868...e-02,   1.3211965...e+00],
        [  3.6484266...e-01,   6.4142775...e-01,   1.3566200...e+00],
        [  1.1022635...e+00,   9.9998888...e-01,   1.3568301...e+00],
        [  1.1021979...e+00,   9.9999709...e-01,   1.3563805...e+00],
        [  8.4318905...e-01,   9.7896870...e-01,   3.6083235...e-02],
        [  9.9668412...e-01,   3.7961756...e-01,   1.3214065...e+00],
        [  3.7581454...e-01,   6.4539130...e-01,   1.3566200...e+00],
        [  1.0913224...e+00,   9.9603645...e-01,   1.3568301...e+00],
        [  1.1022943...e+00,   1.0000000...e+00,   1.3568301...e+00]])
```

colour.volume.solid_RoschMacAdam

colour.volume.**solid_RoschMacAdam**(*cmfs: colour.colorimetry.spectrum.MultiSpectralDistributions | None = None, illuminant: colour.colorimetry.spectrum.SpectralDistribution | None = None, point_order: Union[Literal['Bins', 'Pulse Wave Width'], str] = 'Bins', filter_jagged_points: bool = False, **kwargs: Any*) → NDArrayFloat

Generate the *Ro?sch-MacAdam* colour solid, i.e. *CIE XYZ* colourspace outer surface, for given colour matching functions using multi-spectral conversion of pulse waves to *CIE XYZ* tristimulus values.

Parameters

- **cmfs** (colour.colorimetry.spectrum.MultiSpectralDistributions | None) – Standard observer colour matching functions, default to the *CIE 1931 2 Degree Standard Observer*.
- **illuminant** (colour.colorimetry.spectrum.SpectralDistribution | None) – Illuminant spectral distribution, default to *CIE Illuminant E*.
- **point_order** (Union[Literal['Bins', 'Pulse Wave Width'], str]) – Method for ordering the underlying pulse waves used to generate the *Ro?sch-MacAdam* colour solid. *Bins* is the default order, with *Pulse Wave Width* ordering, instead of iterating over the pulse wave widths first, iteration occurs over the bins, producing blocks of pulse waves with increasing width.
- **filter_jagged_points** (bool) – Whether to filter the underlying jagged pulses. When `point_order` is set to *Pulse Wave Width*, the pulses are ordered by increasing width. Because of the discrete nature of the underlying signal, the resulting pulses will be jagged. For example assuming 5 bins, the center block with the two extreme values added would be as follows:

```
0 0 0 0 0
0 0 1 0 0
0 0 1 1 0 <--
0 1 1 1 0
```

(continues on next page)

(continued from previous page)

```
0 1 1 1 1 <--
1 1 1 1 1
```

Setting the `filter_jagged_points` parameter to *True* will result in the removal of the two marked pulse waves above thus avoiding jagged lines when plotting and having to resort to excessive `bins` values.

- **kwargs** (Any) – {`colour.msds_to_XYZ()`}, See the documentation of the previously listed definition.

Returns *Ro?sch-MacAdam* colour solid, *CIE XYZ* outer surface tristimulus values.

Return type numpy.ndarray

References

[Lin15], [Man18], [MartinezVerduPC+07]

Examples

```
>>> from colour import MSDS_CMFS, SPECTRAL_SHAPE_DEFAULT
>>> shape = SpectralShape(
...     SPECTRAL_SHAPE_DEFAULT.start, SPECTRAL_SHAPE_DEFAULT.end, 84
... )
>>> cmfs = MSDS_CMFS["CIE 1931 2 Degree Standard Observer"]
>>> XYZ_outer_surface(cmfs.copy().align(shape))
array([[  0.0000000...e+00,   0.0000000...e+00,   0.0000000...e+00],
       [  9.6361381...e-05,   2.9056776...e-06,   4.4961226...e-04],
       [  2.5910529...e-01,   2.1031298...e-02,   1.3207468...e+00],
       [  1.0561021...e-01,   6.2038243...e-01,   3.5423571...e-02],
       [  7.2647980...e-01,   3.5460869...e-01,   2.1005149...e-04],
       [  1.0971874...e-02,   3.9635453...e-03,   0.0000000...e+00],
       [  3.0792572...e-05,   1.1119762...e-05,   0.0000000...e+00],
       [  2.5920165...e-01,   2.1034203...e-02,   1.3211965...e+00],
       [  3.6471551...e-01,   6.4141373...e-01,   1.3561704...e+00],
       [  8.3209002...e-01,   9.7499113...e-01,   3.5633622...e-02],
       [  7.3745167...e-01,   3.5857224...e-01,   2.1005149...e-04],
       [  1.1002667...e-02,   3.9746651...e-03,   0.0000000...e+00],
       [  1.2715395...e-04,   1.4025439...e-05,   4.4961226...e-04],
       [  3.6481187...e-01,   6.4141663...e-01,   1.3566200...e+00],
       [  1.0911953...e+00,   9.9602242...e-01,   1.3563805...e+00],
       [  8.4306189...e-01,   9.7895467...e-01,   3.5633622...e-02],
       [  7.3748247...e-01,   3.5858336...e-01,   2.1005149...e-04],
       [  1.1099028...e-02,   3.9775708...e-03,   4.4961226...e-04],
       [  2.5923244...e-01,   2.1045323...e-02,   1.3211965...e+00],
       [  1.0912916...e+00,   9.9602533...e-01,   1.3568301...e+00],
       [  1.1021671...e+00,   9.9998597...e-01,   1.3563805...e+00],
       [  8.4309268...e-01,   9.7896579...e-01,   3.5633622...e-02],
       [  7.3757883...e-01,   3.5858626...e-01,   6.5966375...e-04],
       [  2.7020432...e-01,   2.5008868...e-02,   1.3211965...e+00],
       [  3.6484266...e-01,   6.4142775...e-01,   1.3566200...e+00],
       [  1.1022635...e+00,   9.9998888...e-01,   1.3568301...e+00],
       [  1.1021979...e+00,   9.9999709...e-01,   1.3563805...e+00],
       [  8.4318905...e-01,   9.7896870...e-01,   3.6083235...e-02],
       [  9.9668412...e-01,   3.7961756...e-01,   1.3214065...e+00],
       [  3.7581454...e-01,   6.4539130...e-01,   1.3566200...e+00],
```

(continues on next page)

(continued from previous page)

```
       [  1.0913224...e+00,   9.9603645...e-01,   1.3568301...e+00],
       [  1.1022943...e+00,   1.0000000...e+00,   1.3568301...e+00]])
```

4.1.2 Indices and tables

- genindex
- search

CHAPTER

FIVE

6 SEE ALSO

5.1 6.1 Software

Python

- ColorPy by Kness, M.
- Colorspacious by Smith, N. J., et al.
- python-colormath by Taylor, G., et al.

Go

- go-colorful by Beyer, L., et al.

.NET

- Colourful by Pažourek, T., et al.

Julia

- Colors.jl by Holy, T., et al.

Matlab & Octave

- COLORLAB by Malo, J., et al.
- Psychtoolbox by Brainard, D., et al.
- The Munsell and Kubelka-Munk Toolbox by Centore, P.

CHAPTER

SIX

7 CODE OF CONDUCT

The *Code of Conduct*, adapted from the Contributor Covenant 1.4, is available on the Code of Conduct page.

CHAPTER

SEVEN

8 CONTACT & SOCIAL

The *Colour Developers* can be reached via different means:

- Email
- Facebook
- Github Discussions
- Gitter
- Twitter

CHAPTER

EIGHT

9 ABOUT

Colour by Colour Developers

https://github.com/colour-science/colour

BIBLIOGRAPHY

[APLR17] Mekides Assefa Abebe, Tania Pouli, Mohamed-Chaker Larabi, and Erik Reinhard. Perceptual Lightness Modeling for High-Dynamic-Range Imaging. *ACM Transactions on Applied Perception*, 15(1):1–19, July 2017. doi:10.1145/3086577.

[Bar99] Peter G. Barten. *Contrast Sensitivity of the Human Eye and Its Effects on Image Quality*. Number 1999. SPIE, December 1999. ISBN 978-0-8194-7849-8. doi:10.1117/3.353254.

[Bar03] Peter G. J. Barten. Formula for the contrast sensitivity of the human eye. In Yoichi Miyake and D. Rene Rasmussen, editors, *Proceedings of SPIE*, volume 5294, 231–238. December 2003. doi:10.1117/12.537476.

[BS10] S. Bianco and R. Schettini. Two new von Kries based chromatic adaptation transforms found by numerical optimization. *Color Research & Application*, 35(3):184–192, June 2010. doi:10.1002/col.20573.

[BWDS99] Barry A. Bodhaine, Norman B. Wood, Ellsworth G. Dutton, and James R. Slusser. On Rayleigh Optical Depth Calculations. *Journal of Atmospheric and Oceanic Technology*, 16(11):1854–1861, November 1999. doi:10.1175/1520-0426(1999)016{$<$}1854:ORODC{$>$}2.0.CO;2.

[Bor17] Tim Borer. Private Discussion with Mansencal, T. and Shaw, N. 2017.

[Boua] Paul Bourke. Intersection point of two line segments in 2 dimensions. http://paulbourke.net/geometry/pointlineplane/.

[Boub] Paul Bourke. Trilinear Interpolation. http://paulbourke.net/miscellaneous/interpolation/.

[Bre87] Edwin J. Breneman. Corresponding chromaticities for different states of adaptation to complex visual fields. *Journal of the Optical Society of America A*, 4(6):1115, June 1987. doi:10.1364/JOSAA.4.001115.

[BS08] Michael H. Brill and Sabine Susstrunk. Repairing gamut problems in CIECAM02: A progress report. *Color Research & Application*, 33(5):424–426, October 2008. 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. http://www.cis.rit.edu/mcsl/research/1931.php, 2009.

[BB09] Wilhelm Burger and Mark James Burge. *Principles of Digital Image Processing*. Springer London, London, 2009. ISBN 978-1-84800-194-7. doi:10.1007/978-1-84800-195-4.

[Cab] Ricardo Cabello. PlaneGeometry.js. https://github.com/mrdoob/three.js/blob/dev/src/geometries/PlaneGe

[CTS13] Renbo Cao, H Joel Trussell, and Renzo Shamey. Comparison of the performance of inverse transformation methods from OSA-UCS to CIEXYZ. *Journal of the Optical Society of America A*, 30(8):1508, August 2013. doi:10.1364/JOSAA.30.001508.

[CSH+18] E.C. Carter, J.D. Schanda, R. Hirschler, S. Jost, M.R. Luo, M. Melgosa, Y. Ohno, M.R. Pointer, D.C. Rich, F. Vienot, L. Whitehead, and J.H. Wold. CIE 015:2018 Colorimetry, 4th Edition. Technical Report, International Commission on Illumination, Vienna, October 2018. doi:10.25039/TR.015.2018.

[Cas14] Saullo Castro. Numpy: Fastest way of computing diagonal for each row of a 2d array. http://stackoverflow.com/questions/26511401/numpy-fastest-way-of-computing-diagonal-for-each-row-of-a-2d-array/26517247#26517247, 2014.

[Cen] Paul Centore. The Munsell and Kubelka-Munk Toolbox. http://www.munsellcolourscienceforpainters.com/MunsellAndKubelkaMunkToolbox/MunsellAndKubelka

[Cen12] Paul Centore. An open-source inversion algorithm for the Munsell renotation. *Color Research & Application*, 37(6):455–464, December 2012. doi:10.1002/col.20715.

[Cen14a] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - GeneralRoutines/CIELABtoApproxMunsellSpec.m. 2014.

[Cen14b] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/ChromDiagHueAngleToMunsellHue.m. 2014.

[Cen14c] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/FindHueOnRenotationOvoid.m. 2014.

[Cen14d] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MaxChromaForExtrapolatedRenotation.m. 2014.

[Cen14e] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellHueToASTMHue.m. 2014.

[Cen14f] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellHueToChromDiagHueAngle.m. 2014.

[Cen14g] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellToxyForIntegerMunsellValue.m. 2014.

[Cen14h] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellToxyY.m. 2014.

[Cen14i] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/xyYtoMunsell.m. 2014.

[Cen14j] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellSystemRoutines/BoundingRenotationHues.m. 2014.

[Cen14k] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellSystemRoutines/LinearVsRadialInterpOnRenotationOvoid.m. 2014.

[Cha15] Peter Chamberlain. LUT documentation (to create from another program). https://forum.blackmagicdesign.com/viewtopic.php?f=21&t=40284#p232952, 2015.

[CWCR04] Vien Cheung, Stephen Westland, David Connah, and Caterina Ripamonti. A comparative study of the characterisation of colour cameras by means of neural networks and polynomial transforms. *Coloration Technology*, 120(1):19–25, 2004. doi:10.1111/j.1478-4408.2004.tb00201.x.

[CB22] Sean Cooper and Harald Brendel. ARRI LogC4 Logarithmic Color Space SPECIFICATION. 2022.

[Cot] Russell Cottrell. The Russell RGB working color space.

[CKMW04] Matthew Cowan, Glenn Kennel, Thomas Maier, and Brad Walker. Contrast Sensitivity Experiment to Determine the Bit Depth for Digital Cinema. *SMPTE Motion Imaging Journal*, 113(9):281–292, September 2004. doi:10.5594/j11549.

[CLR+02] G. Cui, M. R. Luo, B. Rigg, G. Roesler, and K. Witt. Uniform colour spaces based on the DIN99 colour-difference formula. *Color Research & Application*, 27(4):282–290, 2002. doi:10.1002/col.10066.

[DFGM15] Maryam Mohammadzadeh Darrodi, Graham Finlayson, Teresa Goodman, and Michal Mackiewicz. Reference data set for camera spectral sensitivity estimation. *Journal of the Optical Society of America A*, 32(3):381, March 2015. doi:10.1364/JOSAA.32.000381.

[DFH+15] Aurelien David, Paul T. Fini, Kevin W. Houser, Yoshi Ohno, Michael P. Royer, Kevin A. G. Smet, Minchen Wei, and Lorne Whitehead. Development of the IES method for evaluating the color rendition of light sources. *Optics Express*, 23(12):15888, June 2015. doi:10.1364/OE.23.015888.

[DO10] Wendy Davis and Yoshiro Ohno. Color quality scale. *Optical Engineering*, 49(3):033602, March 2010. doi:10.1117/1.3360335.

[DFI+17] Scott Dyer, Alexander Forsythe, Jonathon Irons, Thomas Mansencal, and Miaoqi Zhu. RAW to ACES. 2017.

[EF98] Fritz Ebner and Mark D. Fairchild. Finding constant hue surfaces in color space. In Giordano B. Beretta and Reiner Eschbach, editors, *Proc. SPIE 3300, Color Imaging: Device-Independent Color, Color Hardcopy, and Graphic Arts III, (2 January 1998)*, 107–117. January 1998. doi:10.1117/12.298269.

[Erda] U. Murat Erdem. Fast Line Segment Intersection. http://www.mathworks.com/matlabcentral/fileexchange/fast-line-segment-intersection.

[Erdb] Turan Erdogan. How to Calculate Luminosity, Dominant Wavelength, and Excitation Purity.

[FW98] M. Fairchild and D. Wyble. Colorimetric Characterization of The Apple Studio Display (flat panel LCD). 1998.

[FC11] Mark D Fairchild and Ping-hsu Chen. Brightness, lightness, and specifying color in high-dynamic-range scenes and images. In Susan P. Farnand and Frans Gaykema, editors, *Proc. SPIE 7867, Image Quality and System Performance VIII*, 78670O. January 2011. doi:10.1117/12.872075.

[FH22] Mark D Fairchild and Luke Hellwig. Private Discussion with Mansencal, T. 2022.

[Fai] Mark D. Fairchild. Fairchild YSh.

[Fai91] Mark D. Fairchild. Formulation and testing of an incomplete-chromatic-adaptation model. *Color Research & Application*, 16(4):243–250, August 1991. doi:10.1002/col.5080160406.

[Fai96] Mark D. Fairchild. Refinement of the RLAB color space. *Color Research & Application*, 21(5):338–346, October 1996. 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*, pages 289–301. Wiley, 2 edition, 2004.

[Fai13a] Mark D. Fairchild. Chromatic Adaptation Models. In *Color Appearance Models*, pages 4179–4252. Wiley, 3 edition, 2013.

[Fai13b] Mark D. Fairchild. The Hunt Model. In *Color Appearance Models*, pages 5094–5556. Wiley, 3 edition, 2013.

[Fai13c] Mark D. Fairchild. The Nayatani et al. Model. In *Color Appearance Models*, pages 4810–5085. Wiley, 3 edition, 2013.

[Fai13d] Mark D. Fairchild. The RLAB Model. In *Color Appearance Models*, pages 5563–5824. Wiley, 3 edition, 2013.

[Fai13e] Mark D. Fairchild. ATD Model. In *Color Appearance Models*, pages 5852–5991. Wiley, 3 edition, 2013.

[Fai13f] Mark D. Fairchild. FAIRCHILD'S 1990 MODEL. In *Color Appearance Models*, pages 4418–4495. Wiley, 3 edition, 2013.

[Fai13g] Mark D. Fairchild. IPT Colourspace. In *Color Appearance Models*, pages 6197–6223. Wiley, 3 edition, 2013.

[Fai13h] Mark D. Fairchild. LLAB Model. In *Color Appearance Models*, pages 6025–6178. Wiley, 3 edition, 2013.

[Fai20] Mark D. Fairchild. Von Kries 2020: Evolution of degree of chromatic adaptation. *Color and Imaging Conference*, 28(1):252–257, November 2020. doi:10.2352/issn.2169-2629.2020.28.40.

[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.

[Fai85] Hugh S. Fairman. The calculation of weight factors for tristimulus integration. *Color Research & Application*, 10(4):199–203, 1985. doi:10.1002/col.5080100407.

[FBH97] Hugh S. Fairman, Michael H. Brill, and Henry Hemmendinger. How the CIE 1931 color-matching functions were derived from Wright-Guild data. *Color Research & Application*, 22(1):11–23, February 1997. doi:10.1002/(SICI)1520-6378(199702)22:1{$<$}11::AID-COL4{$>$}3.0.CO;2-7.

[FMH15] Graham D. Finlayson, Michal MacKiewicz, and Anya Hurlbert. Color Correction Using Root-Polynomial Regression. *IEEE Transactions on Image Processing*, 24(5):1460–1470, May 2015. doi:10.1109/TIP.2015.2405336.

[For18] Alex Forsythe. Private Discussion with Mansencal, T. 2018.

[Frohlich17] Jan Fröhlich. Encoding high dynamic range and wide color gamut imagery. 2017.

[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.

[GarciaHMC07] Pedro A. García, Rafael Huertas, Manuel Melgosa, and Guihua Cui. Measurement of the relationship between perceived and computed color differences. *Journal of the Optical Society of America A*, 24(7):1823, July 2007. doi:10.1364/JOSAA.24.001823.

[GMRS58] L. G. Glasser, A. H. McKinney, C. D. Reilly, and P. D. Schnelle. Cube-Root Color Coordinate System. *Journal of the Optical Society of America*, 48(10):736, October 1958. doi:10.1364/JOSA.48.000736.

[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. April 1995. doi:10.1117/12.206546.

[HF98] Radim Halir and Jan Flusser. Numerically Stable Direct Least Squares Fitting Of Ellipses. 1998. doi:10.1.1.1.7559.

[Han03] Allan Hanbury. A 3D-Polar Coordinate Colour Representation Well Adapted to Image Analysis. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, 804–811. Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[HF20] Luke Hellwig and Mark D. Fairchild. Using Gaussian Spectra to Derive a Hue-linear Color Space. *Journal of Perceptual Imaging*, 2020. doi:10.2352/J.Percept.Imaging.2020.3.2.020401.

[HF22] Luke Hellwig and Mark D. Fairchild. Brightness, lightness, colorfulness, and chroma in \textsc CIECAM02 and \textsc CAM16. *Color Research & Application*, pages col.22792, March 2022. doi:10.1002/col.22792.

[HSF22] Luke Hellwig, Dale Stolitzka, and Mark D. Fairchild. Extending CIECAM02 and CAM16 for the Helmholtz–Kohlrausch effect. *Color Research & Application*, pages col.22793, June 2022. doi:10.1002/col.22793.

[Hol] Joseph Holmes. Ekta Space PS 5.

[Hou15] Jim Houston. Private Discussion with Mansencal, T. 2015.

[HCM+15] Min Huang, Guihua Cui, Manuel Melgosa, Manuel Sánchez-Marañón, Changjun Li, M. Ronnier Luo, and Haoxue Liu. Power functions improving the performance of color-difference formulas. *Optical Society of America*, 23(1):597–610, 2015. doi:10.1364/OE.23.000597.

[HB95] Po-Chieh Hung and Roy S. Berns. Determination of constant Hue Loci for a CRT gamut and their predictions using color appearance spaces. *Color Research & Application*, 20(5):285–295, October 1995. doi:10.1002/col.5080200506.

[Hun04] R.W.G. Hunt. *The Reproduction of Colour*. John Wiley & Sons, Ltd, Chichester, UK, 6 edition, September 2004. ISBN 978-0-470-02427-0. doi:10.1002/0470024275.

[JH19] Wenzel Jakob and Johannes Hanika. A Low-Dimensional Function Space for Efficient Spectral Upsampling. *Computer Graphics Forum*, 38(2):147–155, May 2019. doi:10.1111/cgf.13626.

[JLGS13] Jun Jiang, Dengyu Liu, Jinwei Gu, and Sabine Susstrunk. What is the space of spectral sensitivity functions for digital color cameras? In *2013 IEEE Workshop on Applications of Computer Vision (WACV)*, 168–179. IEEE, January 2013. doi:10.1109/WACV.2013.6475015.

[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.

[KPK11] Paul Kienzle, Nikunj Patel, and James Krycka. Refl1d.numpyerrors - Refl1D v0.6.19 documentation. http://www.reflectometry.org/danse/docs/refl1d/_modules/refl1d/numpyerrors.html, 2011.

[KWK09] Mh Kim, T Weyrich, and J Kautz. Modeling Human Color Perception under Extended Luminance Levels. *ACM Transactions on Graphics*, 28(3):27:1–27:9, 2009. doi:10.1145/1531326.1531333.

[Kir06] Richard Kirk. Truelight Software Library 2.0. 2006.

[Kir19] Richard A. Kirk. Chromaticity coordinates for graphic arts based on CIE 2006 LMS with even spacing of Munsell colours. *Color and Imaging Conference*, 27(1):215–219, October 2019. doi:10.2352/issn.2169-2629.2019.27.38.

[Kon21] Ivan A. Konovalenko. proLab_param.m. 2021.

[KSNN21] Ivan A. Konovalenko, Anna A. Smagina, Dmitry P. Nikolaev, and Petr P. Nikolaev. ProLab: perceptually uniform projective colour coordinate system. *arXiv:2012.07653 [cs]*, January 2021. arXiv:2012.07653.

[Kry85] M Krystek. An algorithm to calculate correlated colour temperature. *Color Research & Application*, 10(1):38–40, 1985. doi:10.1002/col.5080100109.

[LLW+17] Changjun Li, Zhiqiang Li, Zhifeng Wang, Yang Xu, Ming Ronnier Luo, Guihua Cui, Manuel Melgosa, Michael H Brill, and Michael Pointer. Comprehensive color solutions: CAM16, CAT16, and CAM16-UCS. *Color Research & Application*, 42(6):703–718, December 2017. doi:10.1002/col.22131.

[LLRH02] Changjun Li, Ming Ronnier Luo, Bryan Rigg, and Robert W. G. Hunt. CMC 2000 chromatic adaptation transform: CMCCAT2000. *Color Research & Application*, 27(1):49–58, February 2002. doi:10.1002/col.10005.

[LPLMartinezverdu07] Changjun Li, Esther Perales, Ming Ronnier Luo, and Francisco Martinez-verdu. The Problem with CAT02 and Its Correction. 2007.

[Lin03a] Bruce Lindbloom. Delta E (CIE 1976). http://brucelindbloom.com/Eqn_DeltaE_CIE76.html, 2003.

[Lin03b] Bruce Lindbloom. XYZ to xyY. http://www.brucelindbloom.com/Eqn_XYZ_to_xyY.html, 2003.

[Lin07] Bruce Lindbloom. Spectral Power Distribution of a CIE D-Illuminant. http://www.brucelindbloom.com/Eqn_DIlluminant.html, 2007.

[Lin09a] Bruce Lindbloom. Chromatic Adaptation. http://brucelindbloom.com/Eqn_ChromAdapt.html, 2009.

[Lin09b] Bruce Lindbloom. Delta E (CMC). http://brucelindbloom.com/Eqn_DeltaE_CMC.html, 2009.

[Lin09c] Bruce Lindbloom. xyY to XYZ. http://www.brucelindbloom.com/Eqn_xyY_to_XYZ.html, 2009.

[Lin11] Bruce Lindbloom. Delta E (CIE 1994). http://brucelindbloom.com/Eqn_DeltaE_CIE94.html, 2011.

[Lin14] Bruce Lindbloom. RGB Working Space Information. http://www.brucelindbloom.com/WorkingSpaceInfo.ht 2014.

[Lin15] Bruce Lindbloom. About the Lab Gamut. http://www.brucelindbloom.com/LabGamutDisplayHelp.html, 2015.

[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.

[LCL06] M. Ronnier Luo, Guihua Cui, and Changjun Li. Uniform colour spaces based on CIECAM02 colour appearance model. *Color Research & Application*, 31(4):320–330, August 2006. doi:10.1002/col.20227.

[LR99] M. Ronnier Luo and Peter A. Rhodes. Corresponding-colour datasets. *Color Research & Application*, 24(4):295–296, August 1999. doi:10.1002/(SICI)1520-6378(199908)24:4{$<$}295::AID-COL10{$>$}3.0.CO;2-K.

[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. doi:10.1007/978-1-4419-6190-7.

[LLK96] Ming Ronnier Luo, Mei-Chun Lo, and Wen-Guey Kuo. The LLAB (l:c) colour model. *Color Research & Application*, 21(6):412–429, December 1996. 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.

[Mac35] David L. MacAdam. Maximum Visual Efficiency of Colored Materials. *Journal of the Optical Society of America*, 25(11):361–367, November 1935. doi:10.1364/JOSA.25.000361.

[Mac42] David L. Macadam. Visual Sensitivities to Color Differences in Daylight. *Journal of the Optical Society of America*, 32(5):28, 1942. doi:10.1364/JOSA.32.000247.

[MOF09] G.M. Machado, M.M. Oliveira, and L. Fernandes. A Physiologically-based Model for Simulation of Color Vision Deficiency. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1291–1298, November 2009. doi:10.1109/TVCG.2009.113.

[Mac10] Gustavo Mello Machado. A model for simulation of color vision deficiency and a color contrast enhancement technique for dichromats. 2010.

[MY19] Ian Mallett and Cem Yuksel. Spectral Primary Decomposition for Rendering with sRGB Reflectance. *Eurographics Symposium on Rendering - DL-only and Industry Track*, pages 7 pages, 2019. doi:10.2312/SR.20191216.

[MS03] Henrique Malvar and Gary Sullivan. YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range. 2003.

[Mana] Thomas Mansencal. Lookup.

[Manb] Thomas Mansencal. Structure.

[Man15] Thomas Mansencal. RED Colourspaces Derivation. https://www.colour-science.org/posts/red-colourspaces-derivation, 2015.

[Man18] Thomas Mansencal. How is the visible gamut bounded? https://stackoverflow.com/a/48396021/931625, 2018.

[Man19] Thomas Mansencal. Colour - Datasets. 2019. doi:10.5281/zenodo.3362520.

[Mel13] Manuel Melgosa. CIE / ISO new standard: CIEDE2000. 2013.

[MSHD15] Johannes Meng, Florian Simon, Johannes Hanika, and Carsten Dachsbacher. Physically Meaningful Rendering using Tristimulus Colours. *Computer Graphics Forum*, 34(4):31–40, July 2015. doi:10.1111/cgf.12676.

[Mil14] Scott Miller. A Perceptual EOTF for Extended Dynamic Range Imagery. 2014.

[MT11] Wojciech Mokrzycki and Maciej Tatol. Color difference Delta E - A survey. *Machine Graphics and Vision*, 20:383–411, April 2011.

[Mor03] Nathan Moroney. A Radial Sampling of the OSA Uniform Color Scales. *Color and Imaging Conference*, 2003(1):175–180, 2003.

[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.

[MorovivcL00] Ján Morovič and M. Ronnier Luo. Calculating medium and image gamut boundaries for gamut mapping. *Color Research and Application*, 25(6):394–401, 2000. doi:10.1002/1520-6378(200012)25:63.0.CO;2-Y.

[Nat16] Graeme Nattress. Private Discussion with Shaw, N. 2016.

[Nay97] Yoshinobu Nayatani. Simple estimation methods for the Helmholtz—Kohlrausch effect. *Color Research & Application*, 22(6):385–401, 1997. doi:10.1002/(SICI)1520-6378(199712)22:6{$<$}385::AID-COL6{$>$}3.0.CO;2-R.

[NSY95] Yoshinobu Nayatani, Hiroaki Sobagaki, and Kenjiro Hashimoto Tadashi Yano. Lightness dependency of chroma scales of a nonlinear color-appearance model and its latest formulation. *Color Research & Application*, 20(3):156–167, June 1995. doi:10.1002/col.5080200305.

[NNJ43] Sidney M. Newhall, Dorothy Nickerson, and Deane B. Judd. Final Report of the OSA Subcommittee on the Spacing of the Munsell Colors. *Journal of the Optical Society of America*, 33(7):385, July 1943. doi:10.1364/JOSA.33.000385.

[Ohn05] Yoshi Ohno. Spectral design considerations for white LED color rendering. *Optical Engineering*, 44(11):111302, 2005. doi:10.1117/1.2130694.

[Ohn14] Yoshiro Ohno. Practical Use and Calculation of CCT and Duv. *LEUKOS*, 10(1):47–55, January 2014. doi:10.1080/15502724.2014.839020.

[OD08] Yoshiro Ohno and Wendy Davis. NIST CQS simulation. 2008.

[OD13] Yoshiro Ohno and Wendy Davis. NIST CQS simulation. 2013.

[Oht97] N. Ohta. The basis of color reproduction engineering. 1997.

[OYH18] H. Otsu, M. Yamamoto, and T. Hachisuka. Reproducing Spectral Reflectances From Tristimulus Colours. *Computer Graphics Forum*, 37(6):370–381, September 2018. doi:10.1111/cgf.13332.

[Ott20] Björn Ottosson. A perceptual color space for image processing. https://bottosson.github.io/posts/oklab/, 2020.

[Poi80] Michael R. Pointer. Pointer's Gamut Data. 1980.

[RF21] Luvin Munish Ragoo and Ivar Farup. Optimising a Euclidean Colour Space Transform for Colour Order and Perceptual Uniformity. *Color and Imaging Conference*, 29(1):282–287, November 2021. doi:10.2352/issn.2169-2629.2021.29.282.

[Rak17] Herilalaina Rakotoarison. Bunch. March 2017.

[SCKL17] Muhammad Safdar, Guihua Cui, Youn Jin Kim, and Ming Ronnier Luo. Perceptually uniform color space for image signals including high dynamic range and wide gamut. *Optics Express*, 25(13):15131, June 2017. doi:10.1364/OE.25.015131.

[SHKL18] Muhammad Safdar, Jon Y. Hardeberg, Youn Jin Kim, and Ming Ronnier Luo. A Colour Appearance Model based on J z a z b z Colour Space. *Color and Imaging Conference*, 2018(1):96–101, November 2018. doi:10.2352/ISSN.2169-2629.2018.26.96.

[SHRL21] Muhammad Safdar, Jon Yngve Hardeberg, and Ming Ronnier Luo. ZCAM, a colour appearance model based on a high dynamic range uniform colour space. *Optics Express*, 29(4):6036, February 2021. doi:10.1364/OE.413659.

[Sar21] Madenda Sarifuddin. RGB to HCL and HCL to RGB color conversion. 2021.

[SM05a] Madenda Sarifuddin and Rokia Missaoui. A New Perceptually Uniform Color Space with Associated Color Similarity Measure for ContentBased Image and Video Retrieval. 2005.

[SM05b] Madenda Sarifuddin and Rokia Missaoui. HCL: a new Color Space for a more Effective Content-based Image Retrieval. 2005.

[SWD05] Gaurav Sharma, Wencheng Wu, and Edul N. Dalal. The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1):21–30, February 2005. doi:10.1002/col.20070.

[SH15] Peter Shirley and David Hart. The prismatic color space for rgb computations. 2015.

[Sir18] Daniele Siragusano. Private Discussion with Shaw, Nick. 2018.

[Smi78] Alvy Ray Smith. Color gamut transform pairs. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '78*, 12–19. New York, New York, USA, 1978. ACM Press. doi:10.1145/800248.807361.

[Smi99] Brian Smits. An RGB-to-Spectrum Conversion for Reflectances. *Journal of Graphics Tools*, 4(4):11–22, January 1999. 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.

[Spi15] Nick Spiker. Private Discussion with Mansencal, T. 2015.

[SS88] E. I. Stearns and R. E. Stearns. An example of a method for correcting radiance data for Bandpass error. *Color Research & Application*, 13(4):257–259, August 1988. doi:10.1002/col.5080130410.

[SS00] Andrew Stockman and Lindsay T. Sharpe. Cone Fundamentals. http://www.cvrl.org/cones.htm, 2000.

[SBS99] Sabine Susstrunk, Robert Buckley, and Steve Swen. Standard RGB Color Spaces. 1999.

[SHF00] Sabine E. Susstrunk, Jack M. Holm, and Graham D. Finlayson. Chromatic adaptation performance of different RGB sensors. In Reiner Eschbach and Gabriel G. Marcu, editors, *Photonics West 2001 - Electronic Imaging*, volume 4300, 172–183. December 2000. doi:10.1117/12.410788.

[Tho12] Larry Thorpe. CANON-LOG TRANSFER CHARACTERISTIC. 2012.

[Tri15] Tashi Trieu. Private Discussion with Mansencal, T. 2015.

[War16] Greg Ward. Private Discussion with Mansencal, T. 2016.

[WEydelbergVileshin02] Greg Ward and Elena Eydelberg-Vileshin. Picture Perfect RGB Rendering Using Spectral Prefiltering and Sharp Color Primaries. *Eurographics workshop on Rendering*, pages 117–124, 2002. doi:10.2312/EGWR/EGWR02/117-124.

[WY12] Andrew B. Watson and John I. Yellott. A unified formula for light-adapted pupil size. *Journal of Vision*, 12(10):12, September 2012. doi:10.1167/12.10.12.

[WR04] Stephen Westland and Caterina Ripamonti. Table 8.2. In *Computational Colour Science Using MATLAB*, pages 137. John Wiley & Sons, Ltd, Chichester, UK, 1 edition, March 2004. doi:10.1002/0470020326.

[WRC12a] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Correction for Spectral Bandpass. In *Computational Colour Science Using MATLAB*, pages 38. 2 edition, 2012.

[WRC12b] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Extrapolation Methods. In *Computational Colour Science Using MATLAB*, pages 38. 2 edition, 2012.

[WRC12c] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Interpolation Methods. In *Computational Colour Science Using MATLAB*, pages 29–37. 2 edition, 2012.

[WRC12d] Stephen Westland, Caterina Ripamonti, and Vien Cheung. CMCCAT2000. In *Computational Colour Science Using MATLAB*, pages 83–86. 2 edition, 2012.

[WRC12e] Stephen Westland, Caterina Ripamonti, and Vien Cheung. CMCCAT97. In *Computational Colour Science Using MATLAB*, pages 80. 2 edition, 2012.

[Woo14] Mike Wood. Making the same color twice - A proposed PLASA standard for color communication. 2014.

[Wys63] Günter Wyszecki. Proposal for a New Color-Difference Formula. *Journal of the Optical Society of America*, 53(11):1318, November 1963. doi:10.1364/JOSA.53.001318.

[WS00a] Günther Wyszecki and W S Stiles. Equation I(1.2.1). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 8. Wiley, 2000.

[WS00b] Günther Wyszecki and W S Stiles. Table 2(5.4.1) MacAdam Ellipses (Observer PGN) Observed and Calculated on the Basis of a Normal Distribution of Color Matches about a Color Center (Silberstein and MacAdam, 1945). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 309. Wiley, 2000.

[WS00c] Günther Wyszecki and W. S. Stiles. Integration Replaced by Summation. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 158–163. Wiley, 2000.

[WS00d] 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.

[WS00e] 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.

[WS00f] 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.

[WS00g] 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.

[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. The CIE 1964 Standard Observer. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 141. Wiley, 2000.

[WS00k] 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.

[WS00l] 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.

[WS00m] 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.

[Yor14] Rory Yorke. Python: Change format of np.array or allow tolerance in in1d function. http://stackoverflow.com/a/23521245/931625, 2014.

[ZL18] Qiyan Zhai and Ming R. Luo. Study of chromatic adaptation via neutral white matches on different viewing media. *Optics Express*, 26(6):7724, March 2018. doi:10.1364/OE.26.007724.

[AdobeSystems05] Adobe Systems. Adobe RGB (1998) Color Image Encoding. 2005.

[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::xy_coord. 2013.

[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::Set_xy_coord. 2013.

[AdobeSystems13c] Adobe Systems. Cube LUT Specification. 2013.

[ANSI03] ANSI. Specification of ROMM RGB. 2003.

[ANSIIESCCommittee18] ANSI and IES Color Committee. *ANSI/IES TM-30-18 - IES Method for Evaluating Light Source Color Rendition*. ANSI/IES, 2018. ISBN 978-0-87995-379-9.

[AppleInc19] Apple Inc. displayP3. https://developer.apple.com/documentation/coregraphics/cgcolorspace/1408916-displayp3, 2019.

[ARRI12] ARRI. ALEXA - Log C Curve - Usage in VFX. 2012.

[AssociationoRIaBusinesses15] Association of Radio Industries and Businesses. Essential Parameter Values for the Extended Image Dynamic Range Television (EIDRTV) System for Programme Production. 2015.

[ASTMInternational89] ASTM International. ASTM D1535-89 - Standard Practice for Specifying Color by the Munsell System. 1989.

[ASTMInternational07] ASTM International. ASTM D2244-07 - Standard Practice for Calculation of Color Tolerances and Color Differences from Instrumentally Measured Color Coordinates. 2007. doi:10.1520/D2244-16.

[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.

[ASTMInternational15a] ASTM International. ASTM E308-15 - Standard Practice for Computing the Colors of Objects by Using the CIE System. 2015. doi:10.1520/E0308-15.

[ASTMInternational15b] ASTM International. ASTM E313-15e1 - Standard Practice for Calculating Yellowness and Whiteness Indices from Instrumentally Measured Color Coordinates. 2015. doi:10.1520/E0313-20.

[BabelColor12a] BabelColor. The ColorChecker (since 1976!). http://www.babelcolor.com/main_level/ColorChecker.htm 2012.

[BabelColor12b] BabelColor. ColorChecker RGB and spectra. 2012.

[BlackmagicDesign20a] Blackmagic Design. Wide Gamut Intermediate DaVinci Resolve. 2020.

[BlackmagicDesign20b] Blackmagic Design. DaVinci Wide Gamut - DaVinci Resolve Studio 17 Public Beta 1. November 2020.

[BlackmagicDesign21] Blackmagic Design. Blackmagic Generation 5 Color Science. 2021.

[Canon14] Canon. EOS C500 Firmware Update. https://www.usa.canon.com/internet/portal/us/home/explore/produ showcases/cameras-and-lenses/cinema-eos-firmware/c500, 2014.

[Canon16] Canon. Input Transform Version 201612 for EOS C300 Mark II. https://www.usa.canon.com/internet/portal/us/home/support/details/cameras/cinema-eos/eos-c300-mark-ii, 2016.

[Canon20] Canon. Input Transform Version 202007 for EOS C300 Mark II. https://www.usa.canon.com/internet/portal/us/home/support/details/cameras/cinema-eos/eos-c300-mark-ii, 2020.

[CIED1CIED822] CIE Division 1 and CIE Division 8. *CIE 248:2022 The CIE 2016 Colour Appearance Model for Colour Management Systems: CIECAM16*. Commission Internationale de l'Eclairage, 2022. ISBN 978-3-902842-94-7.

[CIET13294] CIE TC 1-32. *CIE 109-1994 A Method of Predicting Corresponding Colours under Different Chromatic and Illuminance Adaptations*. Commission Internationale de l'Eclairage, 1994. ISBN 978-3-900734-51-0.

[CIET13606] CIE TC 1-36. *CIE 170-1:2006 Fundamental Chromaticity Diagram with Physiological Axes - Part 1*. Commission Internationale de l'Eclairage, 2006. ISBN 978-3-901906-46-6.

[CIET13805a] CIE TC 1-38. 9. INTERPOLATION. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, pages 14–19. 2005.

[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*, pages 1–27. 2005.

[CIET13805c] 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*, pages 19. 2005.

[CIET13805d] CIE TC 1-38. EXTRAPOLATION. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, pages 19–20. 2005.

[CIET14804a] CIE TC 1-48. 3.1 Recommendations concerning standard physical data of illuminants. In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 12–13. 2004.

[CIET14804b] CIE TC 1-48. 9.1 Dominant wavelength and purity. In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 32–33. 2004.

[CIET14804c] CIE TC 1-48. Extrapolation. In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 24. 2004.

[CIET14804d] CIE TC 1-48. The evaluation of whiteness. In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 24. 2004.

[CIET14804e] CIE TC 1-48. APPENDIX E. INFORMATION ON THE USE OF PLANCK'S EQUATION FOR STANDARD AIR. In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 77–82. 2004.

[CIET14804f] CIE TC 1-48. *CIE 015:2004 Colorimetry, 3rd Edition*. Commission Internationale de l'Eclairage, 2004. ISBN 978-3-901906-33-6.

[CIET14804g] CIE TC 1-48. CIE 1976 uniform chromaticity scale diagram (UCS diagram). In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 24. 2004.

[CIET14804h] CIE TC 1-48. CIE 1976 uniform colour spaces. In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 24. 2004.

[CIET14804i] CIE TC 1-48. EXPLANATORY COMMENTS - 5. In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 68–68. 2004.

[CIET19017] CIE TC 1-90. *CIE 2017 colour fidelity index for accurate scientific use*. Number 224 in Technical report / CIE. CIE Central Bureau, Vienna, 2017. ISBN 978-3-902842-61-9.

[CIE] CIE. CIE Spectral Data.

[CIE04] CIE. CIE 15:2004 Tables Data. 2004.

[Colblindora] Colblindor. Deuteranopia - Red-Green Color Blindness. http://www.color-blindness.com/deuteranopia-red-green-color-blindness/.

[Colblindorb] Colblindor. Protanopia - Red-Green Color Blindness. http://www.color-blindness.com/protanopia-red-green-color-blindness/.

[Colblindorc] Colblindor. Tritanopia - Blue-Yellow Color Blindness. http://www.color-blindness.com/tritanopia-blue-yellow-color-blindness/.

[CVRLa] CVRL. Luminous efficiency. http://www.cvrl.org/lumindex.htm.

[CVRLb] CVRL. New CIE XYZ functions transformed from the CIE (2006) LMS functions. http://cvrl.ioo.ucl.ac.uk/ciexyzpr.htm.

[CVRLc] CVRL. Older CIE Standards. http://cvrl.ioo.ucl.ac.uk/cie.htm.

[CVRLd] CVRL. Stiles & Burch individual 10-deg colour matching data. http://www.cvrl.org/stilesburch10_ind.htm.

[CVRLe] CVRL. Stiles & Burch individual 2-deg colour matching data. http://www.cvrl.org/stilesburch2_ind.htm.

[CVRLf] CVRL. CIE (2012) 10-deg XYZ "physiologically-relevant" colour matching functions. http://www.cvrl.org/database/text/cienewxyz/cie2012xyz10.htm.

[CVRLg] CVRL. CIE (2012) 2-deg XYZ "physiologically-relevant" colour matching functions. http://www.cvrl.org/database/text/cienewxyz/cie2012xyz2.htm.

[DigitalCInitiatives07] Digital Cinema Initiatives. Digital Cinema System Specification - Version 1.1. 2007.

[DjangoSFoundation22] Django Software Foundation. Slugify. 2022.

[Dji17] Dji. White Paper on D-Log and D-Gamut of DJI Cinema Color System. 2017.

[Dolby16] Dolby. WHAT IS ICTCP? - INTRODUCTION. 2016.

[EasyRGBa] EasyRGB. CMYK –$>$ CMY. http://www.easyrgb.com/index.php?X=MATH&H=14#text14.

[EasyRGBb] EasyRGB. CMY –$>$ CMYK. http://www.easyrgb.com/index.php?X=MATH&H=13#text13.

[EasyRGBc] EasyRGB. CMY –$>$ RGB. http://www.easyrgb.com/index.php?X=MATH&H=12#text12.

[EasyRGBd] EasyRGB. HSL –$>$ RGB. http://www.easyrgb.com/index.php?X=MATH&H=19#text19.

[EasyRGBe] EasyRGB. HSV –$>$ RGB. http://www.easyrgb.com/index.php?X=MATH&H=21#text21.

[EasyRGBf] EasyRGB. RGB –$>$ CMY. http://www.easyrgb.com/index.php?X=MATH&H=11#text11.

[EasyRGBg] EasyRGB. RGB –$>$ HSL. http://www.easyrgb.com/index.php?X=MATH&H=18#text18.

[EasyRGBh] EasyRGB. RGB –$>$ HSV. http://www.easyrgb.com/index.php?X=MATH&H=20#text20.

[EuropeanBUnion75] European Broadcasting Union. EBU Tech 3213 - EBU Standard for Chromaticity Tolerances for Studio Monitors. August 1975.

[EuropeanCInitiative02] European Color Initiative. ECI RGB v2. 2002.

[FFmpegDevelopers22a] FFmpeg Developers. FFmpeg::AVColorPrimaries. August 2022.

[FFmpegDevelopers22b] FFmpeg Developers. FFmpeg::AVColorSpace. August 2022.

[FFmpegDevelopers22c] FFmpeg Developers. FFmpeg::AVColorTransferCharacteristic. August 2022.

[FiLMiCInc17] FiLMiC Inc. FiLMiC Pro - User Manual v6 - Revision 1. 2017.

[Fujifilm22a] Fujifilm. F-Log Data Sheet Ver.1.1. 2022.

[Fujifilm22b] Fujifilm. F-Log2 Data Sheet Ver.1.0. 2022.

[GoProDM16] GoPro, Haarm-Pieter Duiker, and Thomas Mansencal. Gopro.py. https://github.com/hpd/OpenColorIO-Configs/blob/master/aces_1.0.3/python/aces_ocio/colorspaces/go 2016.

[HernandezAndresLR99] Javier Hernández-Andrés, Raymond L. Lee, and Javier Romero. Calculating correlated color temperatures across the entire gamut of daylight and skylight chromaticities. *Applied Optics*, 38(27):5703, September 1999. doi:10.1364/AO.38.005703.

[HewlettPDCompany09] Hewlett-Packard Development Company. Understanding the HP DreamColor LP2480zx DCI-P3 Emulation Color Space. 2009.

[HunterLab08a] HunterLab. Hunter L,a,b Color Scale. 2008.

[HunterLab08b] HunterLab. Illuminant Factors in Universal Software and EasyMatch Coatings. 2008.

[HunterLab12] HunterLab. Hunter Rd,a,b Color Scale - History and Application. 2012.

[HutchColora] HutchColor. BestRGB (4 K).

[HutchColorb] HutchColor. DonRGB4 (4 K).

[HutchColorc] HutchColor. MaxRGB (4 K).

[HutchColord] HutchColor. XtremeRGB (4 K).

[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*. Illuminating Engineering Society, 2014. ISBN 978-0-87995-295-2.

[ImageEngineering17] Image Engineering. TE226 V2 data sheet. 2017.

[InternationalCConsortium10] International Color Consortium. Specification ICC.1:2010 (Profile version 4.3.0.0). 2010.

[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.

[InternationalOfStandardization02] International Organization for Standardization. INTERNATIONAL STANDARD ISO 7589-2002 - Photography - Illuminants for sensitometry - Specifications for daylight, incandescent tungsten and printer. 2002.

[InternationalOfStandardization12] International Organization for Standardization. INTERNATIONAL STANDARD ISO 17321-1 - Graphic technology and photography - Colour characterisation of digital still cameras (DSCs) - Part 1: Stimuli, metrology and test procedures. 2012.

[InternationalOfStandardization13] International Organization for Standardization. INTERNATIONAL STANDARD ISO/IEC 23001-8 - Information technology - MPEG systems technologies - Part 8: Coding-independent code points. 2013.

[InternationalOfStandardization20] International Organization for Standardization. INTERNATIONAL STANDARD ISO/IEC 14496-10 - Information technology - Coding of audio-visual objects - Part 10: Advanced video coding. 2020.

[InternationalOfStandardization21] International Organization for Standardization. INTERNATIONAL STANDARD ISO/IEC 23091-2 - Information technology - Coding- independent code points - Part 2: Video. 2021.

[InternationalTUnion98a] International Telecommunication Union. Recommendation ITU-R BT.1361 - Worldwide unified colorimetry and related characteristics of future television and imaging systems. 1998.

[InternationalTUnion98b] International Telecommunication Union. Recommendation ITU-R BT.470-6 - CONVENTIONAL TELEVISION SYSTEMS. 1998.

[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.

[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.

[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.

[InternationalTUnion15a] International Telecommunication Union. Recommendation ITU-R BT.2020 - Parameter values for ultra-high definition television systems for production and international programme exchange. 2015.

[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.

[InternationalTUnion15c] International Telecommunication Union. Report ITU-R BT.2246-4 - The present state of ultra-high definition television BT Series Broadcasting service. 2015.

[InternationalTUnion17] 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. 2017.

[InternationalTUnion18] International Telecommunication Union. Recommendation ITU-R BT.2100-2 - Image parameter values for high dynamic range television for use in production and international programme exchange. 2018.

[InternationalTUnion19] International Telecommunication Union. Recommendation ITU-R BT.2124-0 - Objective metric for the assessment of the potential visibility of colour differences in television. 2019.

[InternationalTUnion21] International Telecommunication Union. Recommendation ITU-T H.273 - Coding-independent code points for video signal type identification. 2021.

[Laurent12] Laurent. Reproducibility of python pseudo-random numbers across systems and versions? http://stackoverflow.com/questions/8786084/reproducibility-of-python-pseudo-random-numbers-across-systems-and-versions, 2012.

[LeicaCAG22] Leica Camera AG. Leica L-Log Reference Manual. 2022.

[MartinezVerduPC+07] Francisco Martínez-Verdú, Esther Perales, Elisabet Chorro, Dolores de Fez, Valentín Viqueira, and Eduardo Gilabert. Computation and visualization of the MacAdam limits for any lightness, hue angle, and light source. *Journal of the Optical Society of America A*, 24(6):1501, June 2007. doi:10.1364/JOSAA.24.001501.

[MunsellCSciencea] Munsell Color Science. Macbeth Colorchecker.

[MunsellCScienceb] Munsell Color Science. Munsell Colours Data. 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.

[Nikon18] Nikon. N-Log Specification Document - Version 1.0.0. 2018.

[Panasonic14] Panasonic. VARICAM V-Log/V-Gamut. 2014.

[PLASANAmerica15] PLASA North America. ANSI E1.54 - 2015 - PLASA Standard for Color Communication in Entertainment Lighting. 2015.

[REDDCinema17] RED Digital Cinema. White Paper on REDWideGamutRGB and Log3G10. 2017.

[RenewableRDCenter03] Renewable Resource Data Center. Reference Solar Spectral Irradiance: ASTM G-173. http://rredc.nrel.gov/solar/spectra/am1.5/ASTMG173/ASTMG173.html, 2003.

[RisingSResearch] Rising Sun Research. cineSpace LUT Library. https://sourceforge.net/projects/cinespacelutlib/.

[Saeedn] Saeedn. Extend a line segment a specific distance. http://stackoverflow.com/questions/7740507/extend-a-line-segment-a-specific-distance.

[sastanin] sastanin. How to make scipy.interpolate give an extrapolated result beyond the input range? http://stackoverflow.com/a/2745496/931625.

[SocietyoMPaTEngineers93] Society of Motion Picture and Television Engineers. *RP 177:1993 - Derivation of Basic Television Color Equations*. Volume RP 177:199. The Society of Motion Picture and Television Engineers, January 1993. ISBN 978-1-61482-191-5. 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.

[SocietyoMPaTEngineers04] Society of Motion Picture and Television Engineers. *RP 145:2004: SMPTE C Color Monitor Colorimetry*. Volume RP 145:200. The Society of Motion Picture and Television Engineers, January 2004. ISBN 978-1-61482-164-9. 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. doi:10.5594/SMPTE.ST2084.2014.

[SocietyoMPaTEngineers19] Society of Motion Picture and Television Engineers. ST 428-1:2019 - D-Cinema Distribution Master — Image Characteristic. 2019. doi:10.5594/SMPTE.ST428-1.2019.

[SonyCorporationa] Sony Corporation. S-Gamut3_S-Gamut3Cine_Matrix.xlsx.

[SonyCorporationb] Sony Corporation. S-Log Whitepaper.

[SonyCorporationc] Sony Corporation. Technical Summary for S-Gamut3.Cine/S-Log3 and S-Gamut3/S-Log3.

[SonyCorporation12] Sony Corporation. S-Log2 Technical Paper. 2012.

[SonyECorporation20a] Sony Electronics Corporation. IDT.Sony.Venice_SLog3_SGamut3Cine.ctl. 2020.

[SonyECorporation20b] Sony Electronics Corporation. IDT.Sony.Venice_SLog3_SGamut3.ctl. 2020.

[SonyImageworks12] Sony Imageworks. Make.py. https://github.com/imageworks/OpenColorIO-Configs/blob/master/nuke-default/make.py, 2012.

[TheAoMPAaSciences20] The Academy of Motion Picture Arts and Sciences. Academy Spectral Similarity Index (SSI): Overview. 2020.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteea] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Academy Color Encoding System. http://www.oscars.org/science-technology/council/projects/aces.html.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteeb] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. ACESutil.Lin_to_Log2_param.ctl. https://github.com/ampas/aces-dev/blob/518c27f577e99cdecfddf2ebcfaa53444b1f9343/transforms/ctl/utilities/ACESutil.Lin_to_Log2_pa

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommitteec] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. ACESutil.Log2_to_Lin_param.ctl. https://github.com/ampas/aces-dev/blob/518c27f577e99cdecfddf2ebcfaa53444b1f9343/transforms/ctl/utilities/ACESutil.Log2_to_Lin_pa

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee13] 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. 2013.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a] 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.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b] 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.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Technical Bulletin TB-2014-012 - Academy Color Encoding System Version 1.0 Component Names. 2014.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee15] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Procedure P-2013-001 - Recommended Procedures for the Creation and Use of Digital Camera System Input Device Transforms (IDTs). 2015.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee20] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2014-006 - Common LUT Format (CLF) - A Common File Format for Look-Up Tables. 2020.

[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. http://j.mp/S-2016-001, 2016.

[TheAoMPAaSciencesScienceandTCouncilAcademyCESACESPSubcommittee15] 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.

[VincentJ17] Vincent J. Is there any numpy group by function? https://stackoverflow.com/a/43094244, 2017.

[W3C22] W3C. CSS Color Module Level 3. 2022.

[Wikipedia] Wikipedia. Ellipse. https://en.wikipedia.org/wiki/Ellipse.

[Wikipedia01a] Wikipedia. Approximation. http://en.wikipedia.org/wiki/Color_temperature#Approximation, 2001.

[Wikipedia01b] Wikipedia. Color temperature. http://en.wikipedia.org/wiki/Color_temperature, 2001.

[Wikipedia01c] Wikipedia. Luminance. https://en.wikipedia.org/wiki/Luminance, 2001.

[Wikipedia01d] Wikipedia. Rayleigh scattering. http://en.wikipedia.org/wiki/Rayleigh_scattering, 2001.

[Wikipedia03a] Wikipedia. Lagrange polynomial - Definition. https://en.wikipedia.org/wiki/Lagrange_polynomial#Defi 2003.

[Wikipedia03b] Wikipedia. Luminosity function. https://en.wikipedia.org/wiki/Luminosity_function#Details, 2003.

[Wikipedia03c] Wikipedia. Mean squared error. https://en.wikipedia.org/wiki/Mean_squared_error, 2003.

[Wikipedia03d] Wikipedia. Michaelis-Menten kinetics. https://en.wikipedia.org/wiki/Michaelis%E2%80%93Menten_ki 2003.

[Wikipedia03e] Wikipedia. Rayleigh–Jeans law. https://en.wikipedia.org/wiki/Rayleigh–Jeans_law, 2003.

[Wikipedia03f] Wikipedia. Vandermonde matrix. https://en.wikipedia.org/wiki/Vandermonde_matrix, 2003.

[Wikipedia03g] Wikipedia. HSL and HSV. http://en.wikipedia.org/wiki/HSL_and_HSV, 2003.

[Wikipedia04a] Wikipedia. Peak signal-to-noise ratio. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio, 2004.

[Wikipedia04b] Wikipedia. Surfaces. http://en.wikipedia.org/wiki/Gamut#Surfaces, 2004.

[Wikipedia04c] Wikipedia. Whiteness. http://en.wikipedia.org/wiki/Whiteness, 2004.

[Wikipedia04d] Wikipedia. Wide-gamut RGB color space. http://en.wikipedia.org/wiki/Wide-gamut_RGB_color_space, 2004.

[Wikipedia04e] Wikipedia. YCbCr. https://en.wikipedia.org/wiki/YCbCr, 2004.

[Wikipedia05a] Wikipedia. Lanczos resampling. https://en.wikipedia.org/wiki/Lanczos_resampling, 2005.

[Wikipedia05b] Wikipedia. Luminous Efficacy. https://en.wikipedia.org/wiki/Luminous_efficacy, 2005.

[Wikipedia05c] Wikipedia. Mesopic weighting function. http://en.wikipedia.org/wiki/Mesopic_vision#Mesopic_weighti 2005.

[Wikipedia05d] Wikipedia. CIE 1931 color space. http://en.wikipedia.org/wiki/CIE_1931_color_space, 2005.

[Wikipedia05e] Wikipedia. ISO 31-11. https://en.wikipedia.org/wiki/ISO_31-11, 2005.

[Wikipedia06a] Wikipedia. List of common coordinate transformations. http://en.wikipedia.org/wiki/List_of_common_coordinate_transformations, 2006.

[Wikipedia06b] Wikipedia. White points of standard illuminants. http://en.wikipedia.org/wiki/Standard_illuminant#Wh 2006.

[Wikipedia07a] Wikipedia. Lightness. http://en.wikipedia.org/wiki/Lightness, 2007.

[Wikipedia07b] Wikipedia. The reverse transformation. http://en.wikipedia.org/wiki/CIELUV#The_reverse_transformat 2007.

[Wikipedia07c] Wikipedia. CAT02. http://en.wikipedia.org/wiki/CIECAM02#CAT02, 2007.

[Wikipedia07d] Wikipedia. CIECAM02. http://en.wikipedia.org/wiki/CIECAM02, 2007.

[Wikipedia07e] Wikipedia. CIELUV. http://en.wikipedia.org/wiki/CIELUV, 2007.

[Wikipedia08a] Wikipedia. Color difference. http://en.wikipedia.org/wiki/Color_difference, 2008.

[Wikipedia08b] Wikipedia. Relation to CIE XYZ. http://en.wikipedia.org/wiki/CIE_1960_color_space#Relation_to_CIE_ 2008.

[Wikipedia08c] Wikipedia. CIE 1960 color space. http://en.wikipedia.org/wiki/CIE_1960_color_space, 2008.

[Wikipedia08d] Wikipedia. CIE 1964 color space. http://en.wikipedia.org/wiki/CIE_1964_color_space, 2008.

[Wikipedia15] Wikipedia. HCL color space. https://en.wikipedia.org/wiki/HCL_color_space, 2015.

[XRite16] X-Rite. New color specifications for ColorChecker SG and Classic Charts. http://xritephoto.com/ph_product_overview.aspx?ID=938&Action=Support&SupportID=5884#, 2016.

[XRitePantone12] X-Rite and Pantone. Color iQC and Color iMatch Color Calculations Guide. 2012.

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z