LuxPy Documentation

Release 1.9.8

Kevin A.G. Smet

Jun 23, 2022

CONTENTS:

- Author: K.A.G. Smet (ksmet1977 at gmail.com)

- Version: 1.9.8

- Date: June 23, 2022

- License: GPLv3

ONE

LICENSE: GPLV3

Copyright (C) <2017><Kevin A.G. Smet> (ksmet1977 at gmail.com)

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

INSTALLATION

2.1 Install luxpy

1. **Install miniconda**

   - download the installer from: https://conda.io/miniconda.html or https://repo.continuum.io/miniconda/)

   - e.g. https://repo.continuum.io/miniconda/Miniconda3-latest-Windows-x86_64.exe

   - Make sure 'conda.exe' can be found on the windows system path, if necessary do a manual add.

2. Create a virtual environment with full anaconda distribution by typing the following at the commandline:

   ```
   >> conda create --name py36 python=3.6 anaconda
   ```

3. Activate the virtual environment:

   ```
   >> activate py36
   ```

4. **Install pip to virtual environment (just to ensure any packages to be** installed with pip to this virt. env. will be installed here and not globally):

   ```
   >> conda install -n py36 pip
   ```

5a. Install luxpy package from pypi:

   ```
   >> pip install luxpy
   ```

5b. Install luxpy package from anaconda:

   ```
   >> conda install -c ksmet1977 luxpy
   ```

Note If any errors show up, try and do a manual install of the dependencies: scipy, numpy, pandas, matplotlib and setuptools, either using e.g. `>> conda install scipy` or `>> pip install scipy`, and try and reinstall luxpy using pip.

2.2 Use of LuxPy package in Spyder IDE

6. Install spyder in py36 environment:

   ```
   >> conda install -n py36 spyder
   ```

7. Run spyder

   ```
   >> spyder
   ```

8. To import the luxpy package, on Spyder's commandline for the IPython kernel (or in script) type:

   ```
   import luxpy as lx
   ```

2.3 Use of LuxPy package in Jupyter notebook

6. Install jupyter in py36 environment:

   ```
   >> conda install -n py36 jupyter
   ```

7. Start jupyter notebook:

   ```
   >> jupyter notebook
   ```

8. **Open an existing or new notebook:** e.g. open "luxpy_basic_usage.ipynb" for an overview of how to use the LuxPy package.

9. To import LuxPy package type:

   ```
   import luxpy as lx
   ```

IMPORTED (REQUIRED) PACKAGES

3.1 Core

- import os
- import warnings
- import pathlib
- import importlib
- from collections import OrderedDict as odict
- from mpl_toolkits.mplot3d import Axes3D
- import colorsys
- import itertools
- import copy
- import time
- import tkinter
- import ctypes
- import platform
- import subprocess
- import cProfile
- import pstats
- import io

3.2 3e party dependencies (automatic install)

- import numpy as np
- import pandas as pd
- import matplotlib.pyplot as plt
- import scipy as sp
- import imageio

3.3 3e party dependencies (automatic install on import)

- import pyswarms (when importing particleswarms from math)

3.4 3e party dependencies (requiring manual install)

To control Ocean Optics spectrometers with spectro toolbox:

- import seabreeze (conda install -c poehlmann python-seabreeze)
- pip install pyusb (for use with 'pyseabreeze' backend of python-seabreeze)

FOUR

LUXPY PACKAGE STRUCTURE

4.1 Utils sub-package

py

- __init__.py

- utilities.py

- folder_tree.py

namespace luxpy.utils

luxpy.utils.**np2d**(*data*)
    Make a tuple, list or numpy array at least a 2D numpy array.

   Args:

   data

   tuple, list, ndarray

   Returns:

   returns

   ndarray with .ndim >= 2

luxpy.utils.**np3d**(*data*)
    Make a tuple, list or numpy array at least a 3d numpy array.

   Args:

   data

   tuple, list, ndarray

   Returns:

   returns

   ndarray with .ndim >= 3

luxpy.utils.**np2dT**(*data*)
    Make a tuple, list or numpy array at least a 2D numpy array and transpose.

   Args:

   data

   tuple, list, ndarray

   Returns:

> **returns**
>
>> ndarray with .ndim >= 2 and with transposed axes.

luxpy.utils.**np3dT**(*data*)

Make a tuple, list or numpy array at least a 3d numpy array and transposed first 2 axes.

> **Args:**
>
>> **data**
>>
>>> tuple, list, ndarray
>
> **Returns:**
>
>> **returns**
>>
>>> ndarray with .ndim >= 3 and with first two axes
>>> transposed (axis=3 is kept the same).

luxpy.utils.**put_args_in_db**(*db*, *args*)

Takes the args with not-None input values of a function and overwrites the values of the corresponding keys in dict db. | (args are collected with the built-in function locals(), | See example usage below)

> **Args:**
>
>> **db**
>>
>>> dict
>
> **Returns:**
>
>> **returns**
>>
>>> dict with the values of specific keys overwritten by the
>>>> not-None values of corresponding args of a function fcn.
>
> **Example usage:**
>
>> _db = {'c' : 'c1', 'd' : 10, 'e' : {'e1':'hello', 'e2':1000}}
>>
>> def test_put_args_in_db(a, b, db = None, c = None,d = None,e = None):
>>
>>> args = locals().copy() # get dict with keyword input arguments to
>>>> # function 'test_put_args_in_db'
>>
>>> db = put_args_in_db(db,args) # overwrite non-None args in db copy.
>>
>>> if db is not None: # unpack db for further use
>>>> c,d,e = [db[x] for x in sorted(db.keys())]
>>
>>> print(' a : {}'.format(a))
>>> print(' b : {}'.format(b))
>>> print(' db: {}'.format(db))
>>> print(' c : {}'.format(c))
>>> print(' d : {}'.format(d))
>>> print(' e : {}'.format(e))
>>> print('_db: {}'.format(_db))

`luxpy.utils.`**`vec_to_dict`**(*vec=None*, *dic={}*, *vsize=None*, *keys=None*)

> Convert dict to vec and vice versa.

> **Args:**

>> **vec**

>>> list or vector array, optional

>> **dic**

>>> dict, optional

>> **vsize**

>>> list or vector array with size of values of dict, optional

>> **keys**

>>> list or vector array with keys in dict (must be provided).

> **Returns:**

>> **returns**

>>> x, vsize

>>>> x is an array, if vec is None

>>>> x is a dict, if vec is not None

`luxpy.utils.`**`getdata`**(*data*, *kind='np'*, *columns=None*, *header=None*, *sep=', '*, *datatype='S'*, *copy=True*, *verbosity=True*)

> Get data from csv-file or convert between pandas dataframe and numpy 2d-array.

> **Args:**

>> **data**

>>> - str with path to file containing data
>>> - ndarray with data
>>> - pandas.dataframe with data

>> **kind**

>>> str ['np','df'], optional
>>> Determines type(:returns:), np: ndarray, df: pandas.dataframe

>> **columns**

>>> None or list[str] of column names for dataframe, optional

>> **header**

>>> None, optional
>>>> - None: no header in file
>>>> - 'infer': infer headers from file

>> **sep**

>>> ',' or ' ' or other char, optional
>>> Column separator in data file

>> **datatype'**

>>> 'S',optional
>>> Specifies a type of data.
>>> Is used when creating column headers (:column: is None).

-'S': light source spectrum

-'R': reflectance spectrum

or other.

copy

True, optional

Return a copy of ndarray if kind == 'np', or copy of pd.DataFrame if kind == 'df'

verbosity

True, False, optional

Print warning when inferring headers from file.

Returns:

returns

data as ndarray or pandas.dataframe

luxpy.utils.**dictkv**(*keys=None*, *values=None*, *ordered=True*)

Easy input of of keys and values into dict.

Args:

keys

iterable list[str,...] of keys

values

iterable list[...,...,] of values

ordered

True, False, optional

True: creates an ordered dict using 'collections.OrderedDict()'

Returns:

returns

(ordered) dict

luxpy.utils.**meshblock**(*x*, *y*)

Create a meshed block from x and y.

(Similar to meshgrid, but axis = 0 is retained).

To enable fast blockwise calculation.

Args:

x

ndarray with ndim == 2

y

ndarray with ndim == 2

Returns:

X,Y

> 2 ndarrays with ndim == 3
>> X.shape = (x.shape[0],y.shape[0],x.shape[1])
>>
>> Y.shape = (x.shape[0],y.shape[0],y.shape[1])

luxpy.utils.**asplit**(*data*)

> Split data on last axis

> **Args:**

>> **data**

>>> ndarray

> **Returns:**

>> **returns**

>>> ndarray, ndarray, . . .
>>>> (number of returns is equal data.shape[-1])

luxpy.utils.**ajoin**(*data*)

> Join data on last axis.

> **Args:**

>> **data**

>>> tuple (ndarray, ndarray, . . .)

> **Returns:**

>> **returns**

>>> ndarray (shape[-1] is equal to tuple length)

luxpy.utils.**broadcast_shape**(*data*, *target_shape=None*, *expand_2d_to_3d=None*, *axis0_repeats=None*, *axis1_repeats=None*)

> Broadcasts shapes of data to a target_shape.

> Useful for block/vector calc. when numpy fails to broadcast correctly.

> **Args:**

>> **data**

>>> ndarray

>> **target_shape**

>>> None or tuple with requested shape, optional
>>>> - None: returns unchanged :data:

>> **expand_2d_to_3d**

>>> None (do nothing) or . . . , optional
>>>
>>> If ndim == 2, expand from 2 to 3 dimensions

>> **axis0_repeats**

>>> None or number of times to repeat axis=0, optional
>>>> - None: keep axis=0 same size

>> **axis1_repeats**

None or number of times to repeat axis=1, optional
- None: keep axis=1 same size

Returns:

returns

reshaped ndarray

luxpy.utils.**todim**(*x*, *tshape*, *add_axis=1*, *equal_shape=False*)

Expand x to dims that are broadcast-compatable with shape of another array.

Args:

x

ndarray

tshape

tuple with target shape

add_axis

1, optional
Determines where in x.shape an axis should be added

equal_shape

False or True, optional
True: expand :x: to identical dimensions (speficied by :tshape:)

Returns:

returns

ndarray broadcast-compatable with tshape.

luxpy.utils.**write_to_excel**(*filename*, *df*, *sheet_name='Sheet1'*, *startrow=None*, *truncate_sheet=False*, ***to_excel_kwargs*)

Writes a DataFrame to an existing Excel file into a specified sheet. | If [filename] doesn't exist, then this function will create it.

Args:

filename

File path or existing ExcelWriter
(Example: '/path/to/file.xlsx')

df

dataframe to save to workbook

sheet_name

Name of sheet which will contain DataFrame.
(default: 'Sheet1')

startrow

upper left cell row to dump data frame.
Per default (startrow=None) calculate the last row
in the existing DF and write to the next row. . .

truncate_sheet

> truncate (remove and recreate) [sheet_name]
> before writing DataFrame to Excel file

> **to_excel_kwargs**

> > arguments which will be passed to *DataFrame.to_excel()*
> > [can be dictionary]

Returns: None

Notes: Copied from https://stackoverflow.com/questions/20219254/how-to-write-to-an-existing-excel-file-without-overwriting-d

luxpy.utils.**show_luxpy_tree**(*omit=['.pyc', '__pycache__', '.txt', '.dat', '.csv', '.npz', '.png', '.jpg', '.md', '.pdf', '.ini', '.log', '.rar', 'drivers', 'SDK_', 'dll', 'bak']*)

Show luxpy foler tree.

Args:

> **omit**

> > List of folders and file-extensions to omit.

Returns: None

luxpy.utils.**is_importable**(*string*, *try_pip_install=False*)

Check if string is importable/loadable. If it doesn't then try to 'pip install' it using subprocess. Returns None if succesful, otherwise throws and error or outputs False.

Args:

> **string**

> > string with package or module name

> **try_pip_install**

> > False, optional
> > True: try pip installing it using subprocess

Returns:

> **success**

> > True if importable, False if not.

luxpy.utils.**get_function_kwargs**(*f*)

Get dictionary of a function's keyword arguments and their default values.

Args:

> **f**

> > function name

Returns:

> **dict**

> > Dict with the function's keyword arguments and their default values
> > Is empty if there are no defaults (i.e. f.__defaults__ or f.__kwdefaults__ are None).

luxpy.utils.**profile_fcn**(*fcn*, *profile=True*, *sort_stats='tottime'*, *output_file=None*)

Profile or time a function fcn.

Args:

fcn

function to be profiled or timed (using time.time() difference)

profile

True, optional
Profile the function, otherwise only time it.

sort_stats

'tottime', optional
Sort profile results according to sort_stats ('tottime', 'cumtime',...)

output_file

None, optional
If not None: output result to output_file.

Return:

ps

Profiler output

luxpy.utils.**unique**(*array*, *sort=True*)

Get unique elements from array.

Args:

array

array to get unique elements from.

sort

True, optional
If True: get sorted unique elements.

Returns:

unique_array

ndarray with (sorted) unique elements.

luxpy.utils.**save_pkl**(*filename*, *obj*)

Save an object in a pickle file.

Args:

filename

str with filename of pickle file.

obj

python object to save

Returns:

None

luxpy.utils.**load_pkl**(*filename*)

Load the object in a pickle file.

Args:

filename

> > str with filename of pickle file.

> **Returns:**

> > **obj**

> > loaded python object

`luxpy.utils.`**`tree`**`(dir_path: pathlib.Path, level: int = -1, limit_to_directories: bool = False, length_limit: int = 1000, omit=[])`
Given a directory Path object print a visual tree structure

> **References:**

> > 1. https://stackoverflow.com/questions/9727673/list-directory-tree-structure-in-python

4.2 Math sub-package

> **py**

> > - __init__.py
> > - basics.py
> > - minimizebnd.py
> > - mupolymodel.py
> > - Pyswarms_particleswarm.py
> > - pymoo_nsga_ii.py

> **namespace** luxpy.math

4.2.1 Module with useful math functions

> **normalize_3x3_matrix()** Normalize 3x3 matrix M to xyz0 – > [1,1,1]

> **line_intersect()**

> > Line intersections of series of two line segments a and b.
> > https://stackoverflow.com/questions/3252194/numpy-and-line-intersections

> **positive_arctan()** Calculates the positive angle (0°-360° or 0 - 2*pi rad.) from x and y.

> **dot23()** Dot product of a 2-d ndarray with a (N x K x L) 3-d ndarray using einsum().

> **check_symmetric()** Checks if A is symmetric.

> **check_posdef()** Checks positive definiteness of a matrix via Cholesky.

> **symmM_to_posdefM()**

> > Converts a symmetric matrix to a positive definite one.
> > Two methods are supported:
> > > * 'make': A Python/Numpy port of Muhammad Asim Mubeen's
> > > > matlab function Spd_Mat.m
> > > > (https://nl.mathworks.com/matlabcentral/fileexchange/
> > > > 45873-positive-definite-matrix)
> > > * 'nearest': A Python/Numpy port of John D'Errico's
> > > > 'nearestSPD' MATLAB code.

(https://stackoverflow.com/questions/43238173/
python-convert-matrix-to-positive-semi-definite)

bvgpdf() Evaluate bivariate Gaussian probability density function (BVGPDF) at (x,y) with
center mu and inverse covariance matric, sigmainv.

mahalanobis2() Evaluate the squared mahalanobis distance with center mu and shape and
orientation determined by sigmainv.

rms() Calculates root-mean-square along axis.

geomean() Calculates geometric mean along axis.

polyarea()

Calculates area of polygon.
(First coordinate should also be last)

erf(), erfinv() erf-function and its inverse, direct import from scipy.special

cart2pol() Converts Cartesian to polar coordinates.

pol2cart() Converts polar to Cartesian coordinates.

cart2spher() Converts Cartesian to spherical coordinates.

spher2cart() Converts spherical to Cartesian coordinates.

magnitude_v() Calculates magnitude of vector.

angle_v1v2() Calculates angle between two vectors.

histogram()

Histogram function that can take as bins either the center
(cfr. matlab hist) or bin-edges.

v_to_cik() Calculate 2x2 '(covariance matrix)^-1' elements cik from v-format ellipse descrip-
tor.

cik_to_v() Calculate v-format ellipse descriptor from 2x2 'covariance matrix'^-1 cik.

minimizebnd() scipy.minimize() that allows contrained parameters on unconstrained meth-
ods(port of Matlab's fminsearchbnd). Starting, lower and upper bounds values can also be
provided as a dict.

DEMO Module for Differential Evolutionary Multi-objective Optimization (DEMO).

vec3 Module for spherical vector coordinates.

fmod() Floating point modulus, e.g.: fmod(theta, np.pi * 2) would keep an angle in [0, 2pi]b

fit_ellipse() Fit an ellipse to supplied data points.

fit_cov_ellipse() Fit an covariance ellipse to supplied data points.

interp1() Perform a 1-dimensional linear interpolation (wrapper around
scipy.interpolate.InterpolatedUnivariateSpline).

ndinterp1() Perform n-dimensional interpolation using Delaunay triangulation.

ndinterp1_scipy() Perform n-dimensional interpolation using Delaunay triangulation (wrap-
per around scipy.interpolate.LinearNDInterpolator)

box_m() Performs a Box M test on covariance matrices.

pitman_morgan() Pitman-Morgan Test for the difference between correlated variances with
paired samples.

mupolymod Module for Multivariate Polynomial Model Optimization (2D, 3D)

NOT IMPORTED in math-namespace (to minimize dependencies)

pyswarms_particleswarm Module with particleswarm() function for global minimization using particle swarms (wrapper around pyswarms.single.GlobalBestPSO))

pymoo_nsga_ii Module with nsga_ii() function for pareto-optimal boundary minimization using Non-Dominated-Sort-Genetic-Algorithm NSGA-II (wrapper around pymoo.NSGAII))

---

luxpy.math.**normalize_3x3_matrix**(*M*, *xyz0=array([[1.0000e+00, 1.0000e+00, 1.0000e+00]])*)
  Normalize 3x3 matrix M to xyz0 – > [1,1,1]

  If M.shape == (1,9): M is reshaped to (3,3)

  Args:

  M

  ndarray((3,3) or ndarray((1,9))

  xyz0

  2darray, optional

  Returns:

  returns

  normalized matrix such that M*xyz0 = [1,1,1]

luxpy.math.**symmM_to_posdefM**(*A=None*, *atol=1e-09*, *rtol=1e-09*, *method='make'*, *forcesymm=True*)
  Convert a symmetric matrix to a positive definite one.

  Args:

  A

  ndarray

  atol

  float, optional
  The absolute tolerance parameter (see Notes of numpy.allclose())

  rtol

  float, optional
  The relative tolerance parameter (see Notes of numpy.allclose())

  method

  'make' or 'nearest', optional (see notes for more info)

  forcesymm

  True or False, optional
  If A is not symmetric, force symmetry using:

> A = numpy.triu(A) + numpy.triu(A).T - numpy.diag(numpy.diag(A))

Returns:

> returns
>
>> ndarray with positive-definite matrix.

Notes on supported methods: 1. 'make': A Python/Numpy port of Muhammad Asim Mubeen's matlab function Spd_Mat.m 2. *'nearest': A Python/Numpy port of John D'Errico's 'nearestSPD MATLAB* code. <https://stackoverflow.com/questions/43238173/python-convert-matrix-to-positive-semi-definite>'_

luxpy.math.**check_symmetric**(*A*, *atol=1e-09*, *rtol=1e-09*)

> Check if A is symmetric.

Args:

> **A**
>
>> ndarray
>
> **atol**
>
>> float, optional
>> The absolute tolerance parameter (see Notes of numpy.allclose())
>
> **rtol**
>
>> float, optional
>> The relative tolerance parameter (see Notes of numpy.allclose())

Returns:

> **returns**
>
>> Bool
>> True: the array is symmetric within the given tolerance

luxpy.math.**check_posdef**(*A*, *atol=1e-09*, *rtol=1e-09*)

> Checks positive definiteness of a matrix via Cholesky.

Args:

> **A**
>
>> ndarray
>
> **atol**
>
>> float, optional
>> The absolute tolerance parameter (see Notes of numpy.allclose())
>
> **rtol**
>
>> float, optional
>> The relative tolerance parameter (see Notes of numpy.allclose())

Returns:

> **returns**
>
>> Bool
>> True: the array is positive-definite within the given tolerance

luxpy.math.**positive_arctan**(*x*, *y*, *htype='deg'*)

> Calculate positive angle (0°-360° or 0 - 2*pi rad.) from x and y.

Args:

> **x**
>
> > ndarray of x-coordinates
>
> **y**
>
> > ndarray of y-coordinates
>
> **htype**
>
> > 'deg' or 'rad', optional
> >
> > > - 'deg': hue angle between 0° and 360°
> > > - 'rad': hue angle between 0 and 2pi radians

Returns:

> **returns**
>
> > ndarray of positive angles.

luxpy.math.**line_intersect**(*a1*, *a2*, *b1*, *b2*)

> Line intersections of series of two line segments a and b.
>
> **Args:**
>
> > **a1**
> >
> > > ndarray (.shape = (N,2)) specifying end-point 1 of line a
> >
> > **a2**
> >
> > > ndarray (.shape = (N,2)) specifying end-point 2 of line a
> >
> > **b1**
> >
> > > ndarray (.shape = (N,2)) specifying end-point 1 of line b
> >
> > **b2**
> >
> > > ndarray (.shape = (N,2)) specifying end-point 2 of line b
>
> **Note:** N is the number of line segments a and b.
>
> **Returns:**
>
> > **returns**
> >
> > > ndarray with line-intersections (.shape = (N,2))
>
> **References:**
>
> > 1. https://stackoverflow.com/questions/3252194/numpy-and-line-intersections

luxpy.math.**histogram**(*a*, *bins=10*, *bin_center=False*, *range=None*, *normed=False*, *weights=None*, *density=None*)

> Histogram function that can take as bins either the center (cfr. matlab hist) or bin-edges.
>
> **Args:**
>
> > **bin_center**
> >
> > > False, optional
> > >
> > > False: if :bins: int, str or sequence of scalars:
> > >
> > > > default to numpy.histogram (uses bin edges).
> > >
> > > True: if :bins: is a sequence of scalars:
> > >
> > > > bins (containing centers) are transformed to edges

and nump.histogram is run.

Mimicks matlab hist (uses bin centers).

Note: For other armuments and output, see ?numpy.histogram

Returns:

returns

ndarray with histogram

luxpy.math.**pol2cart**(*theta*, *r=None*, *htype='deg'*)

Convert Cartesion to polar coordinates.

Args:

theta

float or ndarray with theta-coordinates

r

None or float or ndarray with r-coordinates, optional
If None, r-coordinates are assumed to be in :theta:.

htype

'deg' or 'rad, optional
Intput type of :theta:.

Returns:

returns

(float or ndarray of x, float or ndarray of y) coordinates

luxpy.math.**cart2pol**(*x*, *y=None*, *htype='deg'*)

Convert Cartesion to polar coordinates.

Args:

x

float or ndarray with x-coordinates

y

None or float or ndarray with x-coordinates, optional
If None, y-coordinates are assumed to be in :x:.

htype

'deg' or 'rad, optional
Output type of theta.

Returns:

returns

(float or ndarray of theta, float or ndarray of r) values

luxpy.math.**spher2cart**(*theta*, *phi*, *r=1.0*, *deg=True*)

Convert spherical to cartesian coordinates.

Args:

theta

> > Float, int or ndarray
> > Angle with positive z-axis.
>
> > **phi**
> >
> > > Float, int or ndarray
> > > Angle around positive z-axis starting from x-axis.
>
> > **r**
> >
> > > 1, optional
> > > Float, int or ndarray
> > > radius
>
> Returns:
>
> > **x, y, z**
> >
> > > tuple of floats, ints or ndarrays
> > > Cartesian coordinates

luxpy.math.**cart2spher**(*x*, *y*, *z*, *deg=True*)
> Convert cartesian to spherical coordinates.
>
> Args:
>
> > **x, y, z**
> >
> > > tuple of floats, ints or ndarrays
> > > Cartesian coordinates
>
> Returns:
>
> > **theta**
> >
> > > Float, int or ndarray
> > > Angle with positive z-axis.
> >
> > **phi**
> >
> > > Float, int or ndarray
> > > Angle around positive z-axis starting from x-axis.
> >
> > **r**
> >
> > > 1, optional
> > > Float, int or ndarray
> > > radius

luxpy.math.**bvgpdf**(*x*, *y=None*, *mu=None*, *sigmainv=None*)
> Evaluate bivariate Gaussian probability density function (BVGPDF)
>
> Args:
>
> > **x**
> >
> > > scalar or list or ndarray (.ndim = 1 or 2) with
> > > x(y)-coordinates at which to evaluate bivariate Gaussian PD.
> >
> > **y**
> >
> > > None or scalar or list or ndarray (.ndim = 1) with
> > > y-coordinates at which to evaluate bivariate Gaussian PD, optional.
> > > If :y: is None, :x: should be a 2d array.

mu

None or ndarray (.ndim = 2) with center coordinates of
bivariate Gaussian PD, optional.
None defaults to ndarray([0,0]).

sigmainv

None or ndarray with 'inverse covariance matrix', optional
Determines the shape and orientation of the PD.
None default to numpy.eye(2).

Returns:

returns

ndarray with magnitude of BVGPDF(x,y)

luxpy.math.**mahalanobis2**(*x*, *y=None*, *z=None*, *mu=None*, *sigmainv=None*)
Evaluate the squared mahalanobis distance

Args:

x

scalar or list or ndarray (.ndim = 1 or 2) with x(y)-coordinates at which to evaluate the
mahalanobis distance squared.

y

None or scalar or list or ndarray (.ndim = 1) with y-coordinates at which to evaluate the
mahalanobis distance squared, optional.
If :y: is None, :x: should be a 2d array.

z

None or scalar or list or ndarray (.ndim = 1) with z-coordinates at which to evaluate the
mahalanobis distance squared, optional.
If :z: is None & :y: is None, then :x: should be a 2d array.

mu

None or ndarray (.ndim = 1) with center coordinates of the mahalanobis ellipse, optional.
None defaults to zeros(2) or zeros(3).

sigmainv

None or ndarray with 'inverse covariance matrix', optional
Determines the shape and orientation of the PD.
None default to np.eye(2) or eye(3).

Returns:

returns

ndarray with magnitude of mahalanobis2(x,y[,z])

luxpy.math.**dot23**(*A*, *B*, *keepdims=False*)
Dot product of a 2-d ndarray with a (N x K x L) 3-d ndarray using einsum().

Args:

A

ndarray (.shape = (M,N))

> **B**
>
> > ndarray (.shape = (N,K,L))
>
> **Returns:**
>
> > **returns**
> >
> > > ndarray (.shape = (M,K,L))

luxpy.math.**rms**(*data*, *axis=0*, *keepdims=False*)

> Calculate root-mean-square along axis.
>
> **Args:**
>
> > **data**
> >
> > > list of values or ndarray
> >
> > **axis**
> >
> > > 0, optional
> > > Axis along which to calculate rms.
> >
> > **keepdims**
> >
> > > False or True, optional
> > > Keep original dimensions of array.
>
> **Returns:**
>
> > **returns**
> >
> > > ndarray with rms values.

luxpy.math.**geomean**(*data*, *axis=0*, *keepdims=False*)

> Calculate geometric mean along axis.
>
> **Args:**
>
> > **data**
> >
> > > list of values or ndarray
> >
> > **axis**
> >
> > > 0, optional
> > > Axis along which to calculate geomean.
> >
> > **keepdims**
> >
> > > False or True, optional
> > > Keep original dimensions of array.
>
> **Returns:**
>
> > **returns**
> >
> > > ndarray with geomean values.

luxpy.math.**polyarea**(*x*, *y*)

> Calculates area of polygon.
>
> First coordinate should also be last.

Args:

> **x**
>
> > ndarray of x-coordinates of polygon vertices.
>
> **y**
>
> > ndarray of x-coordinates of polygon vertices.

Returns:

> **returns**
>
> > float (area or polygon)

luxpy.math.**magnitude_v**(*v*)

> Calculates magnitude of vector.
>
> **Args:**
>
> > **v**
> >
> > > ndarray with vector
>
> **Returns:**
>
> > **magnitude**
> >
> > > ndarray

luxpy.math.**angle_v1v2**(*v1*, *v2*, *htype='deg'*)

> Calculates angle between two vectors.
>
> **Args:**
>
> > **v1**
> >
> > > ndarray with vector 1
> >
> > **v2**
> >
> > > ndarray with vector 2
> >
> > **htype**
> >
> > > 'deg' or 'rad', optional
> > > Requested angle type.
>
> **Returns:**
>
> > **ang**
> >
> > > ndarray

luxpy.math.**v_to_cik**(*v*, *inverse=False*)

> Calculate 2x2 '(covariance matrix)^-1' elements cik
>
> **Args:**
>
> > **v**
> >
> > > (Nx5) np.ndarray
> > > ellipse parameters [Rmax,Rmin,xc,yc,theta]
> >
> > **inverse**
> >
> > > If True: return inverse of cik.

Returns:

> **cik**
>
> > 'Nx2x2' (covariance matrix)^-1

Notes:

> cik is not actually a covariance matrix,
> only for a Gaussian or normal distribution!

luxpy.math.**cik_to_v**(*cik*, *xyc=None*, *inverse=False*)

> Calculate v-format ellipse descriptor from 2x2 'covariance matrix'^-1 cik

Args:

> **cik**
>
> > 'Nx2x2' (covariance matrix)^-1
>
> **inverse**
>
> > If True: input is inverse of cik.

Returns:

> **v**
>
> > (Nx5) np.ndarray
> > ellipse parameters [Rmax,Rmin,xc,yc,theta]

Notes:

> cik is not actually the inverse covariance matrix,
> only for a Gaussian or normal distribution!

luxpy.math.**fmod**(*x*, *y*)

> Floating point modulus

> e.g., fmod(theta, np.pi * 2) would keep an angle in [0, 2pi]

Args:

> **x**
>
> > angle to restrict
>
> **y**
>
> > end of interval [0, y] to restrict to

Returns:

> **r** floating point modulus

luxpy.math.**remove_outliers**(*data*, *alpha=0.01*)

> Remove multivariate outliers from data when outside of alpha-level confidence ellipsoid.

Args:

> **data**
>
> > Nxp ndarray with multivariate data (N samples, p variables)
>
> **alpha**

0.01, optional

Significance level of confidence ellipsoid marking the boundary for outliers.

Return:

data

(N-... x p) ndarray with multivariate data; outliers removed.

luxpy.math.**fit_ellipse**(*xy*, *center_on_mean_xy=False*)

Fit an ellipse to supplied data points.

Args:

xy

coordinates of points to fit (Nx2 array)

center_on_mean_xy

False, optional

Center ellipse on mean of xy

(otherwise it might be offset due to solving

the contrained minization problem: aT*S*a, see ref below.)

Returns:

v

vector with ellipse parameters [Rmax,Rmin, xc,yc, theta (rad.)]

Reference: 1. Fitzgibbon, A.W., Pilu, M., and Fischer R.B., Direct least squares fitting of ellipsees, Proc. of the 13th Internation Conference on Pattern Recognition, pp 253–257, Vienna, 1996.

luxpy.math.**fit_cov_ellipse**(*xy*, *alpha=0.05*, *pdf='chi2'*, *SE=False*, *robust=False*, *robust_alpha=0.01*)

Fit covariance ellipse to xy data.

Args:

xy

coordinates of points to fit (Nx2 array)

alpha

0.05, optional

alpha significance level

(e.g alpha = 0.05 for 95% confidence ellipse)

pdf

chi2, optional

- 'chi2': Rescale using Chi2-distribution

- 't': Rescale using Student t-distribution

- 'norm': Rescale using normal-distribution

- None: don't rescale using pdf, use alpha as scalefactor (cfr. alpha* 1SD or alpha * 1SE)

SE

False, optional

If false, fit standard error ellipse at alpha significance level

If true, fit standard deviation ellipse at alpha significance level

robust

False, optional
If True: remove outliers beyond the confidence ellipsoid before calculating
the covariances.

robust_alpha

0.01, optional
Significance level of confidence ellipsoid marking the boundary for outliers.

Returns:

v

vector with ellipse parameters [Rmax,Rmin, xc,yc, theta (rad.)]

luxpy.math.**in_hull**(*p*, *hull*)
Test if points in *p* are in *hull*

Args:

p

NxK coordinates of N points in K dimensions

hull

Either a scipy.spatial.Delaunay object or the MxK array of the
coordinates of M points in K dimensions for which Delaunay
triangulation will be computed

Returns:

bool

boolean ndarray with True for in-gamut and False for out-of-gamut points

luxpy.math.**interp1_sprague5**(*x*, *y*, *xn*, *extrap=(nan, nan)*)
Perform a 1-dimensional 5th order Sprague interpolation.

Args:

x

ndarray with n-dimensional coordinates.

y

ndarray with values at coordinates in x.

xn

ndarray of new coordinates.

extrap

(np.nan, np.nan) or string, optional
If tuple: fill with values in tuple (<x[0],>x[-1])
If string: ('zeros','linear', 'nearest', 'nearest-up', 'zero', 'slinear', 'quadratic', 'cubic',
'previous','next')
for more info on the other options see: scipy.interpolate.interp1d?

Returns:

yn

ndarray with values at new coordinates in xn.

luxpy.math.**interp1**(*X, Y, Xnew, kind='linear', ext='extrapolate', w=None, bbox=[None, None], check_finite=False*)

Perform a 1-dimensional linear interpolation (wrapper around scipy.interpolate.InterpolatedUnivariateSpline).

Args:

> **X**
>
>> ndarray with n-dimensional coordinates (last axis represents dimension)
>
> **Y**
>
>> ndarray with values at coordinates in X
>
> **Xnew**
>
>> ndarray of new coordinates (last axis represents dimension)
>
> **kind**
>
>> str or int, optional
>> if str: kind is 'translated' to an int value for input to
>> scipy.interpolate.InterpolatedUnivariateSpline()
>> supported options for str: 'linear', 'quadratic', 'cubic', 'quartic', 'quintic'
>
> **other args**
>
>> see scipy.interpolate.InterpolatedUnivariateSpline()

Returns:

> **Ynew**
>
>> ndarray with new values at coordinates in Xnew

luxpy.math.**ndinterp1**(*X, Y, Xnew*)

Perform nd-dimensional linear interpolation using Delaunay triangulation.

Args:

> **X**
>
>> ndarray with n-dimensional coordinates (last axis represents dimension).
>
> **Y**
>
>> ndarray with values at coordinates in X.
>
> **Xnew**
>
>> ndarray of new coordinates (last axis represents dimension).
>> When outside of the convex hull of X, then a best estimate is
>> given based on the closest vertices.

Returns:

> **Ynew**
>
>> ndarray with new values at coordinates in Xnew.

luxpy.math.**ndinterp1_scipy**(*X, Y, Xnew, fill_value=nan, rescale=False*)

Perform a n-dimensional linear interpolation (wrapper around scipy.interpolate.LinearNDInterpolator).

Args:

> **X**

> ndarray with n-dimensional coordinates (last axis represents dimension)

Y

> ndarray with values at coordinates in X

Xnew

> ndarray of new coordinates (last axis represents dimension)

fill_value

> float, optional
> Value used to fill in for requested points outside of the
> convex hull of the input points. If not provided, then
> the default is `nan`.

rescale

> bool, optional
> Rescale points to unit cube before performing interpolation.
> This is useful if some of the input dimensions have
> incommensurable units and differ by many orders of magnitude.

Returns:

Ynew

> ndarray with new values at coordinates in Xnew

`luxpy.math.`**`box_m`**(*X*, *ni=None*, *verbosity=0*, *robust=False*, *robust_alpha=0.01*)

Perform Box's M test (p>=2) to check equality of covariance matrices or Bartlett's test (p==1) for equality of variances.

Args:

X

> A number (k groups) or list of 2d-ndarrays (rows: samples, cols: variables) with data.
> or a number of 2d-ndarrays with covariance matrices (supply ni!)

ni

> None, optional
> If None: X contains data, else, X contains covariance matrices.

verbosity

> 0, optional
> If 1: print results.

robust

> False, optional
> If True: remove outliers beyond the confidence ellipsoid before calculating
>   the covariances.

robust_alpha

> 0.01, optional
> Significance level of confidence ellipsoid marking the boundary for outliers.

Returns:

statistic

> > F or chi2 value (see len(dfs))

> **pval**

> > p-value

> **df**

> > degrees of freedom.
> > if len(dfs) == 2: F-test was used.
> > if len(dfs) == 1: chi2 approx. was used.

Notes:

> 1. If p==1: Reduces to Bartlett's test for equal variances.

> 2. If (ni>20).all() & (p<6) & (k<6): then a more appropriate chi2 test is used in a some cases.

luxpy.math.**pitman_morgan**(*X*, *Y*, *verbosity=0*)

> Pitman-Morgan Test for the difference between correlated variances with paired samples.

> **Args:**

> > **X,Y**

> > > ndarrays with data.

> > **verbosity**

> > > 0, optional
> > > If 1: print results.

> **Returns:**

> > **tval**

> > > statistic

> > **pval**

> > > p-value

> > **df**

> > > degree of freedom.

> > **ratio**

> > > variance ratio var1/var2 (with var1 > var2).

> **Note:**

> > 1. Based on Gardner, R.C. (2001). Psychological Statistics Using SPSS for Windows. New Jersey, Prentice Hall.

> > 2. Python port from matlab code by Janne Kauttonen (https://nl.mathworks.com/matlabcentral/fileexchange/67910-pitmanmorgantest-x-y; accessed Sep 26, 2019)

luxpy.math.**stress**(*DE*, *DV*, *axis=0*, *max_scale=100*)

> Calculate STandardize-Residual-Sum-of-Squares (STRESS).

> **Args:**

> > **DE, DV**

> > > ndarrays of data to be compared.

> > **axis**

> > 0, optional
> > axis with samples

> **max_scale**

> > 100, optional
> > Maximum of scale.

> **Returns:**

> > **stress**

> > > nadarray with stress value(s).

> **Reference:** 1. Melgosa, M., García, P. A., Gómez-Robledo, L., Shamey, R., Hinks, D., Cui, G., & Luo, M. R. (2011). Notes on the application of the standardized residual sum of squares index for the assessment of intra- and inter-observer variability in color-difference experiments. Journal of the Optical Society of America A, 28(5), 949–953.

luxpy.math.**stress_F_test**(*stressA*, *stressB*, *N*, *alpha=0.05*)
> Perform F-test on significance of difference between STRESS A and STRESS B.

> **Args:**

> > **stressA, stressB**

> > > ndarray with stress(es) values for A and B

> > **N**

> > > int or ndarray with number of samples used to determine stress values.

> > **alpha**

> > > 0.05, optional
> > > significance level

> **Returns:**

> > **Fstats**

> > > Dictionary with keys:
> > > - 'p': p-values
> > > - 'F': F-values
> > > - 'Fc': critcal values
> > > - 'H': string reporting on significance of A compared to B.

luxpy.math.**mean_distance_weighted**(*x*, *axis=0*, *keepdims=False*, *center_x=False*, *rtol=0.001*, *max_iter=100*, *cnt=0*, *mu=None*, *mu0=0*)
> Recursively calculate distance weighted mean.

> **Args:**

> > **x**

> > > ndarray with data

> > **axis**

> > > dimension along which to take mean

> > **keepdims**

> > > False, optional
> > > If True: keep dimension of original ndarray

center_x

> True, optional
> Center data first.

rtol

> 1e-3, optional
> Relative tolerance on recursive mean values. If two sequential
> mean values differ less than this amount, the recursion stops.

max_iter

> 100, optional
> Maximum amount of recursions. If this number is reached the
> recursion stops, even when rtol is not yet achieved. (to avoid
> getting stuck in an infinite loop when the recursion doesn't converge)

cnt,mu,mu0

> Needed for passing values across recursions to be able to stop them.
> DO NOT CHANGE.

Returns:

mu_dw

> distance weighted mean of the array

luxpy.math.**minimizebnd**(*fun*, *x0*, *args=()*, *method='Nelder-Mead'*, *use_bnd=True*, *bounds=(None, None)*, *options=None*, *x0_vsize=None*, *x0_keys=None*, ***kwargs*)

Minimization function that allows for bounds on any type of method in
SciPy's minimize function by transforming the parameters values
(see Matlab's fminsearchbnd).
Starting values, and lower and upper bounds can also be provided as a dict.

Args:

x0

> parameter starting values
> If x0_keys is None then :x0: is vector else, :x0: is dict and
> x0_size should be provided with length/size of values for each of
> > the keys in :x0: to convert it to a vector.

use_bnd

> True, optional
> False: omits bounds and defaults to regular minimize function.

bounds

> (lower, upper), optional
> Tuple of lists or dicts (x0_keys is None) of lower and upper bounds
> for each of the parameters values.

kwargs

> allows input for other type of arguments (e.g. in OutputFcn)

Note: For other input arguments, see ?scipy.optimize.minimize()

Returns:

>**res**
>
>>dict with minimize() output.
>>Additionally, function value, fval, of solution is also in :res:,
>>as well as a vector or dict (if x0 was dict)
>>with final solutions (res['x'])

4.2.2 vec3/

>**py**
>
>>- __init__.py
>>- vec3.py
>
>**namespace** luxpy.math

4.2.3 DEMO/

>**py**
>
>>- __init__.py
>>- DEMO.py
>>- demo_opt.py
>
>**namespace** luxpy.math

4.3 Spectrum sub-package

>**py**
>
>>- __init__.py
>>- spdx_ietm2714.py
>>- **basics/**
>>   - __init__.py
>>   - cmf.py
>>   - spectral.py
>>   - spectral_databases.py
>
>**namespace** luxpy

4.3.1 spectrum: sub-package supporting basic spectral calculations

spectrum/cmf.py

>**luxpy._CMF**
>
>>Dict with keys 'types' and x

x are dicts with keys 'bar', 'K', 'M'

* luxpy._CMF['types'] = ['1931_2','1964_10',
    '2006_2','2006_10','2015_2','2015_10',
    '1931_2_judd1951','1931_2_juddvos1978',
    '1951_20_scotopic']
* luxpy._CMF[x]['bar'] = numpy array with CMFs for type x
    between 360 nm and 830 nm (has shape: (4,471))
* luxpy._CMF[x]['K'] = Constant converting Watt to lumen for CMF type x.
* luxpy._CMF[x]['M'] = XYZ to LMS conversion matrix for CMF type x.
    Matrix is numpy array with shape: (3,3)
* luxpy._CMF[x]['N'] = XYZ to RGB conversion matrix for CMF type x.
    Matrix is numpy array with shape: (3,3)

Notes:

1. **All functions have been expanded (when necessary) using zeros to a**
   full 360-830 range. This way those wavelengths do not contribute in
   the calculation, AND are not extrapolated using the closest known
   value, as per CIE recommendation.

2. **There is no XYZ to LMS conversion matrices defined for the** 1931
   2° Judd corrected (1951) cmf sets. The Hunt-Pointer-Estevez conver-
   sion matrix of the 1931 2° is therefore used as an approximation!

3. **The XYZ to LMS conversion matrix M for the Judd-Vos XYZ CMFs is the one**
   that converts to the 1979 Smith-Pokorny cone fundamentals.

4. **The XYZ to LMS conversion matrix for the 1964 10° XYZ CMFs is set**
   to the one of the CIE 2006 10° cone fundamentals, as not matrix has
   been officially defined for this CMF set.

4. **The K lm to Watt conversion factors for the Judd and Judd-Vos cmf**
   sets have been set to 683.002 lm/W (same as for standard 1931 2°).

5. **The 1951 scoptopic V' function has been replicated in the 3** xbar,
   ybar, zbar columns to obtain a data format similar to the photopic color
   matching functions. This way V' can be called in exactly the same
   way as other V functions can be called from the X,Y,Z cmf sets. The
   K value has been set to 1700.06 lm/W and the conversion matrix has
   been filled with NaN's.

6. The '2015_x' (with x = 2 or 10) are the same XYZ-CMFs as stored in
   '2006_x'.

7. **_CMF[x]['M'] for x equal to '2006_2' (='2015_2') or '2006_10' (='2015_10') is NOT**
   normalized to illuminant E! These are the original matrices as defined
   by [1] & [2].

8. **_CMF[x]['N'] stores known or calculated conversion matrices from**
   xyz to rgb. If not available, N has been filled with NaNs.

spectrum/spectral.py

_WL3 Default wavelength specification in vector-3 format: numpy.array([start, end, spac-
ing])

_INTERP_TYPES Dict with interpolation types associated with various types of spectral data according to CIE recommendation:

_S_INTERP_TYPE Interpolation type for light source spectral data

_R_INTERP_TYPE Interpolation type for reflective/transmissive spectral data

_C_INTERP_TYPE Interpolation type for CMF and cone-fundamental spectral data

getwlr() Get/construct a wavelength range from a (start, stop, spacing) 3-vector.

getwld() Get wavelength spacing of numpy.ndarray with wavelengths.

spd_normalize() Spectrum normalization (supports: area, max, lambda, radiometric, photometric and quantal energy units).

cie_interp() Interpolate / extrapolate spectral data following standard [CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.]

spd()

> All-in-one function that can:
> > 1. Read spectral data from data file or take input directly as pandas.dataframe or ndarray.
> > 2. Convert spd-like data from ndarray to pandas.dataframe and back.
> > 3. Interpolate spectral data.
> > 4. Normalize spectral data.

xyzbar() Get color matching functions.

vlbar() Get Vlambda function.

vlbar_cie_mesopic() Get CIE mesopic luminous efficiency function Vmesm according to CIE191:2010

get_cie_mesopic_adaptation() Get the mesopic adaptation state according to CIE191:2010

spd_to_xyz() Calculates xyz tristimulus values from spectral data.

spd_to_ler() Calculates Luminous efficacy of radiation (LER) from spectral data.

spd_to_power() Calculate power of spectral data in radiometric, photometric or quantal energy units.

detect_peakwl() Detect peak wavelengths and fwhm of peaks in spectrum spd.

spectrum/spectral_databases.py

_S_PATH Path to light source spectra data.

_R_PATH Path to with spectral reflectance data

_IESTM3015 Database with spectral reflectances related to and light source spectra contained excel calculator of IES TM30-15 publication.

_IESTM3018 Database with spectral reflectances related to and light source spectra contained excel calculator of IES TM30-18 publication.

_IESTM3015_S Database with only light source spectra contained in the IES TM30-15 excel calculator.

_IESTM3018_S Database with only light source spectra contained in the IES TM30-18 excel calculator.

4.3. Spectrum sub-package

_CIE_ILLUMINANTS

> Database with CIE illuminants:
>
> * 'E', 'D65', 'A', 'C',
>
> * 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10', 'F11', 'F12'

_CIE_E, _CIE_D65, _CIE_A, _CIE_C, _CIE_F4 Some CIE illuminants for easy use.

_CRI_RFL

> Database with spectral reflectance functions for various color rendition calculators:
>
> * CIE 13.3-1995 (8, 14 munsell samples)
>
> * CIE 224:2015 (99 set)
>
> * CRI2012 (HL17 & HL1000 spectrally uniform and 210 real samples)
>
> * IES TM30 (99, 4880 sepctrally uniform samples)
>
> * MCRI (10 familiar object set)
>
> * CQS (v7.5 and v9.0 sets)

_MUNSELL Database (dict) with 1269 Munsell spectral reflectance functions and Value (V), Chroma (C), hue (h) and (ab) specifications.

_RFL

> Database (dict) with RFLs, including:
>
> * all those in _CRI_RFL,
>
> * the 1269 Matt Munsell samples (see also _MUNSELL),
>
> * the 24 Macbeth ColorChecker samples,
>
> * the 215 samples proposed by Opstelten, J.J. , 1983, The establishment of a representative set of test colours
>
> > for the specification of the colour rendering properties of light sources, CIE-20th session, Amsterdam.
>
> * the 114120 RFLs from capbone.com/spectral-reflectance-database/

spectrum/illuminants.py

_BB Dict with constants for blackbody radiator calculation constant are (c1, c2, n, na, c, h, k).

_S012_DAYLIGHTPHASE ndarray with CIE S0,S1, S2 curves for daylight phase calculation (linearly interpolated to 1 nm).

_CRI_REF_TYPES Dict with blackbody to daylight transition (mixing) ranges for various types of reference illuminants used in color rendering index calculations.

blackbody() Calculate blackbody radiator spectrum.

_DAYLIGHT_LOCI_PARAMETERS dict with parameters for daylight loci for various CMF sets; used by daylightlocus().

_DAYLIGHT_M12_COEFFS dict with coefficients in weights M1 & M2 for daylight phases for various CMF sets.

get_daylightloci_parameters() Get parameters for the daylight loci functions xD(1000/CCT) and yD(xD); used by daylightlocus().

get_daylightphase_Mi_coeffs() Get coefficients of Mi weights of daylight phase for specific cieobs following Judd et al. (1964).

_get_daylightphase_Mi_values() Get daylight phase coefficients M1, M2 following Judd et al. (1964).

_get_daylightphase_Mi() Get daylight phase coefficients M1, M2 following Judd et al. (1964)

daylightlocus() Calculates daylight chromaticity from cct.

daylightphase() Calculate daylight phase spectrum.

cri_ref()

> **Calculates a reference illuminant spectrum based on cct for color** rendering index calculations.
>
> (**CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.,** cie224:2017, CIE 2017 Colour Fidelity Index for accurate scientific use. (2017), ISBN 978-3-902842-61-9., IES-TM-30-15: Method for Evaluating Light Source Color Rendition. New York, NY: The Illuminating Engineering Society of North America.

spd_to_indoor() Convert spd to indoor variant by multiplying it with the CIE spectral transmission for glass.

spectrum/spdx_iestm2714.py

> **_SPDX_TEMPLATE** template dictionary for SPDX data.
>
> **read_spdx()** Read xml file or convert xml string with spdx data to dictionary.
>
> **write_spdx()** Convert spdx dictionary to xml string (and write to .spdx file)

References

1. CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.

2. CIE, and CIE (2006). Fundamental Chromaticity Diagram with Physiological Axes - Part I.(Vienna: CIE).

3. cie224:2017, CIE 2017 Colour Fidelity Index for accurate scientific use. (2017), ISBN 978-3-902842-61-9.

4. IES-TM-30-15: Method for Evaluating Light Source Color Rendition. New York, NY: The Illuminating Engineering Society of North America.

5. Judd, D. B., MacAdam, D. L., Wyszecki, G., Budde, H. W., Condit, H. R., Henderson, S. T., & Simonds, J. L. (1964). Spectral Distribution of Typical Daylight as a Function of Correlated Color Temperature. J. Opt. Soc. Am., 54(8), 1031–1040. https://doi.org/10.1364/JOSA.54.001031

6. http://www.ies.org/iestm2714

---

luxpy.spectrum.**getwlr**(*wl3=None*)
    Get/construct a wavelength range from a 3-vector (start, stop, spacing).

Args:

> **wl3**
>
> > list[start, stop, spacing], optional
> > (defaults to luxpy._WL3)

Returns:

> **returns**
>
> > ndarray (.shape = (n,)) with n wavelengths ranging from
> > start to stop, with wavelength interval equal to spacing.

luxpy.spectrum.**getwld**(*wl*)

> Get wavelength spacing.

Args:

> **wl**
>
> > ndarray with wavelengths

Returns:

> **returns**
>
> > - float: for equal wavelength spacings
> > - ndarray (.shape = (n,)): for unequal wavelength spacings

luxpy.spectrum.**spd_normalize**(*data*, *norm_type=None*, *norm_f=1*, *wl=True*, *cieobs='1931_2'*)

> Normalize a spectral power distribution (SPD).

Args:

> **data**
>
> > ndarray
>
> **norm_type**
>
> > None, optional
> > > - 'lambda': make lambda in norm_f equal to 1
> > > - 'area': area-normalization times norm_f
> > > - 'max': max-normalization times norm_f
> > > - 'ru': to :norm_f: radiometric units
> > > - 'pu': to :norm_f: photometric units
> > > - 'pusa': to :norm_f: photometric units (with Km corrected
> > > > to standard air, cfr. CIE TN003-2015)
> > > - 'qu': to :norm_f: quantal energy units
>
> **norm_f**
>
> > 1, optional
> > Normalization factor that determines the size of normalization
> > for 'max' and 'area'
> > or which wavelength is normalized to 1 for 'lambda' option.
>
> **wl**
>
> > True or False, optional
> > If True, the first column of data contains wavelengths.
>
> **cieobs**
>
> > _CIEOBS or str, optional
> > Type of cmf set to use for normalization using photometric units
> > (norm_type == 'pu')

Returns:

> **returns**
>
>> ndarray with normalized data.

luxpy.spectrum.**cie_interp**(*data*, *wl_new*, *kind=None*, *sprague5_allowed=False*, *negative_values_allowed=False*, *extrap_values='ext'*, *extrap_kind='linear'*, *extrap_log=False*)

Interpolate / extrapolate spectral data following standard CIE15-2018.

The kind of interpolation depends on the spectrum type defined in :kind:.

Args:

> **data**
>
>> ndarray with spectral data
>> (.shape = (number of spectra + 1, number of original wavelengths))
>
> **wl_new**
>
>> ndarray with new wavelengths
>
> **kind**
>
>> None, optional
>>> - If :kind: is None, return original data.
>>> - If :kind: is a spectrum type (see _INTERP_TYPES), the correct
>>>> interpolation type is automatically chosen
>>>>> (The use of the slow(er) 'sprague5' can be toggled on using
>>>>> :sprague5_allowed:).
>>> - Or :kind: can be any interpolation type supported by
>>>> scipy.interpolate.interp1d (or luxpy.math.interp1 if nan's are present!!)
>>>> or can be 'sprague5' (uses luxpy.math.interp1_sprague5).
>
> **sprague5_allowed**
>
>> False, optional
>> If True: When kind is a spectral data type from _INTERP_TYPES['cubic'],
>>> then a cubic spline interpolation will be used in case of
>>> unequal wavelength spacings, otherwise a 5th order Sprague will be used.
>> If False: always use 'cubic', don't use 'sprague5'.
>>> This is the default, as differences are minimal and
>>> use of the 'sprague5' function is a lot slower!
>
> **negative_values_allowed**
>
>> False, optional
>> If False: negative values are clipped to zero.
>
> **extrap_values**
>
>> 'ext', optional
>> If 'ext': extrapolate using 'linear' ('cie167:2005'), 'quadratic' ('cie15:2018')
>>> 'nearest' ('cie15:2004') recommended or other (e.g. 'cubic') methods.
>> If None: use CIE15:2004 recommended 'nearest value' approach when extrapolating.

> If float or list or ndarray, use those values to fill extrapolated value(s).

> **extrap_kind**
>> 'linear', optional
>> Extrapolation method used when :extrap_values: is set to 'ext'.
>> Options: 'linear' ('cie167:2005'), 'quadratic' ('cie15:2018'),
>>> 'nearest' ('cie15:2004'), 'cubic'
>> CIE15:2018 states that based on a 2017 paper by Wang that 'quadratic' is 'better'.
>> However, no significant difference was found between 'quadratic' and 'linear' methods.
>> Also see note 1 below, for why the CIE67:2005 recommended 'linear' extrapolation
>> is set as the default.

> **extrap_log**
>> False, optional
>> If True: extrap the log of the spectral values
>>> (not CIE recommended but in most cases seems to give a
>>> more realistic estimate, but can sometimes seriously fail,
>>> especially for the 'quadratic' extrapolation case (see note 1)!!!)

Returns:

> **returns**
>> ndarray of interpolated spectral data.
>> (.shape = (number of spectra + 1, number of wavelength in wl_new))

Notes:

> 1. Type of extrapolation: 'quadratic' vs 'linear'; impact of extrapolating log spectral values:
>> Using a 'linear' or 'quadratic' extrapolation, as mentioned in
>> CIE167:2005 and CIE15:2018, resp., can lead to extreme large values
>> when setting :extrap_log: (not CIE recommended) to True.
>> A quick test with the IES TM30 spectra (400 nm - 700 nm, 5 nm spacing)
>> shows that 'linear' is better than 'quadratic' in terms of
>> mean, median and max DEu'v' with the original spectra (380 nm - 780 nm, 5 nm spacing).
>> This conferms the recommendation from CIE167:2005 to use 'linear' extrapolation.
>> Setting :extrap_log: to True reduces the median, but inflates the mean due to some
>> extremely large DEu'v' values. However, the increase in mean and max DEu'v' is much
>> larger for the 'quadratic' case, suggesting that 'linear' extrapolation
>> is likely a more suitable recommendation. When using a 1 nm spacing
>> 'linear' is more similar to 'quadratic' when :extrap_log: is False, otherwise 'linear'
>> remains the 'best'. Hence the choice to use the CIE167:2005 recommended linear extrapolation as
>> default!

luxpy.spectrum.**spd**(*data=None*, *interpolation=None*, *kind='np'*, *wl=None*, *columns=None*, *sep=', '*,
> *header=None*, *datatype='S'*, *norm_type=None*, *norm_f=None*)

> All-in-one function that can:
>> 1. Read spectral data from data file or take input directly as pandas.dataframe or ndarray.
>> 2. Convert spd-like data from ndarray to pandas.dataframe and back.
>> 3. Interpolate spectral data.
>> 4. Normalize spectral data.

Args:

data

- str with path to file containing spectral data
- ndarray with spectral data
- pandas.dataframe with spectral data
(.shape = (number of spectra + 1, number of original wavelengths))

interpolation

None, optional
- None: don't interpolate
- str with interpolation type or spectrum type

kind

str ['np','df'], optional
Determines type(:returns:), np: ndarray, df: pandas.dataframe

wl

None, optional
New wavelength range for interpolation.
Defaults to wavelengths specified by luxpy._WL3.

columns

- None or list[str] of column names for dataframe, optional

header

None or 'infer', optional
- None: no header in file
- 'infer': infer headers from file

sep

',' or ' ' or other char, optional
Column separator in case :data: specifies a data file.

datatype'

'S' (light source) or 'R' (reflectance) or other, optional
Specifies a type of spectral data.
Is used when creating column headers when :column: is None.

norm_type

None, optional
- 'lambda': make lambda in norm_f equal to 1
- 'area': area-normalization times norm_f
- 'max': max-normalization times norm_f
- 'ru': to :norm_f: radiometric units
- 'pu': to :norm_f: photometric units
- 'pusa': to :norm_f: photometric units (with Km corrected
    to standard air, cfr. CIE TN003-2015)
- 'qu': to :norm_f: quantal energy units

norm_f

1, optional

Normalization factor that determines the size of normalization for 'max' and 'area' or which wavelength is normalized to 1 for 'lambda' option.

Returns:

returns

ndarray or pandas.dataframe

with interpolated and/or normalized spectral data.

luxpy.spectrum.**xyzbar**(*cieobs='1931_2'*, *scr='dict'*, *wl_new=None*, *kind='np'*)

Get color matching functions.

Args:

cieobs

luxpy._CIEOBS, optional

Sets the type of color matching functions to load.

scr

'dict' or 'file', optional

Determines whether to load cmfs from file (./data/cmfs/)

or from dict defined in .cmf.py

wl

None, optional

New wavelength range for interpolation.

Defaults to wavelengths specified by luxpy._WL3.

kind

str ['np','df'], optional

Determines type(:returns:), np: ndarray, df: pandas.dataframe

Returns:

returns

ndarray or pandas.dataframe with CMFs

References:

1. [CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.](#)

luxpy.spectrum.**vlbar**(*cieobs='1931_2'*, *scr='dict'*, *wl_new=None*, *kind='np'*, *out=1*)

Get Vlambda functions.

Args:

cieobs

str, optional

Sets the type of Vlambda function to obtain.

scr

'dict' or array, optional

- 'dict': get from ybar from _CMF

- 'array': ndarray in :cieobs:

Determines whether to load cmfs from file (./data/cmfs/)

or from dict defined in .cmf.py

Vlambda is obtained by collecting Ybar.

> **wl**
>
> > None, optional
> > New wavelength range for interpolation.
> > Defaults to wavelengths specified by luxpy._WL3.
>
> **kind**
>
> > str ['np','df'], optional
> > Determines type(:returns:), np: ndarray, df: pandas.dataframe
>
> **out**
>
> > 1 or 2, optional
> > > 1: returns Vlambda
> > > 2: returns (Vlambda, Km)

Returns:

> **returns**
>
> > dataframe or ndarray with Vlambda of type :cieobs:

References:

> 1. CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.

luxpy.spectrum.**vlbar_cie_mesopic**(*m=[1], wl_new=None, kind='np', out=1, Lp=None, Ls=None, SP=None*)
Get CIE mesopic luminous efficiency function Vmesm according to CIE191:2010

Args:

> **m**
>
> > float or list or ndarray with mesopic adaptation coefficients
>
> **wl**
>
> > None, optional
> > New wavelength range for interpolation.
> > Defaults to wavelengths specified by luxpy._WL3.
>
> **out**
>
> > 1 or 2, optional
> > > 1: returns Vmesm
> > > 2: returns (Vmes, Kmesm)
>
> **Lp**
>
> > None, optional
> > float or ndarray with photopic adaptation luminance
> > If not None: use this (and SP or Ls) to calculate the
> > mesopic adaptation coefficient
>
> **Ls**
>
> > None, optional
> > float or ndarray with scotopic adaptation luminance
> > If None: SP must be supplied.
>
> **SP**
>
> > None, optional

> S/P ratio
>
> If None: Ls must be supplied.

Returns:

> **Vmes**
>
> > ndarray with mesopic luminous efficiency function
> >
> > for adaptation coefficient(s) m
>
> **Kmes**
>
> > ndarray with luminous efficacies of 555 nm monochromatic light
> >
> > for for adaptation coefficient(s) m

Reference: 1. CIE 191:2010 Recommended System for Mesopic Photometry Based on Visual Performance. (ISBN 978-3-901906-88-6),

luxpy.spectrum.**get_cie_mesopic_adaptation**(*Lp*, *Ls=None*, *SP=None*)

> Get the mesopic adaptation state according to CIE191:2010

Args:

> **Lp**
>
> > float or ndarray with photopic adaptation luminance
>
> **Ls**
>
> > None, optional
> >
> > float or ndarray with scotopic adaptation luminance
> >
> > If None: SP must be supplied.
>
> **SP**
>
> > None, optional
> >
> > S/P ratio
> >
> > If None: Ls must be supplied.

Returns:

> **Lmes**
>
> > mesopic adaptation luminance
>
> **m**
>
> > mesopic adaptation coefficient

Reference: 1. CIE 191:2010 Recommended System for Mesopic Photometry Based on Visual Performance. (ISBN 978-3-901906-88-6),

luxpy.spectrum.**spd_to_xyz**(*data*, *relative=True*, *rfl=None*, *cieobs='1931_2'*, *K=None*, *out=None*, *cie_std_dev_obs=None*)

> Calculates xyz tristimulus values from spectral data.

Args:

> **data**
>
> > ndarray or pandas.dataframe with spectral data
> >
> > (.shape = (number of spectra + 1, number of wavelengths))
> >
> > Note that :data: is never interpolated, only CMFs and RFLs.
> >
> > This way interpolation errors due to peaky spectra are avoided.
> >
> > Conform CIE15-2018.
>
> **relative**

True or False, optional

Calculate relative XYZ (Yw = 100) or absolute XYZ (Y = Luminance)

rfl

ndarray with spectral reflectance functions.

Will be interpolated if wavelengths do not match those of :data:

cieobs

luxpy._CIEOBS or str, optional

Determines the color matching functions to be used in the

calculation of XYZ.

K

None, optional

e.g. K = 683 lm/W for '1931_2' (relative == False)

or K = 100/sum(spd*dl) (relative == True)

out

None or 1 or 2, optional

Determines number and shape of output. (see :returns:)

cie_std_dev_obs

None or str, optional

- None: don't use CIE Standard Deviate Observer function.

- 'f1': use F1 function.

Returns:

returns

If rfl is None:

If out is None: ndarray of xyz values

(.shape = (data.shape[0],3))

If out == 1: ndarray of xyz values

(.shape = (data.shape[0],3))

If out == 2: (ndarray of xyz, ndarray of xyzw) values

Note that xyz == xyzw, with (.shape = (data.shape[0],3))

If rfl is not None:

If out is None: ndarray of xyz values

(.shape = (rfl.shape[0],data.shape[0],3))

If out == 1: ndarray of xyz values

(.shape = (rfl.shape[0]+1,data.shape[0],3))

The xyzw values of the light source spd are the first set

of values of the first dimension. The following values

along this dimension are the sample (rfl) xyz values.

If out == 2: (ndarray of xyz, ndarray of xyzw) values

with xyz.shape = (rfl.shape[0],data.shape[0],3)

and with xyzw.shape = (data.shape[0],3)

References:

1. CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.

luxpy.spectrum.**spd_to_ler**(*data*, *cieobs='1931_2'*, *K=None*)

> Calculates Luminous efficacy of radiation (LER) from spectral data.
>
> > **Args:**
> >
> > > **data**
> > >
> > > > ndarray or pandas.dataframe with spectral data
> > > >
> > > > (.shape = (number of spectra + 1, number of wavelengths))
> > > >
> > > > Note that :data: is never interpolated, only CMFs and RFLs.
> > > >
> > > > This way interpolation errors due to peaky spectra are avoided.
> > > >
> > > > Conform CIE15-2018.
> > >
> > > **cieobs**
> > >
> > > > luxpy._CIEOBS, optional
> > > >
> > > > Determines the color matching function set used in the
> > > >
> > > > calculation of LER. For cieobs = '1931_2' the ybar CMF curve equals
> > > >
> > > > the CIE 1924 Vlambda curve.
> > >
> > > **K**
> > >
> > > > None, optional
> > > >
> > > > > e.g. K = 683 lm/W for '1931_2'
> >
> > **Returns:**
> >
> > > **ler**
> > >
> > > > ndarray of LER values.
> >
> > **References:**
> >
> > > 1. CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.

luxpy.spectrum.**spd_to_power**(*data*, *ptype='ru'*, *cieobs='1931_2'*)

> Calculate power of spectral data in radiometric, photometric or quantal energy units.
>
> > **Args:**
> >
> > > **data**
> > >
> > > > ndarray with spectral data
> > >
> > > **ptype**
> > >
> > > > 'ru' or str, optional
> > > >
> > > > str: - 'ru': in radiometric units
> > > >
> > > > > - 'pu': in photometric units
> > > > >
> > > > > - 'pusa': in photometric units with Km corrected
> > > > >
> > > > > > to standard air (cfr. CIE TN003-2015)
> > > > >
> > > > > - 'qu': in quantal energy units
> > >
> > > **cieobs**
> > >
> > > > _CIEOBS or str, optional
> > > >
> > > > Type of cmf set to use for photometric units.
> >
> > **Returns:**
> >
> > > **returns:**
> > >
> > > > ndarray with normalized spectral data (SI units)

luxpy.spectrum.**detect_peakwl**(*spd*, *n=1*, *verbosity=1*, ***kwargs*)

> Detect primary peak wavelengths and fwhm in spectrum spd.
>
> > **Args:**
> >
> > > **spd**

ndarray with spectral data (2xN).

First row should be wavelengths.

n

1, optional

The number of peaks to try to detect in spd.

verbosity

Make a plot of the detected peaks, their fwhm, etc.

kwargs

Additional input arguments for scipy.signal.find_peaks.

Returns:

prop

list of dictionaries with keys:

- 'peaks_idx' : index of detected peaks

- 'peaks' : peak wavelength values (nm)

- 'heights' : height of peaks

- 'fwhms' : full-width-half-maxima of peaks

- 'fwhms_mid' : wavelength at the middle of the fwhm-range of the peaks (if this is different from the values in 'peaks', then their is some non-symmetry in the peaks)

- 'fwhms_mid_heights' : height at the middle of the peak

luxpy.spectrum.**create_spectral_interpolator**(*S*, *wl=None*, *kind=1*)

Create an interpolator of kind for spectral data S.

Args:

S

Spectral data array

Row 0 should contain wavelengths if :wl: is None.

wl

None, optional

Wavelengths

If wl is None: row 0 of S should contain wavelengths.

kind

1, optional

Order of spline functions used in interpolator (1<=kind<=5)

Interpolator = scipy.interpolate.InterpolatedUnivariateSpline

Returns:

interpolators

List of interpolator functions for each row in S (minus wl-row if present).

Note:

1. Nan's, +infs, -infs will be ignored when generating the interpolators.

luxpy.spectrum.**wls_shift**(*shfts*, *log_shft=False*, *wl=None*, *S=None*, *interpolators=None*, *kind=1*)

Wavelength-shift array S over shft wavelengths.

Args:

shfts

array with wavelength shifts.

> **log_shft**
>
> > False, optional
> > If True: shift in log10 wavelength space.
>
> **wl**
>
> > None, optional
> > Wavelengths to return
> > If wl is None: S will be used and row 0 should contain wavelengths.
>
> **S**
>
> > None, optional
> > Spectral data array.
> > Row 0 should contain wavelengths if :wl: is None.
> > If None: interpolators should be precalculated + wl must contain wavelength array !
>
> **interpolators**
>
> > None, optional
> > Pre-calculated interpolators for the (non-wl) rows in S.
> > If None: will be generated from :S: (which should contain wavelengths on row 0)
> > with specified :kind: using scipy.interpolate.InterpolatedUnivariateSpline
> > If not None and S is not None: interpolators take precedence
>
> **kind**
>
> > 1, optional
> > Order of spline functions used in interpolator (1<=kind<=5)

Returns:

> **wavelength_shifted**
>
> > array with wavelength-shifted S (or interpolators) evaluated at wl.
> > (row 0 contains)

Note:

> 1. Nan's, +infs, -infs will be ignored when generating the interpolators.

luxpy.spectrum.**cri_ref**(*ccts*, *wl3=None*, *ref_type='ciera'*, *mix_range=None*, *cieobs=None*, *norm_type=None*, *norm_f=None*, *force_daylight_below4000K=False*, *n=None*, *daylight_locus=None*)

Calculates a reference illuminant spectrum based on cct for color rendering index calculations .

Args:

> **ccts**
>
> > list of int/floats or ndarray with ccts.
>
> **wl3**
>
> > None, optional
> > New wavelength range for interpolation.
> > Defaults to wavelengths specified by luxpy._WL3.
>
> **ref_type**
>
> > str or list[str], optional
> > Specifies the type of reference spectrum to be calculated.
> > Defaults to luxpy._CRI_REF_TYPE.
> > If :ref_type: is list of strings, then for each cct in :ccts:

a different reference illuminant can be specified.

If :ref_type: == 'spd', then :ccts: is assumed to be an ndarray

of reference illuminant spectra.

mix_range

None or ndarray, optional

Determines the cct range between which the reference illuminant is

a weigthed mean of a Planckian and Daylight Phase spectrum.

Weighthing is done as described in IES TM30:

SPDreference = (Te-T)/(Te-Tb)*Planckian+(T-Tb)/(Te-Tb)*daylight

with Tb and Te are resp. the starting and end CCTs of the

mixing range and whereby the Planckian and Daylight SPDs

have been normalized for equal luminous flux.

If None: use the default specified for :ref_type:.

Can be a ndarray with shape[0] > 1, in which different mixing

ranges will be used for cct in :ccts:.

cieobs

None, optional

Required for the normalization of the Planckian and Daylight SPDs

when calculating a 'mixed' reference illuminant.

Required when calculating daylightphase (adjust locus parameters to cieobs)

If None: _CIEOBS will be used.

norm_type

None, optional

- 'lambda': make lambda in norm_f equal to 1
- 'area': area-normalization times norm_f
- 'max': max-normalization times norm_f
- 'ru': to :norm_f: radiometric units
- 'pu': to :norm_f: photometric units
- 'pusa': to :norm_f: photometric units (with Km corrected
    to standard air, cfr. CIE TN003-2015)
- 'qu': to :norm_f: quantal energy units

norm_f

1, optional

Normalization factor that determines the size of normalization

for 'max' and 'area'

or which wavelength is normalized to 1 for 'lambda' option.

force_daylight_below4000K

False or True, optional

Daylight locus approximation is not defined below 4000 K,

but by setting this to True, the calculation can be forced to

calculate it anyway.

n

None, optional

Refractive index (for use in calculation of blackbody radiators).
If None: use the one stored in _BB['n']

daylight_locus

None, optional
dict with xD(T) and yD(xD) parameters to calculate daylight locus
for specified cieobs.
If None: use pre-calculated values.
If 'calc': calculate them on the fly.

Returns:

returns

ndarray with reference illuminant spectra.
(:returns:[0] contains wavelengths)

Note: Future versions will have the ability to take a dict as input for ref_type. This way other reference
illuminants can be specified than the ones in _CRI_REF_TYPES.

luxpy.spectrum.**blackbody**(*cct*, *wl3=None*, *n=None*, *relative=True*)
Calculate blackbody radiator spectrum for correlated color temperature (cct).
Args:

cct

int or float
(for list of cct values, use cri_ref() with ref_type = 'BB')

wl3

None, optional
New wavelength range for interpolation.
Defaults to wavelengths specified by luxpy._WL3.

n

None, optional
Refractive index.
If None: use the one stored in _BB['n']

relative

False, optional
True: return relative spectrum normalized to 560 nm
False: return absolute spectral radiance (Planck's law; W/(sr.m^2.nm))

Returns:

returns

ndarray with blackbody radiator spectrum
(:returns:[0] contains wavelengths)

References:
1. CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.

luxpy.spectrum.**spd_to_indoor**(*spd*)
Convert spd to indoor variant by multiplying it with the CIE spectral transmission for glass.

luxpy.spectrum.**daylightlocus**(*cct*, *force_daylight_below4000K=False*, *cieobs=None*, *daylight_locus=None*)
Calculates daylight chromaticity (xD,yD) from correlated color temperature (cct).
Args:

cct

int or float or list of int/floats or ndarray

force_daylight_below4000K

> False or True, optional
> Daylight locus approximation is not defined below 4000 K,
> but by setting this to True, the calculation can be forced to
> calculate it anyway.

cieobs

> CMF set corresponding to xD, yD output.
> If None: use default CIE15-20xx locus for '1931_2'
> Else: use the locus specified in :daylight_locus:

daylight_locus

> None, optional
> dict with xD(T) and yD(xD) parameters to calculate daylight locus
> for specified cieobs.
> If None: use pre-calculated values.
> If 'calc': calculate them on the fly.

Returns:

> **(xD, yD)**
>
> > (ndarray of x-coordinates, ndarray of y-coordinates)

References:

> 1. CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.

luxpy.spectrum.**daylightphase**(*cct*, *wl3=None*, *nominal_cct=False*, *force_daylight_below4000K=False*, *verbosity=None*, *n=None*, *cieobs=None*, *daylight_locus=None*, *daylight_Mi_coeffs=None*)

Calculate daylight phase spectrum for correlated color temperature (cct).

Args:

> **cct**
>
> > int or float
> > (for list of cct values, use cri_ref() with ref_type = 'DL')
>
> **wl3**
>
> > None, optional
> > New wavelength range for interpolation.
> > Defaults to wavelengths specified by luxpy._WL3.
>
> **nominal_cct**
>
> > False, optional
> > If cct is nominal (e.g. when calculating D65): multiply cct first
> > by 1.4388/1.4380 to account for change in 'c2' in definition of Planckian.
>
> **cieobs**
>
> > None or str or ndarray, optional
> > CMF set to use when calculating coefficients for daylight locus and for M1, M2
> > weights.
> > If None: use standard coefficients for CIE 1931 2° CMFs (for Si at 10 nm).
> > Else: calculate coefficients following Appendix C of CIE15-2004 and Judd (1964).
>
> **force_daylight_below4000K**

> False or True, optional
> Daylight locus approximation is not defined below 4000 K,
> but by setting this to True, the calculation can be forced to
> calculate it anyway.

verbosity

> None, optional
>> If None: do not print warning when CCT < 4000 K.

n

> None, optional
> Refractive index (for use in calculation of blackbody radiators).
> If None: use the one stored in _BB['n']

daylight_locus

> None, optional
> dict with xD(T) and yD(xD) parameters to calculate daylight locus
> for specified cieobs.
> If None: use pre-calculated values.
> If 'calc': calculate them on the fly.

daylight_Mi_coeffs

> None, optional
> dict with coefficients for M1 & M2 weights for specified cieobs.
> If None: use pre-calculated values.
> If 'calc': calculate them on the fly.

Returns:

returns

> ndarray with daylight phase spectrum
> (:returns:[0] contains wavelengths)

References:

> 1. CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.
> 2. Judd, MacAdam, Wyszecki, Budde, Condit, Henderson, & Simonds (1964). Spectral Distribution of Typical Daylight as a Function of Correlated Color Temperature. J. Opt. Soc. Am., 54(8), 1031–1040.

luxpy.spectrum.**get_daylightloci_parameters**(*ccts=None, cieobs=None, wl3=[300, 830, 10], verbosity=0*)

Get parameters for the daylight loci functions xD(1000/CCT) and yD(xD).

Args:

ccts

> None, optional
> ndarray with CCTs, if None: ccts = np.arange(4000,25000,250)

cieobs

> None or list of str or list of ndarrays, optional
> CMF sets to determine parameters for.
> If None: get for all CMFs sets in _CMF (except scoptopic and deviate observer)

wl3

> [300,830,10], optional
> Wavelength range and spacing of daylight phases to be determined

from '1931_2'. The default setting results in parameters very close
to that in CIE15-2004/2018.

> **verbosity**
>
>> 0, optional
>> print parameters and make plots.

> **Returns:**
>> **dayloci**
>>
>>> dict with parameters for each cieobs
>>> If cieobs contains ndarrays, then keys in dict will be
>>> labeled 'cmf_0', 'cmf_1', . . .

luxpy.spectrum.**get_daylightphase_Mi_coeffs**(*cieobs=None,* *wl3=None,*
S012_daylightphase=None)

> Get coefficients of Mi weights of daylight phase for specific cieobs
> **Args:**
>> **cieobs**
>>
>>> None or str or ndarray or list of str or list of ndarrays, optional
>>> CMF set to get coefficients for.
>>> If None: get coeffs for all CMFs in _CMF
>>
>> **wl3**
>>
>>> None, optional
>>> Wavelength range to interpolate S012_daylightphase to.
>>
>> **S012_daylightphase**
>>
>>> None, optional
>>> Daylight phase component functions.
>>> If None: use _S012_DAYLIGHTPHASE

> **Returns:**
>> **Mcoeffs**
>>
>>> Dictionary with i,j,k,i1,j1,k1,i2,j2,k2 for each cieobs in :cieobs:
>>> If cieobs contains ndarrays, then keys in dict will be
>>> labeled 'cmf_0', 'cmf_1', . . .

luxpy.spectrum.**read_spdx**(*spdx*)

> Read xml file or convert xml string with spdx data to dictionary.
> **Args:**
>> **spdx**
>>
>>> xml string or file with spdx data.

> **Returns:**
>> **spdx_dict**
>>
>>> spdx data in a dictionary.

luxpy.spectrum.**write_spdx**(*spdx_dict*, *filename=None*)

> Convert spdx dictionary to xml string (and write to .spdx file).
> **Args:**
>> **spdx_dict**
>>
>>> dictionary with spdx keys (see _SPDX for keys).
>>
>> **filename**

None, optional

string with filename to write xml data to.

Returns:

> **spdx_xml**
>
> string with xml data in spdx dictionary.

4.3.2 SPD class

> **py**
>
> > • SPD.py
>
> **namespace** luxpy

class luxpy.spectrum.SPD.**SPD**(*spd=None*, *wl=None*, *ax0iswl=True*, *dtype='S'*, *wl_new=None*, *interp_method='auto'*, *negative_values_allowed=False*, *extrap_values='ext'*, *norm_type=None*, *norm_f=1*, *header=None*, *sep=', '*)

> **read_csv_**(*file*, *header=None*, *sep=', '*)
>
> > Reads spectral data from file.
> >
> > **Args:**
> >
> > > **file**
> > >
> > > > filename
> > >
> > > **header**
> > >
> > > > None or 'infer', optional
> > > >
> > > > If 'infer': headers will be inferred from file itself.
> > > >
> > > > If None: no headers are expected from file.
> > >
> > > **sep**
> > >
> > > > ',', optional
> > > >
> > > > Column separator.
> >
> > **Returns:**
> >
> > > **returns**
> > >
> > > > ndarray with spectral data (first row are wavelengths)
> >
> > **Note:** Spectral data in file should be organized in columns with the first column containing the wavelengths.

> **plot**(*ylabel='Spectrum'*, *wavelength_bar=True*, **args*, ***kwargs*)
>
> > Make a plot of the spectral data in SPD instance.
> >
> > **Returns:**
> >
> > > **returns**
> > >
> > > > handle to current axes.

> **mean**()
>
> > Take mean of all spectra in SPD instance.

> **sum**()
>
> > Sum all spectra in SPD instance.

> **dot**(*S*)
>
> > Take dot product with instance of SPD.

---

add(*S*)
    Add instance of SPD.

sub(*S*)
    Subtract instance of SPD.

mul(*S*)
    Multiply by instance of SPD.

div(*S*)
    Divide by instance of SPD.

pow(*n*)
    Raise SPD instance to power n.

get_()
    Get spd as ndarray in instance of SPD.

setwlv(*spd*)
    Store spd ndarray in fields wl and values of instance of SPD.

getwld_()
    Get wavelength spacing of SPD instance.
    Returns:

        returns

          float: for equal wavelength spacings
              ndarray (.shape = (n,)): for unequal wavelength spacings

normalize(*norm_type=None*, *norm_f=1*, *cieobs='1931_2'*)
    Normalize spectral power distributions in SPD instance.
    Args:

        norm_type

        None, optional
            - 'lambda': make lambda in norm_f equal to 1
            - 'area': area-normalization times norm_f
            - 'max': max-normalization times norm_f
            - 'ru': to :norm_f: radiometric units
            - 'pu': to :norm_f: photometric units
            - 'pusa': to :norm_f: photometric units (with Km corrected
                to standard air, cfr. CIE TN003-2015)
            - 'qu': to :norm_f: quantal energy units

        norm_f

        1, optional
        Determines size of normalization for 'max' and 'area' or which wavelength is
        normalized to 1 for 'lambda' option.

        cieobs

        _CIEOBS or str, optional
        Type of cmf set to use for normalization using photometric units (norm_type ==
        'pu')

cie_interp(*wl_new*, *kind='auto'*, *sprague5_allowed=False*, *negative_values_allowed=False*, *extrap_values='ext'*, *extrap_kind='linear'*, *extrap_log=False*)
    Interpolate / extrapolate spectral data following standard CIE15-2018.

The interpolation type depends on the spectrum type defined in :kind:.

Args:

> **wl_new**
>
> > ndarray with new wavelengths
>
> **kind**
>
> > 'auto', optional
> >
> > If :kind: is None, return original data.
> >
> > If :kind: is a spectrum type (see _INTERP_TYPES), the correct
> >
> > > interpolation type if automatically chosen.
> > >
> > > > (The use of the slow(er) 'sprague5' can be toggled on using
> > > > :sprague5_allowed:).
> >
> > If kind = 'auto': use self.dtype
> >
> > Or :kind: can be any interpolation type supported by
> >
> > > scipy.interpolate.interp1d (luxpy.math.interp1 if nan's are present!!)
> > >
> > > or can be 'sprague5' (uses luxpy.math.interp1_sprague5).
>
> **sprague5_allowed**
>
> > False, optional
> >
> > If True: When kind is a spectral data type from _INTERP_TYPES['cubic'],
> >
> > > then a cubic spline interpolation will be used in case of
> > >
> > > unequal wavelength spacings, otherwise a 5th order Sprague will be used.
> >
> > If False: always use 'cubic', don't use 'sprague5'.
> >
> > > This is the default, as differences are minimal and
> > >
> > > use of the 'sprague5' function is a lot slower!
>
> **negative_values_allowed**
>
> > False, optional
> >
> > If False: negative values are clipped to zero
>
> **extrap_values**
>
> > 'ext', optional
> >
> > If 'ext': extrapolate using 'linear' ('cie167:2005' r), 'quadratic' ('cie15:2018')
> >
> > > 'nearest' ('cie15:2004') recommended or other (e.g. 'cubic') methods.
> >
> > If None: use CIE15:2004 recommended 'nearest value' approach when
> > extrapolating.
> >
> > If float or list or ndarray, use those values to fill extrapolated value(s).
>
> **extrap_kind**
>
> > 'linear', optional
> >
> > Extrapolation method used when :extrap_values: is set to 'ext'.
> >
> > Options: 'linear' ('cie167:2005'), 'quadratic' ('cie15:2018'),
> >
> > > 'nearest' ('cie15:2004'), 'cubic'
> >
> > CIE15:2018 states that based on a 2017 paper by Wang that 'quadratic' is
> > 'better'.
> >
> > However, no significant difference was found between 'quadratic' and 'linear'
> > methods.

Also see note 1 below, for why the CIE67:2005 recommended 'linear' extrapolation
is set as the default.

extrap_log

False, optional
If True: extrap the log of the spectral values
> (not CIE recommended but in most cases seems to give a
> more realistic estimate, but can sometimes seriously fail,
> especially for the 'quadratic' extrapolation case (see note 1)!!!)

Returns:

returns

ndarray of interpolated spectral data.
(.shape = (number of spectra+1, number of wavelength in wl_new))

Notes:

1. Type of extrapolation: 'quadratic' vs 'linear'; impact of extrapolating log spectral values:
   > Using a 'linear' or 'quadratic' extrapolation, as mentioned in
   > CIE167:2005 and CIE15:2018, resp., can lead to extreme large values
   > when setting :extrap_log: (not CIE recommended) to True.
   > A quick test with the IES TM30 spectra (400 nm - 700 nm, 5 nm spacing)
   > shows that 'linear' is better than 'quadratic' in terms of
   > mean, median and max DEu'v' with the original spectra (380 nm - 780 nm, 5 nm spacing).
   > This confirms the recommendation from CIE167:2005 to use 'linear' extrapolation.
   > Setting :extrap_log: to True reduces the median, but inflates the mean due to some
   > extremely large DEu'v' values. However, the increase in mean and max DEu'v' is much
   > larger for the 'quadratic' case, suggesting that 'linear' extrapolation
   > is likely a more suitable recommendation. When using a 1 nm spacing
   > 'linear' is more similar to 'quadratic' when :extrap_log: is False, otherwise 'linear'
   > remains the 'best'. Hence the choice to use the CIE167:2005 recommended linear
   > extrapolation as default!

to_xyz (*relative=True*, *rfl=None*, *cieobs='1931_2'*, *out=None*)
Calculates xyz tristimulus values from spectral data and return as instance of XYZ.
Args:

relative

True or False, optional
Calculate relative XYZ (Yw = 100) or absolute XYZ (Y = Luminance)

rfl

ndarray with spectral reflectance functions.
Will be interpolated if wavelengths don't match those of :data:

cieobs

luxpy._CIEOBS, optional
Determines the color matching functions to be used in the calculation of XYZ.

out

None or 1 or 2, optional

Determines number and shape of output. (see :returns:)

Returns:

> **returns**
>
>> luxpy.XYZ instance with ndarray .value field:
>> If rfl is None:
>>> If out is None: ndarray of xyz values
>>>> (.shape = (data.shape[0],3))
>>> If out == 1: ndarray of xyz values
>>>> (.shape = (data.shape[0],3))
>>> If out == 2: (ndarray of xyz , ndarray of xyzw) values
>>>> Note that xyz == xyzw, with (.shape=(data.shape[0],3))
>> If rfl is not None:
>>> If out is None: ndarray of xyz values
>>>> (.shape = (rfl.shape[0],data.shape[0],3))
>>> If out == 1: ndarray of xyz values
>>>> (.shape = (rfl.shape[0]+1,data.shape[0],3))
>>> The xyzw values of the light source spd are the first
>>> set of values of the first dimension.
>>> The following values along this dimension are the
>>> sample (rfl) xyz values.
>>> If out == 2: (ndarray of xyz, ndarray of xyzw) values
>>>> with xyz.shape = (rfl.shape[0],data.shape[0],3)
>>>> and with xyzw.shape = (data.shape[0],3)

References:

> 1. CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018.

4.4 Color sub-package

py

> • __init__.py

namespace luxpy

4.4.1 utils/

py

> • __init__.py
> • plotters.py

namespace luxpy

Module with functions related to plotting of color data

> **get_cmap()** Get an ndarray of rgb values representing a linearly sampled matplotlib colormap

get_subplot_layout() Calculate layout of multiple subplots.

plot_color_data() Plot color data (local helper function)

plotDL() Plot daylight locus.

plotBB() Plot blackbody locus.

plotSL()

>   Plot spectrum locus.
>   (plotBB() and plotDL() are also called, but can be turned off).

plotcerulean()

>   Plot cerulean (yellow (577 nm) - blue (472 nm)) line
>   (Kuehni, CRA, 2014: Table II: spectral lights)
>   Kuehni, R. G. (2014). Unique hues and their stimuli—state of the art. Color Research
>   & Application, 39(3), 279–287.

plotUH()

>   Plot unique hue lines from color space center point xyz0.
>   (Kuehni, CRA, 2014: uY,uB,uG: Table II: spectral lights;
>   uR: Table IV: Xiao data)
>   Kuehni, R. G. (2014). Unique hues and their stimuli—state of the art. Color Research
>   & Application, 39(3), 279–287.

plotcircle() Plot one or more concentric circles.

plotellipse() Plot one or more ellipses.

plot_chromaticity_diagram_colors() Plot the chromaticity diagram colors.

plot_spectrum_colors() Plot spd with spectrum colors.

plot_rfl_color_patches() Create (and plot) an image with colored patches representing a set
of reflectance spectra illuminated by a specified illuminant.

plot_rgb_color_patches() Create (and plot) an image with patches with specified rgb values.

---

luxpy.color.utils.**get_cmap**(*N*, *cmap_name='jet'*)
> Get an ndarray of rgba values representing a linearly sampled matplotlib colormap.
> **Args:**
> > **N**
> >
> > > Number of rgba values in returned cmap.
> >
> > **cmap_name**
> >
> > > 'jet', optional
> > > Matplotlib color map name to sample from.
>
> **Returns:**
> > **cmap**
> >
> > > ndarray with rgba values.

luxpy.color.utils.**get_subplot_layout**(*N*, *min_1xncols=3*)
> Calculate layout of multiple subplots.
> **Args:**
> > **N**

Number of plots.

min_1xncols

Minimum number of columns before splitting over multiple rows.

Returns:

nrows, ncols

`luxpy.color.utils.`**`plotSL`**`(cieobs='1931_2', cspace='Yuv', DL=False, BBL=True, D65=False, EEW=False, cctlabels=False, axh=None, show=True, cspace_pars={}, formatstr='k-', diagram_colors=False, diagram_samples=100, diagram_opacity=1.0, diagram_lightness=0.25, **kwargs)`

Plot spectrum locus for cieobs in cspace.

Args:

DL

True or False, optional
True plots Daylight Locus as well.

BBL

True or False, optional
True plots BlackBody Locus as well.

D65

False or True, optional
True plots D65 chromaticity as well.

EEW

False or True, optional
True plots Equi-Energy-White chromaticity as well.

cctlabels

False or True, optional
Add cct text labels at various points along the blackbody locus.

axh

None or axes handle, optional
Determines axes to plot data in.
None: make new figure.

show

True or False, optional
Invoke matplotlib.pyplot.show() right after plotting

cieobs

luxpy._CIEOBS or str, optional
Determines CMF set to calculate spectrum locus or other.

cspace

luxpy._CSPACE or str, optional
Determines color space / chromaticity diagram to plot data in.
Note that data is expected to be in specified :cspace:

formatstr

'k-' or str, optional
Format str for plotting (see ?matplotlib.pyplot.plot)

cspace_pars

{} or dict, optional
Dict with parameters required by color space specified in :cspace:
(for use with luxpy.colortf())

diagram_colors

False, optional
True: plot colored chromaticity diagram.

diagram_samples

256, optional
Sampling resolution of color space.

diagram_opacity

1.0, optional
Sets opacity of chromaticity diagram

diagram_lightness

0.25, optional
Sets lightness of chromaticity diagram

kwargs

additional keyword arguments for use with matplotlib.pyplot.

Returns:

returns

handle to current axes (:show: == False)

luxpy.color.utils.**plotDL**(*ccts=None*, *cieobs='1931_2'*, *cspace='Yuv'*, *axh=None*, *show=True*, *force_daylight_below4000K=False*, *cspace_pars={}*, *formatstr='k-'*, ***kwargs*)

Plot daylight locus.

Args:

ccts

None or list[float], optional
None defaults to [4000 K to 1e11 K] in 100 steps on a log10 scale.

force_daylight_below4000K

False or True, optional
CIE daylight phases are not defined below 4000 K.
If True plot anyway.

axh

None or axes handle, optional
Determines axes to plot data in.
None: make new figure.

show

True or False, optional
Invoke matplotlib.pyplot.show() right after plotting

cieobs

> luxpy._CIEOBS or str, optional
> Determines CMF set to calculate spectrum locus or other.

cspace

> luxpy._CSPACE or str, optional
> Determines color space / chromaticity diagram to plot data in.
> Note that data is expected to be in specified :cspace:

formatstr

> 'k-' or str, optional
> Format str for plotting (see ?matplotlib.pyplot.plot)

cspace_pars

> {} or dict, optional
> Dict with parameters required by color space specified in :cspace:
> (for use with luxpy.colortf())

kwargs

> additional keyword arguments for use with matplotlib.pyplot.

Returns:

returns

> handle to current axes (:show: == False)

luxpy.color.utils.**plotBB**(*ccts=None, cieobs='1931_2', cspace='Yuv', axh=None, cctlabels=True, show=True, cspace_pars={}, formatstr='k-', **kwargs*)

Plot blackbody locus.

Args:

ccts

> None or list[float], optional
> None defaults to [1000 to 1e19 K].
> Range:
>
> > [1000,1500,2000,2500,3000,3500,4000,5000,6000,8000,10000]
> + [15000 K to 1e11 K] in 100 steps on a log10 scale

cctlabels

> True or False, optional
> Add cct text labels at various points along the blackbody locus.

axh

> None or axes handle, optional
> Determines axes to plot data in.
> None: make new figure.

show

> True or False, optional
> Invoke matplotlib.pyplot.show() right after plotting

cieobs

> luxpy._CIEOBS or str, optional
> Determines CMF set to calculate spectrum locus or other.

cspace

> luxpy._CSPACE or str, optional
> Determines color space / chromaticity diagram to plot data in.
> Note that data is expected to be in specified :cspace:

formatstr

> 'k-' or str, optional
> Format str for plotting (see ?matplotlib.pyplot.plot)

cspace_pars

> {} or dict, optional
> Dict with parameters required by color space specified in :cspace:
> (for use with luxpy.colortf())

kwargs

> additional keyword arguments for use with matplotlib.pyplot.

Returns:

returns

> handle to current axes (:show: == False)

luxpy.color.utils.**plot_color_data**(*x*, *y*, *z=None*, *axh=None*, *show=True*, *cieobs='1931_2'*, *cspace='Yuv'*, *formatstr='k-'*, *legend_loc=None*, ***kwargs*)

> Plot color data from x,y [,z].
>
> **Args:**
>
> **x**
>
> > float or ndarray with x-coordinate data
>
> **y**
>
> > float or ndarray with y-coordinate data
>
> **z**
>
> > None or float or ndarray with Z-coordinate data, optional
> > If None: make 2d plot.
>
> **axh**
>
> > None or axes handle, optional
> > Determines axes to plot data in.
> > None: make new figure.
>
> **show**
>
> > True or False, optional
> > Invoke matplotlib.pyplot.show() right after plotting
>
> **cieobs**
>
> > luxpy._CIEOBS or str, optional
> > Determines CMF set to calculate spectrum locus or other.
>
> **cspace**
>
> > luxpy._CSPACE or str or None, optional
> > Determines color space / chromaticity diagram to plot data in.

Note that data is expected to be in specified :cspace:
If None: don't do any formatting of x,y [z] axes.

formatstr

'k-' or str, optional
Format str for plotting (see ?matplotlib.pyplot.plot)

kwargs

additional keyword arguments for use with matplotlib.pyplot.

Returns:

returns

handle to current axes (:show: == False)

luxpy.color.utils.**plotceruleanline**(*cieobs='1931_2'*,   *cspace='Yuv'*,   *axh=None*,
formatstr='ko-', *cspace_pars={}*)

Plot cerulean (yellow (577 nm) - blue (472 nm)) line

Kuehni, CRA, 2014:
Table II: spectral lights.

Args:

axh

None or axes handle, optional
Determines axes to plot data in.
None: make new figure.

cieobs

luxpy._CIEOBS or str, optional
Determines CMF set to calculate spectrum locus or other.

cspace

luxpy._CSPACE or str, optional
Determines color space / chromaticity diagram to plot data in.
Note that data is expected to be in specified :cspace:

formatstr

'k-' or str, optional
Format str for plotting (see ?matplotlib.pyplot.plot)

cspace_pars

{} or dict, optional
Dict with parameters required by color space specified in :cspace:
(for use with luxpy.colortf())

kwargs

additional keyword arguments for use with matplotlib.pyplot.

Returns:

returns

handle to cerulean line

References: 1. Kuehni, R. G. (2014). Unique hues and their stimuli—state of the art. Color Research & Application, 39(3), 279–287. (see Table II, IV)

luxpy.color.utils.**plotUH**(*xyz0=None, uhues=[0, 1, 2, 3], cieobs='1931_2', cspace='Yuv', axh=None, formatstr=['yo-.', 'bo-.', 'ro-.', 'go-.'], excludefromlegend='', cspace_pars={}*)

Plot unique hue lines from color space center point xyz0.

Kuehni, CRA, 2014:

uY,uB,uG: Table II: spectral lights;

uR: Table IV: Xiao data.

Args:

> **xyz0**
>
> > None, optional
> > Center of color space (unique hue lines are expected to cross here)
> > None defaults to equi-energy-white.
>
> **uhues**
>
> > [0,1,2,3], optional
> > Unique hue lines to plot [0:'yellow',1:'blue',2:'red',3:'green']
>
> **axh**
>
> > None or axes handle, optional
> > Determines axes to plot data in.
> > None: make new figure.
>
> **cieobs**
>
> > luxpy._CIEOBS or str, optional
> > Determines CMF set to calculate spectrum locus or other.
>
> **cspace**
>
> > luxpy._CSPACE or str, optional
> > Determines color space / chromaticity diagram to plot data in.
> > Note that data is expected to be in specified :cspace:
>
> **formatstr**
>
> > ['yo-.','bo-.','ro-.','go-.'] or list[str], optional
> > Format str for plotting the different unique lines
> > (see also ?matplotlib.pyplot.plot)
>
> **excludefromlegend**
>
> > '' or str, optional
> > To exclude certain hues from axes legend.
>
> **cspace_pars**
>
> > {} or dict, optional
> > Dict with parameters required by color space specified in :cspace:
> > (for use with luxpy.colortf())

Returns:

returns

list[handles] to unique hue lines

References: 1. Kuehni, R. G. (2014). Unique hues and their stimuli—state of the art. Color Research & Application, 39(3), 279–287. (see Table II, IV)

luxpy.color.utils.**plotcircle**(*center=array([[0.0000e+00, 0.0000e+00]]), radii=array([0, 10, 20, 30, 40, 50]), angles=array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340]), color='k', linestyle='–', out=None, axh=None, **kwargs*)

Plot one or more concentric circles.

Args:

center

np.array([[0.,0.]]) or ndarray with center coordinates, optional

radii

np.arange(0,60,10) or ndarray with radii of circle(s), optional

angles

np.arange(0,350,10) or ndarray with angles (°), optional

color

'k', optional
Color for plotting.

linestyle

'–', optional
Linestyle of circles.

out

None, optional
If None: plot circles, return (x,y) otherwise.

luxpy.color.utils.**plotellipse**(*v, cspace_in='Yxy', cspace_out=None, nsamples=100, show=True, axh=None, line_color='darkgray', line_style=':', line_width=1, line_marker=", line_markersize=4, plot_center=False, center_marker='o', center_color='darkgray', center_markersize=4, show_grid=False, llabel=", label_fontname='Times New Roman', label_fontsize=12, out=None*)

Plot ellipse(s) given in v-format [Rmax,Rmin,xc,yc,theta].

Args:

v

(Nx5) ndarray
ellipse parameters [Rmax,Rmin,xc,yc,theta]

cspace_in

'Yxy', optional
Color space of v.
If None: no color space assumed. Axis labels assumed ('x','y').

cspace_out

None, optional

Color space to plot ellipse(s) in.
If None: plot in cspace_in.

nsamples

100 or int, optional
Number of points (samples) in ellipse boundary

show

True or boolean, optional
Plot ellipse(s) (True) or not (False)

axh

None, optional
Ax-handle to plot ellipse(s) in.
If None: create new figure with axes.

line_color

'darkgray', optional
Color to plot ellipse(s) in.

line_style

':', optional
Linestyle of ellipse(s).

line_width'

1, optional
Width of ellipse boundary line.

line_marker

'none', optional
Marker for ellipse boundary.

line_markersize

4, optional
Size of markers in ellipse boundary.

plot_center

False, optional
Plot center of ellipse: yes (True) or no (False)

center_color

'darkgray', optional
Color to plot ellipse center in.

center_marker

'o', optional
Marker for ellipse center.

center_markersize

4, optional
Size of marker of ellipse center.

---

show_grid

> False, optional
> Show grid (True) or not (False)

llabel

> None,optional
> Legend label for ellipse boundary.

label_fontname

> 'Times New Roman', optional
> Sets font type of axis labels.

label_fontsize

> 12, optional
> Sets font size of axis labels.

out

> None, optional
> Output of function
> If None: returns None. Can be used to output axh of newly created
>> figure axes or to return Yxys an ndarray with coordinates of
>>> ellipse boundaries in cspace_out (shape = (nsamples,3,N))

Returns:

> **returns** None, or whatever set by :out:.

luxpy.color.utils.**plot_chromaticity_diagram_colors**(*diagram_samples=256,   diagram_opacity=1.0,   diagram_lightness=0.25,   cieobs='1931_2',   cspace='Yxy',   cspace_pars={},   show=True,   axh=None,   show_grid=False,   label_fontname='Times   New   Roman',   label_fontsize=12,   **kwargs*)

Plot the chromaticity diagram colors.

Args:

diagram_samples

> 256, optional
> Sampling resolution of color space.

diagram_opacity

> 1.0, optional
> Sets opacity of chromaticity diagram

diagram_lightness

> 0.25, optional
> Sets lightness of chromaticity diagram

axh

> None or axes handle, optional
> Determines axes to plot data in.
> None: make new figure.

> **show**
>
>> True or False, optional
>>
>> Invoke matplotlib.pyplot.show() right after plotting
>
> **cieobs**
>
>> luxpy._CIEOBS or str, optional
>>
>> Determines CMF set to calculate spectrum locus or other.
>
> **cspace**
>
>> luxpy._CSPACE or str, optional
>>
>> Determines color space / chromaticity diagram to plot data in.
>>
>> Note that data is expected to be in specified :cspace:
>
> **cspace_pars**
>
>> {} or dict, optional
>>
>> Dict with parameters required by color space specified in :cspace:
>>
>> (for use with luxpy.colortf())
>
> **show_grid**
>
>> False, optional
>>
>> Show grid (True) or not (False)
>
> **label_fontname**
>
>> 'Times New Roman', optional
>>
>> Sets font type of axis labels.
>
> **label_fontsize**
>
>> 12, optional
>>
>> Sets font size of axis labels.
>
> **kwargs**
>
>> additional keyword arguments for use with matplotlib.pyplot.

> Returns:

luxpy.color.utils.**plot_spectrum_colors**(*spd=None*, *spdmax=None*, *wavelength_height=-0.05*, *wavelength_opacity=1.0*, *wavelength_lightness=1.0*, *cieobs='1931_2'*, *show=True*, *axh=None*, *show_grid=False*, *ylabel='Spectral intensity (a.u.)'*, *xlim=None*, ***kwargs*)

> Plot the spectrum colors.

> **Args:**
>
>> **spd**
>>
>>> None, optional
>>>
>>> Spectrum
>>
>> **spdmax**
>>
>>> None, optional
>>>
>>> max ylim is set at 1.05 or (1+abs(wavelength_height)*spdmax)
>>
>> **wavelength_opacity**
>>
>>> 1.0, optional

> Sets opacity of wavelength rectangle.

wavelength_lightness

> 1.0, optional
> Sets lightness of wavelength rectangle.

wavelength_height

> -0.05 or 'spd', optional
> Determine wavelength bar height
> if not 'spd': x% of spd.max()

axh

> None or axes handle, optional
> Determines axes to plot data in.
> None: make new figure.

show

> True or False, optional
> Invoke matplotlib.pyplot.show() right after plotting

cieobs

> luxpy._CIEOBS or str, optional
> Determines CMF set to calculate spectrum locus or other.

show_grid

> False, optional
> Show grid (True) or not (False)

ylabel

> 'Spectral intensity (a.u.)' or str, optional
> Set y-axis label.

xlim

> None, optional
> list or ndarray with xlimits.

kwargs

> additional keyword arguments for use with matplotlib.pyplot.

Returns:

luxpy.color.utils.**plot_rfl_color_patches**(*rfl*, *spd=None*, *cieobs='1931_2'*, *patch_shape=(100, 100)*, *patch_layout=None*, *ax=None*, *show=True*)

Create (and plot) an image with colored patches representing a set of reflectance spectra illuminated by a specified illuminant.

Args:

rfl

> ndarray with reflectance spectra

spd

> None, optional
> ndarray with illuminant spectral power distribution

If None: _CIE_D65 is used.

cieobs

'1931_2', optional

CIE standard observer to use when converting rfl to xyz.

patch_shape

(100,100), optional

shape of each of the patches in the image

patch_layout

None, optional

If None: layout is calculated automatically to give a 'good' aspect ratio

ax

None, optional

Axes to plot the image in. If None: a new axes is created.

show

True, optional

If True: plot image in axes and return axes handle; else: return ndarray with image.

Return:

ax or :imagae: | Axes is returned if show == True, else: ndarray with rgb image is returned.

luxpy.color.utils.**plot_rgb_color_patches**(*rgb*, *patch_shape=(100, 100)*, *patch_layout=None*, *ax=None*, *show=True*)

Create (and plot) an image with patches with specified rgb values.

Args:

rgb

ndarray with rgb values for each of the patches

patch_shape

(100,100), optional

shape of each of the patches in the image

patch_layout

None, optional

If None: layout is calculated automatically to give a 'good' aspect ratio

ax

None, optional

Axes to plot the image in. If None: a new axes is created.

show

True, optional

If True: plot image in axes and return axes handle; else: return ndarray with image.

Return:

ax or :imagae: | Axes is returned if show == True, else: ndarray with rgb image is returned.

luxpy.color.utils.**plot_cmfs**(*cmfs, cmf_symbols=['x', 'y', 'z'], cmf_label='', ylabel='Sensitivity', wavelength_bar=True, colors=['r', 'g', 'b'], axh=None, legend=True, **kwargs*)

Plot CMFs.

Args:

> **cmfs**
>
>> ndarray with a set of CMFs.
>
> **cmf_symbols**
>
>> ['x','y','z'], optional
>> Symbols of the CMFs
>> If not a list but a string, the same label will be used for all CMF
>> and the same color will be used ('k' if colors is a list)
>
> **cmf_label**
>
>> '', optional
>> Additional label that will be added in front of the cmf symbols.
>
> **ylabel**
>
>> 'Sensitivity', optional
>> label for y-axis.
>
> **wavelength_bar**
>
>> True, optional
>> Add a colored wavelength bar with spectral colors.
>
> **colors**
>
>> ['r','g','b'], optional
>> Color for plotting each of the individual CMF.
>
> **axh**
>
>> None, optional
>> Axes to plot the image in. If None: a new axes is created.
>
> **kwargs**
>
>> additional kwargs for plt.plot().

Returns:

> **axh**
>
>> figure axes handle.

4.4.2 ctf/

> **py**
>
>> • __init__.py
>>
>> • colortransformations.py
>>
>> • colortf.py
>
> **namespace** luxpy

Module with functions related to basic colorimetry

Note

Note that colorimetric data is always located in the last axis of the data arrays. (See also xyz specification in __doc__ string of luxpy.spd_to_xyz())

colortransforms.py

> **_CSPACE_AXES** dict with list[str,str,str] containing axis labels of defined cspaces
>
> **_IPT_M** Conversion matrix for IPT color space

:_COLORTF_DEFAULT_WHITE_POINT : default white point for colortf (set at Illuminant E)

Supported chromaticity / colorspace functions:

> * xyz_to_Yxy(), Yxy_to_xyz(): (X,Y,Z) <-> (Y,x,y);
> * xyz_to_Yuv(), Yuv_to_Yxy(): (X,Y,Z) <-> CIE 1976 (Y,u',v');
> * xyz_to_Yuv76(), Yuv76_to_Yxy(): (X,Y,Z) <-> CIE 1976 (Y,u',v');
> * xyz_to_Yuv60(), Yuv60_to_Yxy(): (X,Y,Z) <-> CIE 1960 (Y,u,v);
> * xyz_to_xyz(), lms_to_xyz(): (X,Y,Z) <-> (X,Y,Z); for use with colortf()
> * xyz_to_lms(), lms_to_xyz(): (X,Y,Z) <-> (L,M,S) cone fundamental responses
> * xyz_to_lab(), lab_to_xyz(): (X,Y,Z) <-> CIE 1976 (L*a*b*)
> * xyz_to_luv(), luv_to_xyz(): (X,Y,Z) <-> CIE 1976 (L*u*v*)
> * xyz_to_Vrb_mb(),Vrb_mb_to_xyz(): (X,Y,Z) <-> (V,r,b); [Macleod & Boyton, 1979]
> * xyz_to_ipt(), ipt_to_xyz(): (X,Y,Z) <-> (I,P,T); (Ebner et al, 1998)
> * xyz_to_Ydlep(), Ydlep_to_xyz(): (X,Y,Z) <-> (Y,dl, ep);
>> Y, dominant wavelength (dl) and excitation purity (ep)
> * xyz_to_srgb(), srgb_to_xyz(): (X,Y,Z) <-> sRGB; (IEC:61966 sRGB)

References

1. CIE15:2018, "Colorimetry," CIE, Vienna, Austria, 2018. 2. Ebner F, and Fairchild MD (1998). Development and testing of a color space (IPT) with improved hue uniformity. In IS&T 6th Color Imaging Conference, (Scottsdale, Arizona, USA), pp. 8–13. 3. MacLeod DI, and Boynton RM (1979). Chromaticity diagram showing cone excitation by stimuli of equal luminance. J. Opt. Soc. Am. 69, 1183–1186.

luxpy.color.ctf.colortransforms.**xyz_to_Yxy**(*xyz*, ***kwargs*)
> Convert XYZ tristimulus values CIE Yxy chromaticity values.
> **Args:**
>> **xyz**
>>
>>> ndarray with tristimulus values
> **Returns:**
>> **Yxy**
>>
>>> ndarray with Yxy chromaticity values
>>>> (Y value refers to luminance or luminance factor)

luxpy.color.ctf.colortransforms.**Yxy_to_xyz**(*Yxy*, ***kwargs*)
> Convert CIE Yxy chromaticity values to XYZ tristimulus values.
> **Args:**
>> **Yxy**
>>
>>> ndarray with Yxy chromaticity values

> (Y value refers to luminance or luminance factor)

    Returns:

        xyz

            ndarray with tristimulus values

luxpy.color.ctf.colortransforms.**xyz_to_Yuv**(*xyz*, ***kwargs*)

    Convert XYZ tristimulus values CIE 1976 Y,u',v' chromaticity values.

    Args:

        xyz

            ndarray with tristimulus values

    Returns:

        Yuv

            ndarray with CIE 1976 Y,u',v' chromaticity values

                (Y value refers to luminance or luminance factor)

luxpy.color.ctf.colortransforms.**xyz_to_Yuv76**(*xyz*, ***kwargs*)

    Convert XYZ tristimulus values CIE 1976 Y,u',v' chromaticity values.

    Args:

        xyz

            ndarray with tristimulus values

    Returns:

        Yuv

            ndarray with CIE 1976 Y,u',v' chromaticity values

                (Y value refers to luminance or luminance factor)

luxpy.color.ctf.colortransforms.**Yuv_to_xyz**(*Yuv*, ***kwargs*)

    Convert CIE 1976 Y,u',v' chromaticity values to XYZ tristimulus values.

    Args:

        Yuv

            ndarray with CIE 1976 Y,u',v' chromaticity values

                (Y value refers to luminance or luminance factor)

    Returns:

        xyz

            ndarray with tristimulus values

luxpy.color.ctf.colortransforms.**Yuv76_to_xyz**(*Yuv*, ***kwargs*)

    Convert CIE 1976 Y,u',v' chromaticity values to XYZ tristimulus values.

    Args:

        Yuv

            ndarray with CIE 1976 Y,u',v' chromaticity values

                (Y value refers to luminance or luminance factor)

    Returns:

        xyz

            ndarray with tristimulus values

luxpy.color.ctf.colortransforms.**xyz_to_Yuv60**(*xyz*, ***kwargs*)

    Convert XYZ tristimulus values CIE 1960 Y,u,v chromaticity values.

    Args:

        xyz

            ndarray with tristimulus values

    Returns:

> **Yuv**
>
> > > ndarray with CIE 1960 Y,u,v chromaticity values
> > >
> > > > (Y value refers to luminance or luminance factor)

luxpy.color.ctf.colortransforms.**Yuv60_to_xyz**(*Yuv60*, ***kwargs*)

> Convert CIE 1976 Y,u,v chromaticity values to XYZ tristimulus values.
>
> **Args:**
>
> > **Yuv**
> >
> > > ndarray with CIE 1976 Yu'v' chromaticity values
> > >
> > > > (Y value refers to luminance or luminance factor)
>
> **Returns:**
>
> > **xyz**
> >
> > > ndarray with tristimulus values

luxpy.color.ctf.colortransforms.**xyz_to_wuv**(*xyz*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, ***kwargs*)

> Convert XYZ tristimulus values CIE 1964 U*V*W* color space.
>
> **Args:**
>
> > **xyz**
> >
> > > ndarray with tristimulus values
> >
> > **xyzw**
> >
> > > ndarray with tristimulus values of white point, optional
> > >
> > > > (Defaults to luxpy._COLORTF_DEFAULT_WHITE_POINT)
>
> **Returns:**
>
> > **wuv**
> >
> > > ndarray with W*U*V* values

luxpy.color.ctf.colortransforms.**wuv_to_xyz**(*wuv*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, ***kwargs*)

> Convert CIE 1964 U*V*W* color space coordinates to XYZ tristimulus values.
>
> **Args:**
>
> > **wuv**
> >
> > > ndarray with W*U*V* values
> >
> > **xyzw**
> >
> > > ndarray with tristimulus values of white point, optional
> > >
> > > > (Defaults to luxpy._COLORTF_DEFAULT_WHITE_POINT)
>
> **Returns:**
>
> > **xyz**
> >
> > > ndarray with tristimulus values

luxpy.color.ctf.colortransforms.**xyz_to_xyz**(*xyz*, ***kwargs*)

> Convert XYZ tristimulus values to XYZ tristimulus values.
>
> **Args:**
>
> > **xyz**
> >
> > > ndarray with tristimulus values
>
> **Returns:**
>
> > **xyz**
> >
> > > ndarray with tristimulus values

`luxpy.color.ctf.colortransforms.`**`xyz_to_lms`**(*xyz*, *cieobs='1931_2'*, *M=None*, ***kwargs*)

>   Convert XYZ tristimulus values to LMS cone fundamental responses.

>   **Args:**

>>   **xyz**

>>>   ndarray with tristimulus values

>>   **cieobs**

>>>   _CIEOBS or str, optional

>>   **M**

>>>   None, optional
>>>   Conversion matrix for xyz to lms.
>>>>   If None: use the one defined by :cieobs:

>   **Returns:**

>>   **lms**

>>>   ndarray with LMS cone fundamental responses

`luxpy.color.ctf.colortransforms.`**`lms_to_xyz`**(*lms*, *cieobs='1931_2'*, *M=None*, ***kwargs*)

>   Convert LMS cone fundamental responses to XYZ tristimulus values.

>   **Args:**

>>   **lms**

>>>   ndarray with LMS cone fundamental responses

>>   **cieobs**

>>>   _CIEOBS or str, optional

>>   **M**

>>>   None, optional
>>>   Conversion matrix for xyz to lms.
>>>>   If None: use the one defined by :cieobs:

>   **Returns:**

>>   **xyz**

>>>   ndarray with tristimulus values

`luxpy.color.ctf.colortransforms.`**`xyz_to_lab`**(*xyz*,   *xyzw=None*,   *cieobs='1931_2'*, ***kwargs*)

>   Convert XYZ tristimulus values to CIE 1976 L*a*b* (CIELAB) coordinates.

>   **Args:**

>>   **xyz**

>>>   ndarray with tristimulus values

>>   **xyzw**

>>>   None or ndarray with tristimulus values of white point, optional
>>>   None defaults to xyz of CIE D65 using the :cieobs: observer.

>>   **cieobs**

>>>   luxpy._CIEOBS, optional
>>>   CMF set to use when calculating xyzw.

>   **Returns:**

>>   **lab**

>>>   ndarray with CIE 1976 L*a*b* (CIELAB) color coordinates

luxpy.color.ctf.colortransforms.**lab_to_xyz**(*lab*, *xyzw=None*, *cieobs='1931_2'*, ***kwargs*)

Convert CIE 1976 L*a*b* (CIELAB) color coordinates to XYZ tristimulus values.

Args:

> **lab**
>
>> ndarray with CIE 1976 L*a*b* (CIELAB) color coordinates
>
> **xyzw**
>
>> None or ndarray with tristimulus values of white point, optional
>> None defaults to xyz of CIE D65 using the :cieobs: observer.
>
> **cieobs**
>
>> luxpy._CIEOBS, optional
>> CMF set to use when calculating xyzw.

Returns:

> **xyz**
>
>> ndarray with tristimulus values

luxpy.color.ctf.colortransforms.**xyz_to_luv**(*xyz*, *xyzw=None*, *cieobs='1931_2'*, ***kwargs*)

Convert XYZ tristimulus values to CIE 1976 L*u*v* (CIELUV) coordinates.

Args:

> **xyz**
>
>> ndarray with tristimulus values
>
> **xyzw**
>
>> None or ndarray with tristimulus values of white point, optional
>> None defaults to xyz of CIE D65 using the :cieobs: observer.
>
> **cieobs**
>
>> luxpy._CIEOBS, optional
>> CMF set to use when calculating xyzw.

Returns:

> **luv**
>
>> ndarray with CIE 1976 L*u*v* (CIELUV) color coordinates

luxpy.color.ctf.colortransforms.**luv_to_xyz**(*luv*, *xyzw=None*, *cieobs='1931_2'*, ***kwargs*)

Convert CIE 1976 L*u*v* (CIELUVB) coordinates to XYZ tristimulus values.

Args:

> **luv**
>
>> ndarray with CIE 1976 L*u*v* (CIELUV) color coordinates
>
> **xyzw**
>
>> None or ndarray with tristimulus values of white point, optional
>> None defaults to xyz of CIE D65 using the :cieobs: observer.
>
> **cieobs**
>
>> luxpy._CIEOBS, optional
>> CMF set to use when calculating xyzw.

Returns:

> **xyz**

ndarray with tristimulus values

`luxpy.color.ctf.colortransforms.`**`xyz_to_Vrb_mb`**(*xyz, cieobs='1931_2', scaling=[1, 1], M=None, **kwargs*)

Convert XYZ tristimulus values to V,r,b (Macleod-Boynton) color coordinates.

Macleod Boynton: V = R+G, r = R/V, b = B/V
Note that R,G,B ~ L,M,S

Args:

xyz

ndarray with tristimulus values

cieobs

luxpy._CIEOBS, optional
CMF set to use when getting the default M, which is the xyz to lms conversion matrix.

scaling

list of scaling factors for r and b dimensions.

M

None, optional
Conversion matrix for going from XYZ to RGB (LMS)
If None, :cieobs: determines the M (function does inversion)

Returns:

Vrb

ndarray with V,r,b (Macleod-Boynton) color coordinates

Reference:

1. MacLeod DI, and Boynton RM (1979). Chromaticity diagram showing cone excitation by stimuli of equal luminance. J. Opt. Soc. Am. 69, 1183–1186.

`luxpy.color.ctf.colortransforms.`**`Vrb_mb_to_xyz`**(*Vrb, cieobs='1931_2', scaling=[1, 1], M=None, Minverted=False, **kwargs*)

Convert V,r,b (Macleod-Boynton) color coordinates to XYZ tristimulus values.

Macleod Boynton: V = R+G, r = R/V, b = B/V
Note that R,G,B ~ L,M,S

Args:

Vrb

ndarray with V,r,b (Macleod-Boynton) color coordinates

cieobs

luxpy._CIEOBS, optional
CMF set to use when getting the default M, which is
the xyz to lms conversion matrix.

scaling

list of scaling factors for r and b dimensions.

> **M**
>
>> None, optional
>>
>> Conversion matrix for going from XYZ to RGB (LMS)
>>> If None, :cieobs: determines the M (function does inversion)
>
> **Minverted**
>
>> False, optional
>>
>> Bool that determines whether M should be inverted.
>
> **Returns:**
>
>> **xyz**
>>
>>> ndarray with tristimulus values
>
> **Reference:**
>
>> 1. MacLeod DI, and Boynton RM (1979). Chromaticity diagram showing cone excitation by stimuli of equal luminance. J. Opt. Soc. Am. 69, 1183–1186.

luxpy.color.ctf.colortransforms.**xyz_to_ipt**(*xyz*, *cieobs='1931_2'*, *xyzw=None*, *M=None*, ***kwargs*)

> Convert XYZ tristimulus values to IPT color coordinates.

> I: Lightness axis, P, red-green axis, T: yellow-blue axis.

> **Args:**
>
>> **xyz**
>>
>>> ndarray with tristimulus values
>>
>> **xyzw**
>>
>>> None or ndarray with tristimulus values of white point, optional
>>> None defaults to xyz of CIE D65 using the :cieobs: observer.
>>
>> **cieobs**
>>
>>> luxpy._CIEOBS, optional
>>> CMF set to use when calculating xyzw for rescaling M
>>> (only when not None).
>>
>> **M**
>>
>>> None, optional
>>> None defaults to xyz to lms conversion matrix determined by :cieobs:
>
> **Returns:**
>
>> **ipt**
>>
>>> ndarray with IPT color coordinates
>
> **Note:**
>
>> **xyz** is assumed to be under D65 viewing conditions! If necessary perform chromatic adaptation !
>
> **Reference:**
>
>> 1. Ebner F, and Fairchild MD (1998). Development and testing of a color space (IPT) with improved hue uniformity. In IS&T 6th Color Imaging Conference, (Scottsdale, Arizona, USA), pp. 8–13.

luxpy.color.ctf.colortransforms.**ipt_to_xyz**(*ipt*, *cieobs='1931_2'*, *xyzw=None*, *M=None*, ***kwargs*)

> Convert XYZ tristimulus values to IPT color coordinates.

I: Lightness axis, P, red-green axis, T: yellow-blue axis.

Args:

> **ipt**
>
> > ndarray with IPT color coordinates
>
> **xyzw**
>
> > None or ndarray with tristimulus values of white point, optional
> > None defaults to xyz of CIE D65 using the :cieobs: observer.
>
> **cieobs**
>
> > luxpy._CIEOBS, optional
> > CMF set to use when calculating xyzw for rescaling Mxyz2lms
> > (only when not None).
>
> **M**
>
> > None, optional
> > None defaults to xyz to lms conversion matrix determined by:cieobs:

Returns:

> **xyz**
>
> > ndarray with tristimulus values

Note:

> **xyz** is assumed to be under D65 viewing conditions! If necessary perform chromatic adaptation !

Reference:

> 1. Ebner F, and Fairchild MD (1998). Development and testing of a color space (IPT) with improved hue uniformity. In IS&T 6th Color Imaging Conference, (Scottsdale, Arizona, USA), pp. 8–13.

luxpy.color.ctf.colortransforms.**xyz_to_Ydlep**(*xyz,* *cieobs='1931_2',* *xyzw=array([[1.0000e+02,* *1.0000e+02,* *1.0000e+02]]),* *flip_axes=False,* *SL_max_lambda=None, **kwargs*)

Convert XYZ tristimulus values to Y, dominant (complementary) wavelength and excitation purity.

Args:

> **xyz**
>
> > ndarray with tristimulus values
>
> **xyzw**
>
> > None or ndarray with tristimulus values of a single (!) native white point, optional
> > None defaults to xyz of CIE D65 using the :cieobs: observer.
>
> **cieobs**
>
> > luxpy._CIEOBS, optional
> > CMF set to use when calculating spectrum locus coordinates.
>
> **flip_axes**
>
> > False, optional
> > If True: flip axis 0 and axis 1 in Ydelep to increase speed of loop in function.
> > > (single xyzw with is not flipped!)
>
> **SL_max_lambda**

None or float, optional

Maximum wavelength of spectrum locus before it turns back on itelf in the high
wavelength range (~700 nm)

Returns:

Ydlep

ndarray with Y, dominant (complementary) wavelength
and excitation purity

luxpy.color.ctf.colortransforms.**Ydlep_to_xyz**(*Ydlep*, *cieobs='1931_2'*,
xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), *flip_axes=False*,
SL_max_lambda=None, ***kwargs*)

Convert Y, dominant (complementary) wavelength and excitation purity to XYZ tristimulus values.

Args:

Ydlep

ndarray with Y, dominant (complementary) wavelength and excitation purity

xyzw

None or narray with tristimulus values of a single (!) native white point, optional
None defaults to xyz of CIE D65 using the :cieobs: observer.

cieobs

luxpy._CIEOBS, optional
CMF set to use when calculating spectrum locus coordinates.

flip_axes

False, optional
If True: flip axis 0 and axis 1 in Ydelep to increase speed of loop in function.
(single xyzw with is not flipped!)

SL_max_lambda

None or float, optional
Maximum wavelength of spectrum locus before it turns back on itelf in the high
wavelength range (~700 nm)

Returns:

xyz

ndarray with tristimulus values

luxpy.color.ctf.colortransforms.**xyz_to_srgb**(*xyz*, *gamma=2.4*, *offset=-0.055*,
use_linear_part=True, *M=None*, ***kwargs*)

Calculates IEC:61966 sRGB values from xyz.

Args:

xyz

ndarray with relative tristimulus values.

gamma

2.4, optional
Gamma compression in gamma-function gf(x): see notes

offset

-0.055, optional
Offset in gamma-function gf(x): see notes

> **use_linear_part**
>
>> True, optional
>>
>> If False: omit linear part at low RGB values and use gamma function throughout
>
> **M**
>
>> None, optional
>>
>> xyz to linear srgb conversion matrix.
>>
>> If None: use predefined matrix

Returns:

> **rgb**
>
>> ndarray with R,G,B values (uint8).

Notes:

> 1. Gamma-function: gf(x) = ((1-offset)*x**gamma + offset)*255
> 2. dark values use linear function: lf(x) = x[dark] * 12.92 * 255
> 3. To use a pure gamma function, set offset to zero and use_linear_part to False.

`luxpy.color.ctf.colortransforms.`**`srgb_to_xyz`**(*rgb*, *gamma=2.4*, *offset=-0.055*, *use_linear_part=True*, *M=None*, ***kwargs*)

Calculates xyz from IEC:61966 sRGB values.

Args:

> **rgb**
>
>> ndarray with srgb values (uint8).
>
> **gamma**
>
>> 2.4, optional
>>
>> Gamma compression in gamma-function gf(x): see notes
>
> **offset**
>
>> -0.055, optional
>>
>> Offset in gamma-function gf(x): see notes
>
> **use_linear_part**
>
>> True, optional
>>
>> If False: omit linear part at low RGB values and use gamma function throughout
>
> **M**
>
>> None, optional
>>
>> xyz to linear srgb conversion matrix
>>
>> (!!! Don't give inverse matrix as input, function will take inverse of input to M!!!).
>>
>> If None: use predefined inverse matrix

Returns:

> **xyz**
>
>> ndarray with xyz tristimulus values.

Notes:

> 1. Gamma-function: gf(x) = ((1-offset)*x**gamma + offset)*255
> 2. dark values use linear function: lf(x) = x[dark] * 12.92 * 255
> 3. To use a pure gamma function, set offset to zero and use_linear_part to False.

Extension of basic colorimetry module

Global internal variables:

> **_COLORTF_DEFAULT_WHITE_POINT** ndarray with XYZ values of default white
> point (equi-energy white) for color transformation if none is supplied.

Functions:

> **colortf()** Calculates conversion between any two color spaces ('cspace') for which functions
> xyz_to_cspace() and cspace_to_xyz() are defined.

---

luxpy.color.ctf.colortf.**colortf**(*data*, *tf='Yuv'*, *fwtf={}*, *bwtf={}*, ***kwargs*)

> Wrapper function to perform various color transformations.
>
> **Args:**
>> **data**
>>
>>> ndarray
>>
>> **tf**
>>
>>> _CSPACE or str specifying transform type, optional
>>>> E.g. tf = 'spd>xyz' or 'spd>Yuv' or 'Yuv>cct'
>>>>> or 'Yuv' or 'Yxy' or …
>>>> If tf is for example 'Yuv', it is assumed to be a transformation
>>>> of type: 'xyz>Yuv'
>>
>> **fwtf**
>>
>>> dict with parameters (keys) and values required
>>> by some color transformations for the forward transform:
>>>> i.e. 'xyz>…'
>>
>> **bwtf**
>>
>>> dict with parameters (keys) and values required
>>> by some color transformations for the backward transform:
>>>> i.e. '…>xyz'
>
> **Returns:**
>> **returns**
>>
>>> ndarray with data transformed to new color space
>
> **Note:** For the forward transform ('xyz>…'), one can input the keyword arguments specifying the transform
> parameters directly without having to use the dict :fwtf: (should be empty!) [i.e. kwargs overwrites empty
> fwtf dict]

4.4.3 cct/

> **py**
>
>> - __init__.py
>> - cct.py
>> - cct_legacy.py
>> - cctduv_ohno_CORM2011.py
>
> **namespace** luxpy

cct: Module with functions related to correlated color temperature calculations

These methods supersede earlier methods in cct_legacy.y (prior to Nov 2021)

_CCT_MAX (= 1e11 K), max. value that does not cause overflow problems.

_CCT_MIN (= 550 K), min. value that does not cause underflow problems.

_CCT_FALLBACK_N Number of intervals to divide an ndarray with CCTs.

_CCT_FALLBACK_UNIT Type of scale (units) an ndarray will be subdivided.

_CCT_LUT_PATH Folder with Look-Up-Tables (LUT) for correlated color temperature calculations.

_CCT_LUT Dict with pre-calculated LUTs with structure LUT[mode][cspace][cieobs][lut i].

_CCT_LUT_CALC Boolean determining whether to force LUT calculation, even if the LUT.pkl files can be found in ./data/cctluts/.

_CCT_LUT_RESOLUTION_REDUCTION_FACTOR number of subdivisions when performing a cascading lut calculation to zoom-in progressively on the CCT (until a certain tolerance is met)

_CCT_CSPACE default chromaticity space to calculate CCT and Duv in.

_CCT_CSPACE_KWARGS nested dict with cspace parameters for forward and backward modes.

get_tcs4() Get an ndarray of Tc's obtained from a list or tuple of tc4 4-vectors.

calculate_lut() Function that calculates the LUT for the input ccts.

generate_luts() Generate a number of luts and store them in a nested dictionary. (Structure: lut[cspace][cieobs][lut type])

xyz_to_cct() Calculates CCT, Duv from XYZ (wraps a variety of methods)

xyz_to_duv() Calculates Duv, (CCT) from XYZ (wrapper around xyz_to_cct, but with Duv output.)

cct_to_xyz() Calculates xyz from CCT, Duv by estimating the line perpendicular to the planckian locus (=iso-T line).

cct_to_xyz() Calculates xyz from CCT, Duv [_CCT_MIN < CCT < _CCT_MAX]

xyz_to_cct_mcamy1992()

Calculates CCT from XYZ using Mcamy model:

McCamy, Calvin S. (April 1992). Correlated color temperature as an explicit function of chromaticity coordinates. Color Research & Application. 17 (2): 142–144.

xyz_to_cct_hernandez1999()

Calculate CCT from XYZ using Hernández-Andrés et al. model.

Hernández-Andrés, Javier; Lee, RL; Romero, J (September 20, 1999). Calculating Correlated Color Temperatures Across the Entire Gamut of Daylight and Skylight Chromaticities. Applied Optics. 38 (27): 5703–5709. PMID 18324081.

xyz_to_cct_ohno2014()

Calculates CCT, Duv from XYZ using a Ohno's 2014 LUT method.

Ohno Y. (2014) Practical use and calculation of CCT and Duv. Leukos. 2014 Jan 2;10(1):47-55.

xyz_to_cct_zhang2019()

Calculates CCT, Duv from XYZ using Zhang's 2019 golden-ratio search algorithm

Zhang, F. (2019). High-accuracy method for calculating correlated color temperature with a lookup table based on golden section search. Optik, 193, 163018.

xyz_to_cct_robertson1968()

Calculates CCT, Duv from XYZ using a Robertson's 1968 search method.

Robertson, A. R. (1968). Computation of Correlated Color Temperature and Distribution Temperature. Journal of the Optical Society of America, 58(11), 1528–1535.

xyz_to_cct_li2016()

Calculates CCT, Duv from XYZ using a Li's 2019 Newton-Raphson method.

Li, C., Cui, G., Melgosa, M., Ruan, X., Zhang, Y., Ma, L., Xiao, K., & Luo, M. R. (2016). Accurate method for computing correlated color temperature. Optics Express, 24(13), 14066–14078.

xyz_to_cct_fibonacci()

Calculates CCT, Duv from XYZ using a Fibonacci search method.

cct_to_mired() Converts from CCT to Mired scale (or back).

xyz_to_cct_ohno2011() Calculate cct and Duv from CIE 1931 2° xyz following Ohno (CORM 2011).

cct_legacy module with old (pre Nov 2021 cct conversion functions)

---

luxpy.color.cct.**cct_to_mired**(*data*)

Convert cct to Mired scale (or back).

Args:

> **data**
>
> > ndarray with cct or Mired values.

Returns:

> **returns**
>
> > ndarray ((10**6) / data)

luxpy.color.cct.**xyz_to_cct_mcamy1992**(*xyzw*, *cieobs='1931_2'*, *wl=None*, *out='cct'*, *cspace='Yuv60'*, *cspace_kwargs={'bwtf': {}, 'fwtf': {}}*)

Convert XYZ tristimulus values to correlated color temperature (CCT) using the mccamy approximation (!!! only valid for CIE 1931 2° input !!!).

Only valid for approx. 3000 < T < 9000, if < 6500, error < 2 K

Args:

> **xyzw**
>
> > ndarray of tristimulus values
>
> **cieobs**
>
> > '1931_2', optional
> > CMF set used to calculated xyzw.

> Note: since the parameter values in Mcamy's equation were optimized,
>> using the 1931 2° CMFs, this is only valid for that CMF set.
>> It can be changed, but will only impact the calculation of Duv and
>> thereby causing a potential mismatch/error. Change at own discretion.

out

> 'cct' (or 1), optional
> Determines what to return.
> Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)

wl

> None, optional
> Wavelengths used when calculating Planckian radiators when determining Duv.
>> (!!CCT is determined using a fixed set of equations optimized for the 1931 2°
>> CMFS!!)

cspace

> _CCT_SPACE, optional
> Color space to do calculations in.
> Options:
>> - cspace string:
>>> e.g. 'Yuv60' for use with luxpy.colortf()
>> - tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)]
>> functions
>>> (and an optional string describing the cspace):
>>>> e.g. (forward, backward) or (forward, backward, cspace
>>>> string) or (forward, cspace string)
>> - dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str'
>> (cspace string)]
>> Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in
>> the CIE 1976 u'v' diagram

cspace_kwargs

> _CCT_CSPACE_KWARGS, optional
> Parameter nested dictionary for the forward and backward transforms.

Returns:

> **cct**
>
>> ndarray of correlated color temperatures estimates

References: 1. McCamy, Calvin S. (April 1992). "Correlated color temperature as an explicit function of chromaticity coordinates". Color Research & Application. 17 (2): 142–144.

luxpy.color.cct.**xyz_to_cct_hernandez1999**(*xyzw*, *cieobs='1931_2'*, *wl=None*, *out='cct'*, *cspace='Yuv60'*, *cspace_kwargs={'bwtf': {}*, *'fwtf': {}}*)

Convert XYZ tristimulus values to correlated color temperature (CCT) using the mccamy approximation (!!! only valid for CIE 1931 2° input !!!).

According to paper small error from 3000 - 800 000 K

Args:

xyzw

> ndarray of tristimulus values

cieobs

> '1931_2', optional
>
> CMF set used to calculated xyzw.
>
> Note: since the parameter values in the HA equations were optimized,
>> using the 1931 2° CMFs, this is only valid for that CMF set.
>> It can be changed, but will only impact the calculation of Duv and
>> thereby causing a potential mismatch/error. Change at own discretion.

out

> 'cct' (or 1), optional
>
> Determines what to return.
>
> Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)

wl

> None, optional
>
> Wavelengths used when calculating Planckian radiators when determining Duv.
>> (!!CCT is determined using a fixed set of equations optimized for the 1931 2°
>> CMFS!!)

cspace

> _CCT_SPACE, optional
>
> Color space to do calculations in.
>
> Options:
>> - cspace string:
>>> e.g. 'Yuv60' for use with luxpy.colortf()
>> - tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)]
>> functions
>>> (and an optional string describing the cspace):
>>>> e.g. (forward, backward) or (forward, backward, cspace
>>>> string) or (forward, cspace string)
>> - dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str'
>> (cspace string)]
>>
>> Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in
>> the CIE 1976 u'v' diagram

cspace_kwargs

> _CCT_CSPACE_KWARGS, optional
>
> Parameter nested dictionary for the forward and backward transforms.

Returns:

cct

> ndarray of correlated color temperatures estimates

References: 1. Hernández-Andrés, Javier; Lee, RL; Romero, J (September 20, 1999). Calculating Correlated Color Temperatures Across the Entire Gamut of Daylight and Skylight Chromaticities. Applied Optics. 38 (27), 5703–5709. P

luxpy.color.cct.**xyz_to_cct_robertson1968**(*xyzw, cieobs='1931_2', out='cct', is_uv_input=False, wl=None, atol=0.1, rtol=1e-05, force_tolerance=True, tol_method='newton-raphson', lut_resolution_reduction_factor=4, split_calculation_at_N=25, max_iter=10, cspace='Yuv60', cspace_kwargs={'bwtf': {}, 'fwtf': {}}, lut=None, luts_dict=None, ignore_wl_diff=False, use_fast_duv=True, **kwargs*)

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv(distance above (> 0) or below (< 0) the Planckian locus) using Robertson's 1968 search method.

Args:

> **xyzw**
>
>> ndarray of tristimulus values
>
> **cieobs**
>
>> luxpy._CIEOBS, optional
>> CMF set used to calculated xyzw.
>
> **out**
>
>> 'cct' (or 1), optional
>> Determines what to return.
>> Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)
>
> **is_uv_input**
>
>> False, optional
>> If True: xyzw contain uv input data, not xyz data!
>
> **wl**
>
>> None, optional
>> Wavelengths used when calculating Planckian radiators.
>> If None: use same wavelengths as CMFs in :cieobs:.
>
> **rtol**
>
>> 1e-5, float, optional
>> Stop search when cct a relative tolerance is reached.
>> The relative tolerance is calculated as dCCT/CCT_est,
>> with CCT_est the current intermediate estimate in the
>> search and with dCCT the difference between
>> the present and former estimates.
>
> **atol**
>
>> 0.1, optional
>> Stop search when cct a absolute tolerance (K) is reached.
>
> **force_tolerance**
>
>> True, optional
>> If False: search only using the list of CCTs in the used lut.
>>> Only one loop of the full algorithm is performed.
>>> Accuracy depends on CCT of test source and the location
>>> and spacing of the CCTs in the list.

If True: search will use adjacent CCTs to test source to create a new LUT,
>   (repeat the algoritm at higher resolution, progessively zooming in
>   > toward the ground-truth) for tol_method == 'cl'; when
>   tol_method == 'nr' a newton-raphson method is used.
>   Because the CCT for multiple source is calculated in one go,
>   the atol and rtol values have to be met for all!

tol_method

> 'newton-raphson', optional
> (Additional) method to try and achieve set tolerances.
> Options:
> - 'cl', 'cascading-lut': use increasingly higher CCT-resolution
>   > to 'zoom-in' on the ground-truth.
> - 'nr', 'newton-raphson': use the method as described in Li, 2016.

lut_resolution_reduction_factor

> _CCT_LUT_RESOLUTION_REDUCTION_FACTOR, optional
> Number of times the interval spanned by the adjacent Tc in a search or lut
> method is downsampled (the search process will then start again)

max_iter

> _CCT_MAX_ITER, optional
> Maximum number of iterations used by the cascading-lut or newton-raphson methods.

split_calculation_at_N

> _CCT_SPLIT_CALC_AT_N, optional
> Split calculation when xyzw.shape[0] > split_calculation_at_N.
> Splitting speeds up the calculation. If None: no splitting is done.

lut

> None, optional
> Look-Up-Table with Ti, u,v,u',v',u'',v'',slope values of Planckians.
> Options:
>   > - None: defaults to the lut specified in
>   > _CCT_LUT['robertson1968']['lut_type_def'].
>   > - list (lut,lut_kwargs): use this pre-calculated lut
>   >   > (add additional kwargs for the lut_generator_fcn(), defaults to None if
>   >   > omitted)
>   > - tuple: must be key (label) in :luts_dict: (pre-calculated dict of luts),
>   >   > if not: then a new lut will be generated from scratch using the info in the
>   >   > tuple.
>   > - str: must be key (label) in :luts_dict: (pre-calculated dict of luts)
>   > - ndarray [Nx1]: list of luts for which to generate a lut
>   > - ndarray [Nxn] with n>3: pre-calculated lut (last col must contain slope of the
>   > isotemperature lines).

luts_dict

> None, optional
> Dictionary of pre-calculated luts for various cspaces and cmf sets.
>   > Must have structure luts_dict[cspace][cieobs][lut_label] with the

---

> lut part of a two-element list [lut, lut_kwargs]. It must contain
> at the top-level a key 'wl' containing the wavelengths of the
> Planckians used to generate the luts in this dictionary.

If None: luts_dict defaults to _CCT_LUT['robertson1968']['luts']

cspace

> _CCT_SPACE, optional
> Color space to do calculations in.
> Options:
>
> > - cspace string:
> >
> > > e.g. 'Yuv60' for use with luxpy.colortf()
> >
> > - tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)]
> > functions
> >
> > > (and an optional string describing the cspace):
> > >
> > > > e.g. (forward, backward) or (forward, backward, cspace
> > > > string) or (forward, cspace string)
> >
> > - dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str'
> > (cspace string)]
>
> Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in
> the CIE 1976 u'v' diagram

cspace_kwargs

> _CCT_CSPACE_KWARGS, optional
> Parameter nested dictionary for the forward and backward transforms.

ignore_wl_diff

> False, optional
> When getting a lut from the dictionary, if differences are
> detected in the wavelengts of the lut and the ones used to calculate any
> plankcians then a new lut should be generated. Seting this to True ignores
> these differences and proceeds anyway.

use_fast_duv

> _CCT_FAST_DUV, optional
> If True: use a fast estimator of the Duv
>
> > (one that avoids calculation of Planckians and uses the former
> >
> > > best estimate's u,v coordinates. This method is accurate enough
> > > when the atol is small enough -> as long as abs(T-T_former)<=1K
> > > the Duv estimate should be ok.)

Returns:

returns

> ndarray with:
> > cct: out == 'cct' (or 1)
> > duv: out == 'duv' (or -1)
> > cct, duv: out == 'cct,duv' (or 2)
> > [cct,duv]: out == "[cct,duv]" (or -2)

Note: 1. Out-of-lut CCTs are encoded as negative CCTs (with as absolute value the value of the closest CCT
from the lut.)

References: 1. *Robertson, A. R. (1968). Computation of Correlated Color Temperature and Distribution Temperature. Journal of the Optical Society of America, 58(11), 1528–1535. <https://doi.org/10.1364/JOSA.58.001528>*

2. Smet K.A.G., Royer M., Baxter D., Bretschneider E., Esposito E., Houser K., Luedtke W., Man K., Ohno Y. (2022), Recommended method for determining the correlated color temperature and distance from the Planckian Locus of a light source (in preparation, LEUKOS?)

3. Baxter D., Royer M., Smet K.A.G. (2022) Modifications of the Robertson Method for Calculating Correlated Color Temperature to Improve Accuracy and Speed (in preparation, LEUKOS?)

4. Li, C., Cui, G., Melgosa, M., Ruan, X., Zhang, Y., Ma, L., Xiao, K., & Luo, M. R. (2016). Accurate method for computing correlated color temperature. Optics Express, 24(13), 14066–14078.

luxpy.color.cct.**xyz_to_cct_ohno2014**(*xyzw*, *cieobs='1931_2'*, *out='cct'*, *is_uv_input=False*, *wl=None*, *atol=0.1*, *rtol=1e-05*, *force_tolerance=True*, *tol_method='newton-raphson'*, *lut_resolution_reduction_factor=4*, *duv_triangular_threshold=0.002*, *split_calculation_at_N=25*, *max_iter=10*, *cspace='Yuv60'*, *cspace_kwargs={'bwtf': {}, 'fwtf': {}}*, *lut=None*, *luts_dict=None*, *ignore_wl_diff=False*, *use_fast_duv=True*, ***kwargs*)

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv (distance above (>0) or below (<0) the Planckian locus) using Ohno's 2014 method.

Args:

> **xyzw**
>
>> ndarray of tristimulus values
>
> **cieobs**
>
>> luxpy._CIEOBS, optional
>> CMF set used to calculated xyzw.
>
> **out**
>
>> 'cct' (or 1), optional
>> Determines what to return.
>> Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)
>
> **is_uv_input**
>
>> False, optional
>> If True: xyzw contain uv input data, not xyz data!
>
> **wl**
>
>> None, optional
>> Wavelengths used when calculating Planckian radiators.
>> If None: use same wavelengths as CMFs in :cieobs:.
>
> **rtol**
>
>> 1e-5, float, optional
>> Stop search when cct a relative tolerance is reached.
>> The relative tolerance is calculated as dCCT/CCT_est,
>> with CCT_est the current intermediate estimate in the
>> search and with dCCT the difference between
>> the present and former estimates.

atol

> 0.1, optional
> Stop search when cct a absolute tolerance (K) is reached.

force_tolerance

> True, optional
> If False: search only using the list of CCTs in the used lut.
>> Only one loop of the full algorithm is performed.
>> Accuracy depends on CCT of test source and the location
>> and spacing of the CCTs in the list.
> If True: search will use adjacent CCTs to test source to create a new LUT,
>> (repeat the algoritm at higher resolution, progessively zooming in
>>> toward the ground-truth) for tol_method == 'cl'; when
>> tol_method == 'nr' a newton-raphson method is used.
>> Because the CCT for multiple source is calculated in one go,
>> the atol and rtol values have to be met for all!

tol_method

> 'newton-raphson', optional
> (Additional) method to try and achieve set tolerances.
> Options:
> - 'cl', 'cascading-lut': use increasingly higher CCT-resolution
>> to 'zoom-in' on the ground-truth.
> - 'nr', 'newton-raphson': use the method as described in Li, 2016.

lut_resolution_reduction_factor

> _CCT_LUT_RESOLUTION_REDUCTION_FACTOR, optional
> Number of times the interval spanned by the adjacent Tc in a search or lut
> method is downsampled (the search process will then start again)

duv_triangular_threshold

> 0.002, optional
> Threshold for use of the triangular solution.
>> (if smaller use triangular solution, else use the non-triangular one -> 3e-order
>> poly)

max_iter

> _CCT_MAX_ITER, optional
> Maximum number of iterations used by the cascading-lut or newton-raphson methods.

split_calculation_at_N

> _CCT_SPLIT_CALC_AT_N, optional
> Split calculation when xyzw.shape[0] > split_calculation_at_N.
> Splitting speeds up the calculation. If None: no splitting is done.

lut

> None, optional
> Look-Up-Table with Ti, u,v,u',v',u'',v'',slope values of Planckians.
> Options:

- None: defaults to the lut specified in
_CCT_LUT['ohno2014']['lut_type_def'].
- list (lut,lut_kwargs): use this pre-calculated lut
> (add additional kwargs for the lut_generator_fcn(), defaults to None if
> omitted)
- tuple: must be key (label) in :luts_dict: (pre-calculated dict of luts),
> if not: then a new lut will be generated from scratch using the info in the
> tuple.
- str: must be key (label) in :luts_dict: (pre-calculated dict of luts)
- ndarray [Nx1]: list of luts for which to generate a lut
- ndarray [Nxn] with n>3: pre-calculated lut (last col must contain slope of the
isotemperature lines).

luts_dict

> None, optional
> Dictionary of pre-calculated luts for various cspaces and cmf sets.
> > Must have structure luts_dict[cspace][cieobs][lut_label] with the
> > > lut part of a two-element list [lut, lut_kwargs]. It must contain
> > > at the top-level a key 'wl' containing the wavelengths of the
> > > Planckians used to generate the luts in this dictionary.
> > If None: luts_dict defaults to _CCT_LUT['ohno2014']['luts']

cspace

> _CCT_SPACE, optional
> Color space to do calculations in.
> Options:
> > - cspace string:
> > > e.g. 'Yuv60' for use with luxpy.colortf()
> > - tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)]
> > functions
> > > (and an optional string describing the cspace):
> > > > e.g. (forward, backward) or (forward, backward, cspace
> > > > string) or (forward, cspace string)
> > - dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str'
> > (cspace string)]
> > Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in
> > the CIE 1976 u'v' diagram

cspace_kwargs

> _CCT_CSPACE_KWARGS, optional
> Parameter nested dictionary for the forward and backward transforms.

ignore_wl_diff

> False, optional
> When getting a lut from the dictionary, if differences are
> detected in the wavelengts of the lut and the ones used to calculate any
> plankcians then a new lut should be generated. Seting this to True ignores
> these differences and proceeds anyway.

use_fast_duv

> _CCT_FAST_DUV, optional
>> If True: use a fast estimator of the Duv
>>> (one that avoids calculation of Planckians and uses the former
>>>> best estimate's u,v coordinates. This method is accurate enough
>>>> when the atol is small enough -> as long as abs(T-T_former)<=1K
>>>> the Duv estimate should be ok.)

Returns:

> **returns**
>> ndarray with:
>>> cct: out == 'cct' (or 1)
>>> duv: out == 'duv' (or -1)
>>> cct, duv: out == 'cct,duv' (or 2)
>>> [cct,duv]: out == "[cct,duv]" (or -2)

Note: 1. Out-of-lut CCTs are encoded as negative CCTs (with as absolute value the value of the closest CCT from the lut.)

References: 1. Ohno Y. Practical use and calculation of CCT and Duv. Leukos. 2014 Jan 2;10(1):47-55.

2. Li, C., Cui, G., Melgosa, M., Ruan, X., Zhang, Y., Ma, L., Xiao, K., & Luo, M. R. (2016). Accurate method for computing correlated color temperature. Optics Express, 24(13), 14066–14078.

luxpy.color.cct.**xyz_to_cct_li2016**(*xyzw*, *cieobs='1931_2'*, *out='cct'*, *is_uv_input=False*, *wl=None*, *atol=0.1*, *rtol=1e-05*, *max_iter=10*, *split_calculation_at_N=25*, *lut=None*, *luts_dict=None*, *ignore_wl_diff=False*, *lut_resolution_reduction_factor=4*, *cspace='Yuv60'*, *cspace_kwargs={'bwtf': {}, 'fwtf': {}}*, *first_guess_mode='robertson1968'*, *fgm_kwargs={}*, *use_fast_duv=True*, ***kwargs*)

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv(distance above (> 0) or below (< 0) the Planckian locus) using the Newton-Raphson method described in Li et al. (2016).

Args:

> **xyzw**
>> ndarray of tristimulus values

> **cieobs**
>> luxpy._CIEOBS, optional
>> CMF set used to calculated xyzw.

> **out**
>> 'cct' (or 1), optional
>> Determines what to return.
>> Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)

> **is_uv_input**
>> False, optional
>> If True: xyzw contain uv input data, not xyz data!

> **wl**
>> None, optional
>> Wavelengths used when calculating Planckian radiators.
>> If None: use same wavelengths as CMFs in :cieobs:.

> **rtol**

1e-5, float, optional
Stop method when cct a relative tolerance is reached.
The relative tolerance is calculated as dCCT/CCT_est,
with CCT_est the current intermediate estimate in the
search and with dCCT the difference between
the present and former estimates.

atol

0.1, optional
Stop method when cct a absolute tolerance (K) is reached.

max_iter

_CCT_MAX_ITER, optional
Maximum number of iterations used newton-raphson methods.

lut_resolution_reduction_factor

_CCT_LUT_RESOLUTION_REDUCTION_FACTOR, optional
Number of times the interval spanned by the adjacent Tc in a search or lut
method is downsampled (the search process will then start again)

split_calculation_at_N

_CCT_SPLIT_CALC_AT_N, optional
Split calculation when xyzw.shape[0] > split_calculation_at_N.
Splitting speeds up the calculation. If None: no splitting is done.

lut

None, optional
Look-Up-Table with Ti, u,v,u',v',u",v",slope values of Planckians.
Options:
- None: defaults to the lut specified in
_CCT_LUT[first_guess_mode]['lut_type_def'].
- list (lut,lut_kwargs): use this pre-calculated lut
(add additional kwargs for the lut_generator_fcn(), defaults to None if
omitted)
- tuple: must be key (label) in :luts_dict: (pre-calculated dict of luts),
if not: then a new lut will be generated from scratch using the info in the
tuple.
- str: must be key (label) in :luts_dict: (pre-calculated dict of luts)
- ndarray [Nx1]: list of luts for which to generate a lut
- ndarray [Nxn] with n>3: pre-calculated lut (last col must contain slope of the
isotemperature lines).

luts_dict

None, optional
Dictionary of pre-calculated luts for various cspaces and cmf sets.
Must have structure luts_dict[cspace][cieobs][lut_label] with the
lut part of a two-element list [lut, lut_kwargs]. It must contain
at the top-level a key 'wl' containing the wavelengths of the
Planckians used to generate the luts in this dictionary.
If None: luts_dict defaults to _CCT_LUT[first_guess_mode]['luts']

4.4. Color sub-package 97

cspace

_CCT_SPACE, optional

Color space to do calculations in.

Options:

- cspace string:

e.g. 'Yuv60' for use with luxpy.colortf()

- tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)] functions

(and an optional string describing the cspace):

e.g. (forward, backward) or (forward, backward, cspace string) or (forward, cspace string)

- dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str' (cspace string)]

Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in the CIE 1976 u'v' diagram

cspace_kwargs

_CCT_CSPACE_KWARGS, optional

Parameter nested dictionary for the forward and backward transforms.

ignore_wl_diff

False, optional

When getting a lut from the dictionary, if differences are detected in the wavelengts of the lut and the ones used to calculate any plankcians then a new lut should be generated. Seting this to True ignores these differences and proceeds anyway.

first_guess_mode

'robertson1968', optional

Method used to get an approximate (first guess) estimate of the cct, after which the newton-raphson method is started.

Options: 'robertson1968', 'ohno2014', 'zhang2019'

fgm_kwargs

Dict with keyword arguments for the selected first_guess_mode.

use_fast_duv

_CCT_FAST_DUV, optional

If True: use a fast estimator of the Duv

(one that avoids calculation of Planckians and uses the former best estimate's u,v coordinates. This method is accurate enough when the atol is small enough -> as long as abs(T-T_former)<=1K the Duv estimate should be ok.)

Returns:

returns

ndarray with:

cct: out == 'cct' (or 1)

duv: out == 'duv' (or -1)

cct, duv: out == 'cct,duv' (or 2)

[cct,duv]: out == "[cct,duv]" (or -2)

Note: 1. Out-of-lut (of first_guess_mode) CCTs are encoded as negative CCTs (with as absolute value the value of the closest CCT from the lut.)

References: 1. Li, C., Cui, G., Melgosa, M., Ruan, X., Zhang, Y., Ma, L., Xiao, K., & Luo, M. R. (2016). Accurate method for computing correlated color temperature. Optics Express, 24(13), 14066–14078.

2. Robertson, A. R. (1968). Computation of Correlated Color Temperature and Distribution Temperature. Journal of the Optical Society of America, 58(11), 1528–1535.

luxpy.color.cct.**xyz_to_cct_li2022**(*xyzw*, *cieobs='1931_2'*, *out='cct'*, *is_uv_input=False*, *wl=None*, *atol=0.1*, *rtol=1e-05*, *force_tolerance=True*, *tol_method='newton-raphson'*, *lut_resolution_reduction_factor=4*, *duv_triangular_threshold=0.002*, *split_calculation_at_N=25*, *max_iter=10*, *cspace='Yuv60'*, *cspace_kwargs={'bwtf': {}, 'fwtf': {}}*, *lut=None*, *luts_dict=None*, *ignore_wl_diff=False*, *use_fast_duv=True*, ***kwargs*)

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv (distance above (>0) or below (<0) the Planckian locus) using Li's 2022 update (proposal 2) of Ohno's 2014 method.

Args:

    xyzw

        ndarray of tristimulus values

    cieobs

        luxpy._CIEOBS, optional
        CMF set used to calculated xyzw.

    out

        'cct' (or 1), optional
        Determines what to return.
        Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)

    is_uv_input

        False, optional
        If True: xyzw contain uv input data, not xyz data!

    wl

        None, optional
        Wavelengths used when calculating Planckian radiators.
        If None: use same wavelengths as CMFs in :cieobs:.

    rtol

        1e-5, float, optional
        Stop search when cct a relative tolerance is reached.
        The relative tolerance is calculated as dCCT/CCT_est,
        with CCT_est the current intermediate estimate in the
        search and with dCCT the difference between
        the present and former estimates.

    atol

        0.1, optional
        Stop search when cct a absolute tolerance (K) is reached.

force_tolerance

> True, optional
> If False: search only using the list of CCTs in the used lut.
>> Only one loop of the full algorithm is performed.
>> Accuracy depends on CCT of test source and the location
>> and spacing of the CCTs in the list.
> If True: search will use adjacent CCTs to test source to create a new LUT,
>> (repeat the algoritm at higher resolution, progessively zooming in
>>> toward the ground-truth) for tol_method == 'cl'; when
>> tol_method == 'nr' a newton-raphson method is used.
>> Because the CCT for multiple source is calculated in one go,
>> the atol and rtol values have to be met for all!

tol_method

> 'newton-raphson', optional
> (Additional) method to try and achieve set tolerances.
> Options:
> - 'cl', 'cascading-lut': use increasingly higher CCT-resolution
>> to 'zoom-in' on the ground-truth.
> - 'nr', 'newton-raphson': use the method as described in Li, 2016.

lut_resolution_reduction_factor

> _CCT_LUT_RESOLUTION_REDUCTION_FACTOR, optional
> Number of times the interval spanned by the adjacent Tc in a search or lut
> method is downsampled (the search process will then start again)

duv_triangular_threshold

> 0.002, optional
> Threshold for use of the triangular solution
>> (if smaller use triangular solution, else use the non-triangular (third order
>> polynomial))

max_iter

> _CCT_MAX_ITER, optional
> Maximum number of iterations used by the cascading-lut or newton-raphson methods.

split_calculation_at_N

> _CCT_SPLIT_CALC_AT_N, optional
> Split calculation when xyzw.shape[0] > split_calculation_at_N.
> Splitting speeds up the calculation. If None: no splitting is done.

lut

> None, optional
> Look-Up-Table with Ti, u,v,u',v',u",v",slope values of Planckians.
> Options:
>> - None: defaults to the lut specified in _CCT_LUT['li2022']['lut_type_def'].
>> - list (lut,lut_kwargs): use this pre-calculated lut
>>> (add additional kwargs for the lut_generator_fcn(), defaults to None if
>>> omitted)

- tuple: must be key (label) in :luts_dict: (pre-calculated dict of luts),
    if not: then a new lut will be generated from scratch using the info in the
    tuple.
- str: must be key (label) in :luts_dict: (pre-calculated dict of luts)
- ndarray [Nx1]: list of luts for which to generate a lut
- ndarray [Nxn] with n>3: pre-calculated lut (last col must contain slope of the
isotemperature lines).

luts_dict

None, optional
Dictionary of pre-calculated luts for various cspaces and cmf sets.
    Must have structure luts_dict[cspace][cieobs][lut_label] with the
        lut part of a two-element list [lut, lut_kwargs]. It must contain
        at the top-level a key 'wl' containing the wavelengths of the
        Planckians used to generate the luts in this dictionary.
If None: luts_dict defaults to _CCT_LUT['li2022']['luts']

cspace

_CCT_SPACE, optional
Color space to do calculations in.
Options:
        - cspace string:
            e.g. 'Yuv60' for use with luxpy.colortf()
        - tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)]
        functions
            (and an optional string describing the cspace):
                e.g. (forward, backward) or (forward, backward, cspace
                string) or (forward, cspace string)
        - dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str'
        (cspace string)]
    Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in
    the CIE 1976 u'v' diagram

cspace_kwargs

_CCT_CSPACE_KWARGS, optional
Parameter nested dictionary for the forward and backward transforms.

ignore_wl_diff

False, optional
When getting a lut from the dictionary, if differences are
detected in the wavelengts of the lut and the ones used to calculate any
plankcians then a new lut should be generated. Seting this to True ignores
these differences and proceeds anyway.

use_fast_duv

_CCT_FAST_DUV, optional
If True: use a fast estimator of the Duv
    (one that avoids calculation of Planckians and uses the former
        best estimate's u,v coordinates. This method is accurate enough
        when the atol is small enough -> as long as abs(T-T_former)<=1K

the Duv estimate should be ok.)

Returns:

> **returns**
>
> > ndarray with:
> >
> > > cct: out == 'cct' (or 1)
> > >
> > > duv: out == 'duv' (or -1)
> > >
> > > cct, duv: out == 'cct,duv' (or 2)
> > >
> > > [cct,duv]: out == "[cct,duv]" (or -2)

Note: 1. Out-of-lut CCTs are encoded as negative CCTs (with as absolute value the value of the closest CCT from the lut.)

References: 1. Ohno Y. Practical use and calculation of CCT and Duv. Leukos. 2014 Jan 2;10(1):47-55.

> 2. Li, Y., Gao, C., Melgosa, M. and Li, C. (2022). Improved methods for computing CCT and Duv. LEUKOS, (in press).

luxpy.color.cct.**xyz_to_cct_zhang2019**(*xyzw*, *cieobs='1931_2'*, *out='cct'*, *is_uv_input=False*, *wl=None*, *atol=0.1*, *rtol=1e-05*, *force_tolerance=True*, *tol_method='newton-raphson'*, *lut_resolution_reduction_factor=4*, *split_calculation_at_N=25*, *max_iter=10*, *cspace='Yuv60'*, *cspace_kwargs={'bwtf': {}, 'fwtf': {}}*, *lut=None*, *luts_dict=None*, *ignore_wl_diff=False*, *use_fast_duv=True*, ***kwargs*)

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv(distance above (> 0) or below (< 0) the Planckian locus) using the golden-ratio search method described in Zhang et al. (2019).

Args:

> **xyzw**
>
> > ndarray of tristimulus values
>
> **cieobs**
>
> > luxpy._CIEOBS, optional
> > CMF set used to calculated xyzw.
>
> **out**
>
> > 'cct' (or 1), optional
> > Determines what to return.
> > Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)
>
> **is_uv_input**
>
> > False, optional
> > If True: xyzw contain uv input data, not xyz data!
>
> **wl**
>
> > None, optional
> > Wavelengths used when calculating Planckian radiators.
> > If None: use same wavelengths as CMFs in :cieobs:.
>
> **rtol**
>
> > 1e-5, float, optional
> > Stop search when cct a relative tolerance is reached.
> > The relative tolerance is calculated as dCCT/CCT_est,
> > with CCT_est the current intermediate estimate in the

> search and with dCCT the difference between
> the present and former estimates.

atol

> 0.1, optional
> Stop search when cct a absolute tolerance (K) is reached.

force_tolerance

> True, optional
> If False: search only using the list of CCTs in the used lut.
> > Only one loop of the full algorithm is performed.
> > Accuracy depends on CCT of test source and the location
> > and spacing of the CCTs in the list.
> If True: search will use adjacent CCTs to test source to create a new LUT,
> > (repeat the algoritm at higher resolution, progessively zooming in
> > > toward the ground-truth) for tol_method == 'cl'; when
> > tol_method == 'nr' a newton-raphson method is used.
> > Because the CCT for multiple source is calculated in one go,
> > the atol and rtol values have to be met for all!

tol_method

> 'newton-raphson', optional
> (Additional) method to try and achieve set tolerances.
> Options:
> - 'cl', 'cascading-lut': use increasingly higher CCT-resolution
> > to 'zoom-in' on the ground-truth.
> - 'nr', 'newton-raphson': use the method as described in Li, 2016.

lut_resolution_reduction_factor

> _CCT_LUT_RESOLUTION_REDUCTION_FACTOR, optional
> Number of times the interval spanned by the adjacent Tc in a search or lut
> method is downsampled (the search process will then start again)

max_iter

> _CCT_MAX_ITER, optional
> Maximum number of iterations used by the cascading-lut or newton-raphson methods.

split_calculation_at_N

> _CCT_SPLIT_CALC_AT_N, optional
> Split calculation when xyzw.shape[0] > split_calculation_at_N.
> Splitting speeds up the calculation. If None: no splitting is done.

lut

> None, optional
> Look-Up-Table with Ti, u,v,u',v',u'',v'',slope values of Planckians.
> Options:
> > - None: defaults to the lut specified in
> > _CCT_LUT['zhang2019']['lut_type_def'].
> > - list (lut,lut_kwargs): use this pre-calculated lut

(add additional kwargs for the lut_generator_fcn(), defaults to None if omitted)

- tuple: must be key (label) in :luts_dict: (pre-calculated dict of luts),

> if not: then a new lut will be generated from scratch using the info in the tuple.

- str: must be key (label) in :luts_dict: (pre-calculated dict of luts)

- ndarray [Nx1]: list of luts for which to generate a lut

- ndarray [Nxn] with n>3: pre-calculated lut (last col must contain slope of the isotemperature lines).

luts_dict

> None, optional
>
> Dictionary of pre-calculated luts for various cspaces and cmf sets.
>> Must have structure luts_dict[cspace][cieobs][lut_label] with the
>>> lut part of a two-element list [lut, lut_kwargs]. It must contain at the top-level a key 'wl' containing the wavelengths of the Planckians used to generate the luts in this dictionary.
>
> If None: luts_dict defaults to _CCT_LUT['zhang2019']['luts']

cspace

> _CCT_SPACE, optional
>
> Color space to do calculations in.
>
> Options:
>> - cspace string:
>>> e.g. 'Yuv60' for use with luxpy.colortf()
>>
>> - tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)] functions
>>> (and an optional string describing the cspace):
>>>> e.g. (forward, backward) or (forward, backward, cspace string) or (forward, cspace string)
>>
>> - dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str' (cspace string)]
>
>> Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in the CIE 1976 u'v' diagram

cspace_kwargs

> _CCT_CSPACE_KWARGS, optional
>
> Parameter nested dictionary for the forward and backward transforms.

ignore_wl_diff

> False, optional
>
> When getting a lut from the dictionary, if differences are detected in the wavelengts of the lut and the ones used to calculate any plankcians then a new lut should be generated. Seting this to True ignores these differences and proceeds anyway.

use_fast_duv

> _CCT_FAST_DUV, optional
>
> If True: use a fast estimator of the Duv
>> (one that avoids calculation of Planckians and uses the former

> > best estimate's u,v coordinates. This method is accurate enough
> > when the atol is small enough -> as long as abs(T-T_former)<=1K
> > the Duv estimate should be ok.)

Returns:

> **returns**
>
> > ndarray with:
> >
> > > cct: out == 'cct' (or 1)
> > >
> > > duv: out == 'duv' (or -1)
> > >
> > > cct, duv: out == 'cct,duv' (or 2)
> > >
> > > [cct,duv]: out == "[cct,duv]" (or -2)

Note: 1. Out-of-lut CCTs are encoded as negative CCTs (with as absolute value the value of the closest CCT from the lut.)

References: 1. Zhang, F. (2019). High-accuracy method for calculating correlated color temperature with a lookup table based on golden section search. Optik, 193, 163018.

> 2. Li, C., Cui, G., Melgosa, M., Ruan, X., Zhang, Y., Ma, L., Xiao, K., & Luo, M. R. (2016). Accurate method for computing correlated color temperature. Optics Express, 24(13), 14066–14078.

luxpy.color.cct.**xyz_to_cct_fibonacci**(*xyzw*, *cieobs='1931_2'*, *out='cct'*, *is_uv_input=False*, *wl=None*, *atol=0.1*, *rtol=1e-05*, *force_tolerance=True*, *tol_method='newton-raphson'*, *lut_resolution_reduction_factor=4*, *split_calculation_at_N=25*, *max_iter=10*, *cspace='Yuv60'*, *cspace_kwargs={'bwtf': {}, 'fwtf': {}}*, *lut=None*, *luts_dict=None*, *ignore_wl_diff=False*, *use_fast_duv=True*, ***kwargs*)

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv(distance above (> 0) or below (< 0) the Planckian locus) using a Fibonacci search.

Args:

> **xyzw**
>
> > ndarray of tristimulus values
>
> **cieobs**
>
> > luxpy._CIEOBS, optional
> > CMF set used to calculated xyzw.
>
> **out**
>
> > 'cct' (or 1), optional
> > Determines what to return.
> > Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)
>
> **is_uv_input**
>
> > False, optional
> > If True: xyzw contain uv input data, not xyz data!
>
> **wl**
>
> > None, optional
> > Wavelengths used when calculating Planckian radiators.
> > If None: use same wavelengths as CMFs in :cieobs:.
>
> **rtol**
>
> > 1e-5, float, optional

> Stop search when cct a relative tolerance is reached.
> The relative tolerance is calculated as dCCT/CCT_est,
> with CCT_est the current intermediate estimate in the
> search and with dCCT the difference between
> the present and former estimates.

atol

> 0.1, optional
> Stop search when cct a absolute tolerance (K) is reached.

force_tolerance

> True, optional
> If False: search only using the list of CCTs in the used lut.
>> Only one loop of the full algorithm is performed.
>> Accuracy depends on CCT of test source and the location
>> and spacing of the CCTs in the list.
> If True: search will use adjacent CCTs to test source to create a new LUT,
>> (repeat the algoritm at higher resolution, progessively zooming in
>>> toward the ground-truth) for tol_method == 'cl'; when
>> tol_method == 'nr' a newton-raphson method is used.
>> Because the CCT for multiple source is calculated in one go,
>> the atol and rtol values have to be met for all!

tol_method

> 'newton-raphson', optional
> (Additional) method to try and achieve set tolerances.
> Options:
> - 'cl', 'cascading-lut': use increasingly higher CCT-resolution
>> to 'zoom-in' on the ground-truth.
> - 'nr', 'newton-raphson': use the method as described in Li, 2016.

lut_resolution_reduction_factor

> _CCT_LUT_RESOLUTION_REDUCTION_FACTOR, optional
> Number of times the interval spanned by the adjacent Tc in a search or lut
> method is downsampled (the search process will then start again)

max_iter

> _CCT_MAX_ITER, optional
> Maximum number of iterations used by the cascading-lut or newton-raphson methods.

split_calculation_at_N

> _CCT_SPLIT_CALC_AT_N, optional
> Split calculation when xyzw.shape[0] > split_calculation_at_N.
> Splitting speeds up the calculation. If None: no splitting is done.

lut

> None, optional
> Look-Up-Table with Ti, u,v,u',v',u",v",slope values of Planckians.
> Options:
>> - None: defaults to the lut specified in _CCT_LUT['fibonacci']['lut_type_def'].

- list (lut,lut_kwargs): use this pre-calculated lut
    (add additional kwargs for the lut_generator_fcn(), defaults to None if
    omitted)
- tuple: must be key (label) in :luts_dict: (pre-calculated dict of luts),
    if not: then a new lut will be generated from scratch using the info in the
    tuple.
- str: must be key (label) in :luts_dict: (pre-calculated dict of luts)
- ndarray [Nx1]: list of luts for which to generate a lut
- ndarray [Nxn] with n>3: pre-calculated lut (last col must contain slope of the
  isotemperature lines).

luts_dict

None, optional
Dictionary of pre-calculated luts for various cspaces and cmf sets.
    Must have structure luts_dict[cspace][cieobs][lut_label] with the
        lut part of a two-element list [lut, lut_kwargs]. It must contain
        at the top-level a key 'wl' containing the wavelengths of the
        Planckians used to generate the luts in this dictionary.
If None: luts_dict defaults to _CCT_LUT['fibonacci']['luts']

cspace

_CCT_SPACE, optional
Color space to do calculations in.
Options:
        - cspace string:
            e.g. 'Yuv60' for use with luxpy.colortf()
        - tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)]
        functions
            (and an optional string describing the cspace):
                e.g. (forward, backward) or (forward, backward, cspace
                string) or (forward, cspace string)
        - dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str'
        (cspace string)]
    Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in
    the CIE 1976 u'v' diagram

cspace_kwargs

_CCT_CSPACE_KWARGS, optional
Parameter nested dictionary for the forward and backward transforms.

ignore_wl_diff

False, optional
When getting a lut from the dictionary, if differences are
detected in the wavelengts of the lut and the ones used to calculate any
plankcians then a new lut should be generated. Seting this to True ignores
these differences and proceeds anyway.

use_fast_duv

_CCT_FAST_DUV, optional
If True: use a fast estimator of the Duv

(one that avoids calculation of Planckians and uses the former

best estimate's u,v coordinates. This method is accurate enough

when the atol is small enough -> as long as abs(T-T_former)<=1K

the Duv estimate should be ok.)

Returns:

> **returns**
>
>> ndarray with:
>>
>>> cct: out == 'cct' (or 1)
>>>
>>> duv: out == 'duv' (or -1)
>>>
>>> cct, duv: out == 'cct,duv' (or 2)
>>>
>>> [cct,duv]: out == "[cct,duv]" (or -2)

Note: 1. Out-of-lut CCTs (or close to) are encoded as negative CCTs (with as absolute value the value of the closest CCT from the lut.)

luxpy.color.cct.**xyz_to_cct**(*xyzw, mode='robertson1968', cieobs='1931_2', out='cct', is_uv_input=False, wl=None, atol=0.1, rtol=1e-05, force_tolerance=True, tol_method='newton-raphson', lut_resolution_reduction_factor=4, split_calculation_at_N=25, max_iter=10, cspace='Yuv60', cspace_kwargs={'bwtf': {}, 'fwtf': {}}, lut=None, luts_dict=None, ignore_wl_diff=False, duv_triangular_threshold=0.002, first_guess_mode='robertson1968', fgm_kwargs={}, use_fast_duv=True, **kwargs*)*

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv (distance above (>0) or below (<0) the Planckian locus) using a number of modes (methods).

Args:

> **xyzw**
>
>> ndarray of tristimulus values
>
> **mode**
>
>> 'robertson1968', optional
>>
>> String with name of method to use.
>>
>> Options: 'robertson1968', 'ohno2014', 'li2016', 'li2022','zhang2019', 'fibonacci',
>>
>>> (also, but see note below: 'mcamy1992', 'hernandez1999')
>>
>> Note: first_guess_mode for li2016 can also be specified using a ':' separator,
>>
>>> e.g. 'li2016:robertson1968'
>
> **cieobs**
>
>> luxpy._CIEOBS, optional
>>
>> CMF set used to calculated xyzw.
>
> **out**
>
>> 'cct' (or 1), optional
>>
>> Determines what to return.
>>
>> Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)
>
> **is_uv_input**
>
>> False, optional
>>
>> If True: xyzw contain uv input data, not xyz data!
>
> **wl**
>
>> None, optional

Wavelengths used when calculating Planckian radiators.
If None: use same wavelengths as CMFs in :cieobs:.

rtol

1e-5, float, optional
Stop search when cct a relative tolerance is reached.
The relative tolerance is calculated as dCCT/CCT_est,
with CCT_est the current intermediate estimate in the
search and with dCCT the difference between
the present and former estimates.

atol

0.1, optional
Stop search when cct a absolute tolerance (K) is reached.

force_tolerance

True, optional
If False: search only using the list of CCTs in the used lut.
    Only one loop of the full algorithm is performed.
    Accuracy depends on CCT of test source and the location
    and spacing of the CCTs in the list.
If True: search will use adjacent CCTs to test source to create a new LUT,
    (repeat the algoritm at higher resolution, progessively zooming in
        toward the ground-truth) for tol_method == 'cl'; when
    tol_method == 'nr' a newton-raphson method is used.
    Because the CCT for multiple source is calculated in one go,
    the atol and rtol values have to be met for all!

tol_method

'newton-raphson', optional
(Additional) method to try and achieve set tolerances.
Options:
- 'cl', 'cascading-lut': use increasingly higher CCT-resolution
    to 'zoom-in' on the ground-truth. (not for mode == 'li2016')
- 'nr', 'newton-raphson': use the method as described in Li, 2016.

lut_resolution_reduction_factor

_CCT_LUT_RESOLUTION_REDUCTION_FACTOR, optional
Number of times the interval spanned by the adjacent Tc in a search or lut
method is downsampled (the search process will then start again)

max_iter

_CCT_MAX_ITER, optional
Maximum number of iterations used by the cascading-lut or newton-raphson methods.

split_calculation_at_N

_CCT_SPLIT_CALC_AT_N, optional
Split calculation when xyzw.shape[0] > split_calculation_at_N.
Splitting speeds up the calculation. If None: no splitting is done.

lut

> None, optional
>
> Look-Up-Table with Ti, u,v,u',v',u",v",slope values of Planckians.
>
> Options:
>
>> - None: defaults to the lut specified in _CCT_LUT[mode]['lut_type_def'].
>> - list (lut,lut_kwargs): use this pre-calculated lut
>>> (add additional kwargs for the lut_generator_fcn(), defaults to None if omitted)
>> - tuple: must be key (label) in :luts_dict: (pre-calculated dict of luts),
>>> if not: then a new lut will be generated from scratch using the info in the tuple.
>> - str: must be key (label) in :luts_dict: (pre-calculated dict of luts)
>> - ndarray [Nx1]: list of luts for which to generate a lut
>> - ndarray [Nxn] with n>3: pre-calculated lut (last col must contain slope of the isotemperature lines).

luts_dict

> None, optional
>
> Dictionary of pre-calculated luts for various cspaces and cmf sets.
>> Must have structure luts_dict[cspace][cieobs][lut_label] with the
>>> lut part of a two-element list [lut, lut_kwargs]. It must contain
>>> at the top-level a key 'wl' containing the wavelengths of the
>>> Planckians used to generate the luts in this dictionary.
>
> If None: the default dict for the mode is used
>> (e.g. _CCT_LUT['ohno2014']['lut_type_def'], for mode=='ohno2014').

cspace

> _CCT_SPACE, optional
>
> Color space to do calculations in.
>
> Options:
>
>> - cspace string:
>>> e.g. 'Yuv60' for use with luxpy.colortf()
>> - tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)]
>> functions
>>> (and an optional string describing the cspace):
>>>> e.g. (forward, backward) or (forward, backward, cspace string) or (forward, cspace string)
>> - dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str' (cspace string)]
>
> Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in the CIE 1976 u'v' diagram

cspace_kwargs

> _CCT_CSPACE_KWARGS, optional
>
> Parameter nested dictionary for the forward and backward transforms.

ignore_wl_diff

> False, optional
>
> When getting a lut from the dictionary, if differences are

detected in the wavelengts of the lut and the ones used to calculate any
plankcians then a new lut should be generated. Seting this to True ignores
these differences and proceeds anyway.

duv_triangular_threshold

0.002, optional
Threshold for use of the triangular solution.
> (if smaller use triangular solution, else use the non-triangular one:
> > If mode == 'ohno2014' -> parabolic, if mode == 'li2022' -> 3e-order
> > poly)

first_guess_mode

'robertson1968', optional (cfr. mode == 'li2016')
Method used to get an approximate (first guess) estimate of the cct,
after which the newton-raphson method is started.
Options: 'robertson1968', 'ohno2014', 'zhang2019','li2022'

use_fast_duv

_CCT_FAST_DUV, optional
If True: use a fast estimator of the Duv
> (one that avoids calculation of Planckians and uses the former
> > best estimate's u,v coordinates. This method is accurate enough
> > when the atol is small enough -> as long as abs(T-T_former)<=1K
> > the Duv estimate should be ok.)

Returns:

returns

ndarray with:
> cct: out == 'cct' (or 1)
> duv: out == 'duv' (or -1)
> cct, duv: out == 'cct,duv' (or 2)
> [cct,duv]: out == "[cct,duv]" (or -2)

Note: 1. Using the 'mcamy1992' and 'hernandez1999' options will result in additional errors when cieobs
is different from '1931_2' as for these options the CCT is determined using a fixed set of equations
optimized for the 1931 2° CMFs!! The only impact will be on the calculation of the Duv from the CCT.
That does depend on the settings of cieobs and cspace! Change at own discretion. 2. Out-of-lut CCTs are
encoded as negative CCTs (with as absolute value the value of the closest CCT from the lut.)

References: 1. Robertson, A. R. (1968). Computation of Correlated Color Temperature and Distribution Temperature. Journal of the Optical Society of America, 58(11), 1528–1535.

2. Smet K.A.G., Royer M., Baxter D., Bretschneider E., Esposito E., Houser K., Luedtke W., Man K.,
Ohno Y. (2022), Recommended method for determining the correlated color temperature and distance
from the Planckian Locus of a light source (in preparation, LEUKOS?)

3. Baxter D., Royer M., Smet K.A.G. (2022) Modifications of the Robertson Method for Calculating
Correlated Color Temperature to Improve Accuracy and Speed (in preparation, LEUKOS?)

4. Ohno Y. Practical use and calculation of CCT and Duv. Leukos. 2014 Jan 2;10(1):47-55.

5. Zhang, F. (2019). High-accuracy method for calculating correlated color temperature with a lookup
table based on golden section search. Optik, 193, 163018.

6. Li, C., Cui, G., Melgosa, M., Ruan, X., Zhang, Y., Ma, L., Xiao, K., & Luo, M. R. (2016). Accurate
method for computing correlated color temperature. Optics Express, 24(13), 14066–14078.

7. McCamy, Calvin S. (April 1992). "Correlated color temperature as an explicit function of chromaticity coordinates". Color Research & Application. 17 (2): 142–144.

8. Hernández-Andrés, Javier; Lee, RL; Romero, J (September 20, 1999). Calculating Correlated Color Temperatures Across the Entire Gamut of Daylight and Skylight Chromaticities. Applied Optics. 38 (27), 5703–5709. P

9. Li, Y., Gao, C., Melgosa, M. and Li, C. (2022). Improved methods for computing CCT and Duv. LEUKOS, (in press).

luxpy.color.cct.**cct_to_xyz**(*ccts*, *duv=None*, *cct_offset=None*, *cieobs='1931_2'*, *wl=None*, *cspace='Yuv60'*, *cspace_kwargs={'bwtf': {}, 'fwtf': {}}*)

Convert correlated color temperature (550 K <= CCT <= 1e11 K) and Duv (distance above (>0) or below (<0) the Planckian locus) to XYZ tristimulus values.

Finds xyzw_estimated by determining the iso-temperature line

(= line perpendicular to the Planckian locus):

Option 1 (fastest):

First, the angle between the coordinates corresponding to ccts

and ccts-cct_offset are calculated, then 90° is added, and finally

the new coordinates are determined, while taking sign of duv into account.

Option 2 (slowest, about 55% slower):

Calculate the slope of the iso-T-line directly using the Planckian

spectrum and its derivative.

Args:

ccts

ndarray [N,1] of cct values

duv

None or ndarray [N,1] of duv values, optional

Note that duv can be supplied together with cct values in :ccts:

as ndarray with shape [N,2].

cct_offset

None, optional

If None: use option 2 (direct iso-T slope calculation, more accurate,

but slower: about 1.55 slower)

else: use option 1 (estimate slope from 90° + angle of small cct_offset)

cieobs

luxpy._CIEOBS, optional

CMF set used to calculated xyzw.

wl

None, optional

Wavelengths used when calculating Planckian radiators.

If None: use same wavelengths as CMFs in :cieobs:.

cspace

_CCT_SPACE, optional

Color space to do calculations in.
Options:

>> - cspace string:

>>> e.g. 'Yuv60' for use with luxpy.colortf()

>> - tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)]
>> functions

>>> (and an optional string describing the cspace):

>>>> e.g. (forward, backward) or (forward, backward, cspace
>>>> string) or (forward, cspace string)

>> - dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str'
>> (cspace string)]

> Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in
> the CIE 1976 u'v' diagram

> **cspace_kwargs**

>> _CCT_CSPACE_KWARGS, optional
>> Parameter nested dictionary for the forward and backward transforms.

Returns:

> **returns**

>> ndarray with estimated XYZ tristimulus values

Note: 1. If duv is not supplied (:ccts:.shape is (N,1) and :duv: is None), source is assumed to be on the Planckian
locus. 2. Minimum CCT is 550 K (lower than 550 K, some negative Duv values will result in coordinates
outside of the Spectrum Locus !!!)

luxpy.color.cct.**calculate_lut**(*ccts, cieobs, wl=None, lut_vars=['T', 'uv', 'uvp', 'uvpp', 'iso-T-
slope'], cspace='Yuv60', cspace_kwargs={'bwtf': {}, 'fwtf': {}})*
Function that calculates a LUT for the specified calculation method for the input ccts. Calculation is performed
for CMF set specified in cieobs and in the chromaticity diagram in cspace.

Args:

> **ccts**

>> ndarray [Nx1] or str
>> list of ccts for which to (re-)calculate the LUTs.
>> If str, ccts contains path/filename.dat to list.

> **cieobs**

>> None or str, optional
>> str specifying cmf set.

> **wl**

>> None, optional
>> Generate luts based on Planckians with wavelengths (range).
>> If None: use same wavelengths as CMFs in :cieobs:.

> **lut_vars**

>> ['T','uv','uvp','uvpp','iso-T-slope'], optional
>> Data the lut should contain. Must follow this order
>> and minimum should be ['T']

> **cspace**

>> _CCT_SPACE, optional
>> Color space to do calculations in.

Options:

- cspace string:

e.g. 'Yuv60' for use with luxpy.colortf()

- tuple with forward (i.e. xyz_to..) [and backward (i.e. ..to_xyz)] functions

(and an optional string describing the cspace):

e.g. (forward, backward) or (forward, backward, cspace string) or (forward, cspace string)

- dict with keys: 'fwtf' (foward), 'bwtf' (backward) [, optional: 'str' (cspace string)]

Note: if the backward tf is not supplied, optimization in cct_to_xyz() is done in the CIE 1976 u'v' diagram

cspace_kwargs

_CCT_CSPACE_KWARGS, optional
Parameter nested dictionary for the forward and backward transforms.

Returns:

returns

lut

ndarray with T, u, v, u', v', u'', v'', slope (note ':1st deriv., '':2nd deriv.).

luxpy.color.cct.**generate_luts**(*types=[None],   seamless_stitch=True, fallback_unit='K-1',   fallback_n=50,   cct_min=450, cct_max=100000000000.0,   lut_file=None,   load=False, lut_path='C:\\Users\\u0032318\\OneDrive - KU Leuven\\Documents\\Github\\luxpy\\luxpy\\data\\cctluts\\', save_luts=True, wl=None, cieobs=['1931_2'], lut_vars=['T', 'uv', 'uvp', 'uvpp', 'iso-T-slope'], cspace=['Yuv60'], cspace_kwargs=[{'fwtf':   {}, 'bwtf':   {}}], verbosity=0,   lut_generator_fcn=<function   _generate_lut>, lut_generator_kwargs={}*)

Generate a number of luts and store them in a nested dictionary. Structure: lut[cspace][cieobs][lut type].

Args:

lut_file

None, optional
string specifying the filename to save the lut (as .pkl) to.
If None: don't save anything when generated (i.e. load==False).

load

True, optional
If True: load previously generated dictionary.
If False: generate from scratch.

lut_path

_CCT_LUT_PATH, optional
Path to file.

wl

None, optional
Wavelength for Planckian spectrum generation.

> If None: use same wavelengths as CMFs in :cieobs:.

cieobs

> [_CIEOBS] or list, optional
> Generate a LUT for each one in the list.
> If None: generate for all cmfs in _CMF.

types

> [None], optional
> List of lut specifiers of format [(Tmin,Tmax,Tinterval,unit),. . .]
> If units are in MK-1 then the range is also!
>> Unit options are:
>> - '%': equal relative Tc spacing (in %, cfr. (Ti+1 - Ti-1)/Ti-1).
>> - 'K' equal absolute Tc spacing (in K, cfr. (Ti+1 - Ti-1).
>> - '%-1': equal relative reciprocal Tc (MK-1 = mired).
>> - 'K-1': equal absolute reciprocal Tc (MK-1 = mired).
> If the last element of the list is a bool, then the way the different
> lists of Tcs generated by each list element can be set. If True:
> the Tcs will be 'seamlessly' stitched together (this does have an
> an impact on the min-max range of each Tc set) so that there are no
> discontinuities in terms of the intervals.

seamless_stitch

> True, optional
> When stitching (creating) LUTs composed of several CCT ranges with different
> intervals, these do not always 'match' well, in the sense that discontinuities
> might be generated. This can be avoided (at the expense of possibly slightly changed
> ranges)
> by setting the :seamless_stitch: argument to True. Is overriden when
> the last element in the lut list is a boolean.

cct_max

> _CCT_MAX, optional
> Limit Tc's to a maximum value of cct_max

cct_min

> _CCT_MIN, optional
> Limit Tc's to a minimum value of cct_max

fallback_unit

> _CCT_FALLBACK_UNIT, optional
> Unit to fall back on when the input unit in tc4 (of first list) is 'au'.
> As there is no common distancing of the unit types ['K','%','%-1','K-1']
> the Tc's are generated by dividing the min-max range into
> a number of divisions, specified by the negative 3 element (or when
> positive or NaN, the number of divisions is set by :fallback_divisions:)

fallback_n

> _CCT_FALLBACK_N, optional
> Number of divisions the min-max range is divided into, in the

---

> fallback case in which unit=='au' and the 3e 4-vector element
> is NaN or positive.

lut_vars

> ['T','uv','uvp','uvpp','iso-T-slope'], optional
> Data the lut should contain. Must follow this order
> and minimum should be ['T']

cspace,cspace_kwargs

> Lists with the cspace and cspace_kwargs for which luts will be generated.
> Default is single chromaticity diagram in _CCT_CSPACE.

verbosity

> 0, optional
> If > 0: give some intermediate feedback while generating luts.

lut_generator_fcn

> _generate_lut, optional
> Lets a user specify his own lut generation function (must output a list of 1 lut).
> Default is the general function. There is a specific one for
> Ohno's 2014 method as that one requires a different correction factor
> for each lut for the parabolic solutions. This optimized value is specified in the
> second list index. (see _generate_lut_ohno2014()).

lut_generator_kwargs

> {}, optional
> Dict with keyword arguments specific to the (user) lut_generator_fcn.
> > (e.g. {'f_corr':0.9991} for _generate_lut_ohno2014())

Returns:

> **dict**

> > Dictionary with luts for the specified mode, cieobs(s) and cspace(s).
> > Structure: lut[cspace][cieobs][lut type]
> > At the upper dict level there is also a key 'wl' which contains a dict with keys
> > the cieobs and with values the wavelengths used to calculate the Planckians for
> > each lut for the specified cieobs; as well as a key with the lut_vars
> > The luts contains as data the variables as specified in lut_vars:
> > - T: (in K)
> > - uv: chromaticity coordinates of planckians
> > - uvp: chromaticity coordinates of 1st derivative of the planckians.
> > - uvpp: chromaticity coordinates of 2nd derivative of the planckians.
> > - iso-T-slope: slope of isotemperature lines (calculated as in Robertson, 1968).

luxpy.color.cct.**get_tcs4**(*tc4*, *uin=None*, *seamless_stitch=True*, *fallback_unit='K-1'*, *fallback_n=50*)

> Get an ndarray of Tc's obtained from a list or tuple of tc4 4-vectors.

> **Args:**
> > **tc4**

> > > list or tuple of 4-vectors.
> > > > e.g. (tc4_1, tc4_2, tc4_3,...) or (tc4_1, tc4_2, tc4_3,..., bool::seamless_stitch)

When the last element of the list/tuple is a bool, then this specifies
> how the Tc arrays generated for each of the 4-vector elements need to be
> stitched together. This overrides the seamless_stitch input argument.

Vector elements are:
> [Tmin, Tmax inclusive, Tinterval(or number of intervals), unit]

> Unit specifies unit of the Tc interval, i.e. it determines the
> > type of scale in which the spacing of the Tc are done.

> Unit options are:
> > - '%': equal relative Tc spacing (in %, cfr. (Ti+1 - Ti-1)/Ti-1).
> > - 'K' equal absolute Tc spacing (in K, cfr. (Ti+1 - Ti-1).
> > - '%-1': equal relative reciprocal Tc (MK-1 = mired).
> > - 'K-1': equal absolute reciprocal Tc (MK-1 = mired).

> If the 'interval' element is negative, it actually represents
> the number of intervals between Tmin, Tmax (included).

uin

> None, optional
> Unit of input Tmin, Tmax (by default it is assumed to be the same
> as the scale 'unit').

seamless_stitch

> True, optional
> Determines how the Tc arrays generated for each of the 4-vector
> elements are stitched together. Is overriden by the presence of a
> bool as last list/tuple element in :tc4:.
> For a seamless stitch, all units for all 4-vectors should be the same!!

fallback_unit

> _CCT_FALLBACK_UNIT, optional
> Unit to fall back on when the input unit in tc4 (of first list) is 'au'.
> As there is no common distancing of the unit types ['K','%','%-1','K-1']
> the Tc's are generated by dividing the min-max range into
> a number of divisions, specified by the negative 3 element (or when
> positive or NaN, the number of divisions is set by :fallback_divisions:)

fallback_n

> _CCT_FALLBACK_N, optional
> Number of divisions the min-max range is divided into, in the
> fallback case in which unit=='au' and the 3e 4-vector element
> is NaN or positive.

Returns:

tcs

> ndarray with Tcs

luxpy.color.cct.**_get_lut**(*lut, uin=None, seamless_stitch=True, fallback_unit='K-1', fall-back_n=50, resample_ndarray=False, cct_max=100000000000.0, cct_min=450, luts_dict=None, lut_type_def=None, lut_vars=['T', 'uv', 'uvp', 'uvpp', 'iso-T-slope'], cieobs='1931_2', cspace_str=None, wl=None, ignore_unequal_wl=False, lut_generator_fcn=<function _generate_lut>, lut_generator_kwargs={}, cspace='Yuv60', cspace_kwargs={'bwtf': {}, 'fwtf': {}}, **kwargs*)

   Get an ndarray LUT from various sources.

   Args:

   > **lut**
   >
   > > Look-Up-Table with Ti, u,v,u',v',u'',v'',slope values of Planckians, or
   > > whatever quantities are specified in lut_vars ('T','uv' is always part of the lut).
   > > Options:
   > > - list: must have two elements: [lut,lut_kwargs]
   > > - None: lut from luts_dict with lut_type_def as key
   > > - str: lut from luts_dict at key :lut:
   > > - ndarray [Nxn, with n>1]: precalculated lut (only processing will be to keep it with cct_min-cct_max range)
   > > - ndarray [Nx1]: list of Tc's from which a new lut will be calculated.
   > > - tuple of 4-vectors: used as key in luts_dict or to generate new lut from scratch
   > >
   > > > 4-vector info:
   > > >
   > > > > + format: e.g. (tc4_1, tc4_2, tc4_3,…) or (tc4_1, tc4_2, tc4_3,…, bool::seamless_stitch)
   > > > > + When the last element of the list/tuple is a bool, then this specifies
   > > > >
   > > > > > how the Tc arrays generated for each of the 4-vector elements need to be
   > > > > > stitched together. This overrides the seamless_stitch input argument.
   > > > >
   > > > > + Vector elements are:
   > > > >
   > > > > > [Tmin, Tmax inclusive, Tinterval(or number of intervals), unit]
   > > > > >
   > > > > > Unit specifies unit of the Tc interval, i.e. it determines the
   > > > > > type of scale in which the spacing of the Tc are done.
   > > > > > Unit options are:
   > > > > >
   > > > > > > - '%': equal relative Tc spacing (in %, cfr. (Ti+1 - Ti-1)/Ti-1).
   > > > > > > - 'K' equal absolute Tc spacing (in K, cfr. (Ti+1 - Ti-1).
   > > > > > > - '%-1': equal relative reciprocal Tc (MK-1 = mired).
   > > > > > > - 'K-1': equal absolute reciprocal Tc (MK-1 = mired).
   > > > > >
   > > > > > If the 'interval' element is negative, it actually represents
   > > > > > the number of intervals between Tmin, Tmax (included).

   > **uin**
   >
   > > None, optional
   > > Unit of input Tmin, Tmax (by default it is assumed to be the same
   > > as the scale 'unit') in Tc generation from tuple.

   > **seamless_stitch**
   >
   > > True, optional
   > > Determines how the Tc arrays generated for each of the 4-vector

elements are stitched together. Is overriden by the presence of a
bool as last list/tuple element in :tc4:.
For a seamless stitch, all units for all 4-vectors should be the same!!

cct_max

_CCT_MAX, optional
Limit Tc's to a maximum value of cct_max

cct_min

_CCT_MIN, optional
Limit Tc's to a minimum value of cct_max

fallback_unit

_CCT_FALLBACK_UNIT, optional
Unit to fall back on when the input unit in tc4 (of first list) is 'au'.
As there is no common distancing of the unit types ['K','%','%-1','K-1']
the Tc's are generated by dividing the min-max range into
a number of divisions, specified by the negative 3 element (or when
positive or NaN, the number of divisions is set by :fallback_divisions:)

fallback_n

_CCT_FALLBACK_N, optional
Number of divisions the min-max range is divided into, in the
fallback case in which unit=='au' and the 3e 4-vector element
is NaN or positive.

resample_tc4_array

False, optional
If False: do not resample Tc's of an ndarray input for tc4
else: divide min-max range in fallback_n intervals. Uses fallback_unit
to determine the scale for the resampling.

wl

None, optional
Wavelength for Planckian spectrum generation.
If None: use same wavelengths as CMFs in :cieobs:.

cieobs

_CIEOBS, optional
CMF set used to convert Planckian spectra to chromaticity coordinates

lut_type_def

None, placeholder
Default lut (tuple key) to read from luts_dict.

luts_dict

None, optional
Dictionary of pre-calculated luts for various cspaces and cmf sets.
    Must have structure luts_dict[cspace][cieobs][lut_label] with the
        lut part of a two-element list [lut, lut_kwargs]. It must contain

---

at the top-level a key 'wl' containing the wavelengths of the
Planckians used to generate the luts in this dictionary.

If None: the default dict for the mode is used

(e.g. _CCT_LUT['ohno2014']['lut_type_def'], for mode=='ohno2014').

lut_vars

['T','uv','uvp','uvpp','iso-T-slope'], optional
Data the lut should contain. Must follow this order
and minimum should be ['T']

cspace,cspace_kwargs

Lists with the cspace and cspace_kwargs for which luts will be generated.
Default is single chromaticity diagram in _CCT_CSPACE.

ignore_unequal_wl

False, optional
If True: ignore any differences in the wavelengths used to calculate
the lut (cfr. Planckians) from the luts_dict and the requested
wavelengths in :wl:

lut_generator_fcn

_generate_lut, optional
Lets a user specify his own lut generation function (must output a list of 1 lut).
Default is the general function. There is a specific one for
Ohno's 2014 method as that one requires a different correction factor
for each lut for the parabolic solutions. This optimized value is specified in the
second list index. (see _generate_lut_ohno2014()).

lut_generator_kwargs

{}, optional
Dict with keyword arguments specific to the (user) lut_generator_fcn.
(e.g. {'f_corr':0.9991} for _generate_lut_ohno2014())

Returns:

lut

List with an ndarray with in the columns whatever is specified in
lut_vars (Tc and uv are always present!).
Default lut_vars = ['T','uv','uvp','uvpp','iso-T-slope']
- Tc: (in K)
- u,v: chromaticity coordinates of planckians
- u'v': chromaticity coordinates of 1st derivative of the planckians.
- u",v": chromaticity coordinates of 2nd derivative of the planckians.
- slope of isotemperature lines (calculated as in Robertson, 1968).

lut_kwargs

{}
Dictionary with additional parameters related to the generation of the
lut.

---

luxpy.color.cct.**_generate_tcs**(*tc4*, *uin=None*, *seamless_stitch=True*, *cct_max=100000000000.0*, *cct_min=450*, *fallback_unit='K-1'*, *fallback_n=50*, *resample_ndarray=False*)

> Get an ndarray of Tc's obtained from a list or tuple of tc4 4-vectors (or ndarray).
>
> **Args:**
>> **tc4**
>>
>>> list or tuple of 4-vectors or ndarray.
>>>
>>> If ndarray: return tc4 limited to a cct_min-cct_max range (do nothing else).
>>>
>>> If list/tuple: e.g. (tc4_1, tc4_2, tc4_3,…) or (tc4_1, tc4_2, tc4_3,…, bool::seamless_stitch)
>>>
>>> When the last element of the list/tuple is a bool, then this specifies
>>>> how the Tc arrays generated for each of the 4-vector elements need to be stitched together. This overrides the seamless_stitch input argument.
>>>
>>> Vector elements are:
>>>> [Tmin, Tmax inclusive, Tinterval(or number of intervals), unit]
>>>> Unit specifies unit of the Tc interval, i.e. it determines the
>>>>> type of scale in which the spacing of the Tc are done.
>>>> Unit options are:
>>>>> - '%': equal relative Tc spacing (in %, cfr. (Ti+1 - Ti-1)/Ti-1).
>>>>> - 'K' equal absolute Tc spacing (in K, cfr. (Ti+1 - Ti-1).
>>>>> - '%-1': equal relative reciprocal Tc (MK-1 = mired).
>>>>> - 'K-1': equal absolute reciprocal Tc (MK-1 = mired).
>>>> If the 'interval' element is negative, it actually represents
>>>>> the number of intervals between Tmin, Tmax (included).
>>
>> **uin**
>>
>>> None, optional
>>> Unit of input Tmin, Tmax (by default it is assumed to be the same as the scale 'unit').
>>
>> **seamless_stitch**
>>
>>> True, optional
>>> Determines how the Tc arrays generated for each of the 4-vector elements are stitched together. Is overriden by the presence of a bool as last list/tuple element in :tc4:.
>>> For a seamless stitch, all units for all 4-vectors should be the same!!
>>
>> **cct_max**
>>
>>> _CCT_MAX, optional
>>> Limit Tc's to a maximum value of cct_max
>>
>> **cct_min**
>>
>>> _CCT_MIN, optional
>>> Limit Tc's to a minimum value of cct_max
>>
>> **fallback_unit**
>>
>>> _CCT_FALLBACK_UNIT, optional
>>> Unit to fall back on when the input unit in tc4 (of first list) is 'au'.
>>> As there is no common distancing of the unit types ['K','%','%-1','K-1']
>>> the Tc's are generated by dividing the min-max range into

a number of divisions, specified by the negative 3 element (or when positive or NaN, the number of divisions is set by :fallback_divisions:)

fallback_n

_CCT_FALLBACK_N, optional
Number of divisions the min-max range is divided into, in the fallback case in which unit=='au' and the 3e 4-vector element is NaN or positive.

resample_ndarray

False, optional
If False: do not resample Tc's of an ndarray input for tc4
else: divide min-max range in fallback_n intervals. Uses fallback_unit to determine the scale for the resampling.

Returns:

tcs

ndarray with Tcs

luxpy.color.cct.**_generate_lut**(*tc4, uin=None, seamless_stitch=True, fallback_unit='K-1', fallback_n='K-1', resample_ndarray=False, cct_max=100000000000.0, cct_min=450, wl=None, cieobs='1931_2', lut_vars=['T', 'uv', 'uvp', 'uvpp', 'iso-T-slope'], cspace='Yuv60', cspace_kwargs={'bwtf': {}, 'fwtf': {}}, **kwargs*)

Get an ndarray LUT for Tc's obtained from a list or tuple of tc4 4-vectors (or ndarray).

Args:

tc4

list or tuple of 4-vectors or ndarray.
If ndarray: return tc4 limited to a cct_min-cct_max range (do nothing else).
If list/tuple: e.g. (tc4_1, tc4_2, tc4_3,. . .) or (tc4_1, tc4_2, tc4_3,. . . , bool::seamless_stitch)
When the last element of the list/tuple is a bool, then this specifies

how the Tc arrays generated for each of the 4-vector elements need to be stitched together. This overrides the seamless_stitch input argument.

Vector elements are:

[Tmin, Tmax inclusive, Tinterval(or number of intervals), unit]
Unit specifies unit of the Tc interval, i.e. it determines the

type of scale in which the spacing of the Tc are done.
Unit options are:

- '%': equal relative Tc spacing (in %, cfr. (Ti+1 - Ti-1)/Ti-1).
- 'K' equal absolute Tc spacing (in K, cfr. (Ti+1 - Ti-1).
- '%-1': equal relative reciprocal Tc (MK-1 = mired).
- 'K-1': equal absolute reciprocal Tc (MK-1 = mired).

If the 'interval' element is negative, it actually represents

the number of intervals between Tmin, Tmax (included).

uin

None, optional
Unit of input Tmin, Tmax (by default it is assumed to be the same as the scale 'unit').

seamless_stitch

> True, optional
> Determines how the Tc arrays generated for each of the 4-vector
> elements are stitched together. Is overriden by the presence of a
> bool as last list/tuple element in :tc4:.
> For a seamless stitch, all units for all 4-vectors should be the same!!

cct_max

> _CCT_MAX, optional
> Limit Tc's to a maximum value of cct_max

cct_min

> _CCT_MIN, optional
> Limit Tc's to a minimum value of cct_max

fallback_unit

> _CCT_FALLBACK_UNIT, optional
> Unit to fall back on when the input unit in tc4 (of first list) is 'au'.
> As there is no common distancing of the unit types ['K','%','%-1','K-1']
> the Tc's are generated by dividing the min-max range into
> a number of divisions, specified by the negative 3 element (or when
> positive or NaN, the number of divisions is set by :fallback_divisions:)

fallback_n

> _CCT_FALLBACK_N, optional
> Number of divisions the min-max range is divided into, in the
> fallback case in which unit=='au' and the 3e 4-vector element
> is NaN or positive.

resample_tc4_array

> False, optional
> If False: do not resample Tc's of an ndarray input for tc4
> else: divide min-max range in fallback_n intervals. Uses fallback_unit
> to determine the scale for the resampling.

wl

> None, optional
> Wavelength for Planckian spectrum generation.
> If None: use same wavelengths as CMFs in :cieobs:.

cieobs

> [_CIEOBS] or list, optional
> Generate a LUT for each one in the list.
> If None: generate for all cmfs in _CMF.

lut_vars

> ['T','uv','uvp','uvpp','iso-T-slope'], optional
> Data the lut should contain. Must follow this order
> and minimum should be ['T']

> **cspace,cspace_kwargs**
>
>> Lists with the cspace and cspace_kwargs for which luts will be generated.
>> Default is single chromaticity diagram in _CCT_CSPACE.
>
> **Returns:**
>
>> **lut**
>>
>>> List with an ndarray with in the columns whatever is specified in
>>> lut_vars (Tc and uv are always present!).
>>> Default lut_vars = ['T','uv','uvp','uvpp','iso-T-slope']
>>> - Tc: (in K)
>>> - u,v: chromaticity coordinates of planckians
>>> - u'v': chromaticity coordinates of 1st derivative of the planckians.
>>> - u",v": chromaticity coordinates of 2nd derivative of the planckians.
>>> - slope of isotemperature lines (calculated as in Robertson, 1968).
>>
>> **lut_kwargs**
>>
>>> {},
>>> Dictionary with additional parameters related to the generation of the
>>> lut.

luxpy.color.cct.**_generate_lut_ohno2014**(*lut, uin=None, seamless_stitch=True, fallback_unit='K-1', fallback_n=50, resample_ndarray=False, cct_max=100000000000.0, cct_min=450, luts_dict=None, lut_type_def=None, lut_vars=['T', 'uv'], cieobs='1931_2', cspace_str=None, wl=None, ignore_unequal_wl=False, lut_generator_fcn=<function _generate_lut>, lut_generator_kwargs={}, cspace='Yuv60', cspace_kwargs={'bwtf': {}, 'fwtf': {}}, f_corr=None, ignore_f_corr_is_None=False, ignore_wl_diff=False, **kwargs*)

Lut generator function for ohno2014.

Args:

> **...**
>
>> see docstring for _generate_lut
>
> **f_corr**
>
>> Tc,x correction factor for the parabolic solution in Ohno2014.
>>> If None, it will be recalculated (note that it depends on the lut) for increased
>>> accuracy.
>
> **ignore_f_corr_is_None**
>
>> If True, ignore f_corr is None, i.e. don't re-calculate f_corr.

Returns:

> **lut**
>
>> an ndarray with the lut
>
> **dict**
>
>> a dictionary with the (re-optmized) value for f_corr and for ignore_f_cor_is_None.)

luxpy.color.cct.**_generate_lut_li2022**(*lut, uin=None, seamless_stitch=True, fallback_unit='K-1', fallback_n=50, resample_ndarray=False, cct_max=100000000000.0, cct_min=450, luts_dict=None, lut_type_def=None, lut_vars=['T', 'uv', 'uvp', 'uvpp'], cieobs='1931_2', cspace_str=None, wl=None, ignore_unequal_wl=False, lut_generator_fcn=<function _generate_lut>, lut_generator_kwargs={}, cspace='Yuv60', cspace_kwargs={'bwtf': {}, 'fwtf': {}}, f_corr=None, ignore_f_corr_is_None=False, ignore_wl_diff=False, **kwargs*)

Lut generator function for li2022 (= updated ohno2014).

Args:

> **…**
>
>> see docstring for _generate_lut
>
> **f_corr**
>
>> Tc,x correction factor for the non-triangular solution in Ohno2014.
>>> If None, it will be recalculated (note that it depends on the lut) for increased accuracy.
>
> **ignore_f_corr_is_None**
>
>> If True, ignore f_corr is None, i.e. don't re-calculate f_corr.

Returns:

> **lut**
>
>> an ndarray with the lut
>
> **dict**
>
>> a dictionary with the (re-optmized) value for f_corr and for ignore_f_cor_is_None.)

luxpy.color.cct.**xyz_to_cct_ohno2011**(*xyz*)

> Calculate cct and Duv from CIE 1931 2° xyz following Ohno (2011).

Args:

> **xyz**
>
>> ndarray with CIE 1931 2° X,Y,Z tristimulus values

Returns:

> **cct, duv**
>
>> ndarrays with correlated color temperatures and distance to blackbody locus in CIE 1960 uv

> **References:** 1. Ohno, Y. (2011). Calculation of CCT and Duv and Practical Conversion Formulae. CORM 2011 Conference, Gaithersburg, MD, May 3-5, 2011

4.4.4 cct/robertson1968

> **py**
>
>> - __init__.py
>>
>> - robertson1968.py
>
> **namespace** luxpy.color.cct.robertson1968

Standalone (no luxpy required) module with (updated, 2022) Robertson1968 CCT functions

(includes correction near slope-sign-change of iso-temperature-lines)

> **cct_to_xyz()** Calculates xyz from CCT, Duv by estimating the line perpendicular to the planckian locus (=iso-T line).

> **cct_to_xyz()** Calculates xyz from CCT, Duv [_CCT_MIN < CCT < _CCT_MAX]

References: 1. Robertson, A. R. (1968). Computation of Correlated Color Temperature and Distribution Temperature. Journal of the Optical Society of America, 58(11), 1528–1535.

2. Smet K.A.G., Royer M., Baxter D., Bretschneider E., Esposito E., Houser K., Luedtke W., Man K., Ohno Y. (2022), Recommended method for determining the correlated color temperature and distance from the Planckian Locus of a light source (in preparation, LEUKOS?)

3. Baxter D., Royer M., Smet K.A.G. (2022) Modifications of the Robertson Method for Calculating Correlated Color Temperature to Improve Accuracy and Speed (in preparation, LEUKOS?)

`luxpy.color.cct.robertson1968.`**`save_pkl`**(*filename*, *obj*)

> Save an object in a pickle file.

> **Args:**

>> **filename**

>>> str with filename of pickle file.

>> **obj**

>>> python object to save

> **Returns:**

>> **None**

`luxpy.color.cct.robertson1968.`**`load_pkl`**(*filename*)

> Load the object in a pickle file.

> **Args:**

>> **filename**

>>> str with filename of pickle file.

> **Returns:**

>> **obj**

>>> loaded python object

`luxpy.color.cct.robertson1968.`**`get_tcs4`**(*tc4*, *cct_min=450*, *cct_max=100000000000.0*)

> Generate list of Tc of Planckians from (Tmin, Tmax inclusive, Tincrement, unit)

> **Args:**

>> **tc4**

>>> 4-element list or tuple

>>> Elements are: [Tmin, Tmax inclusive, Tincrement, unit]

>>> > Unit specifies unit of the Tc interval, i.e. it determines the

>>> > > type of scale in which the spacing of the Tc are done.

>>> > Unit options are:

>>> > > - '%': equal relative Tc spacing (in %, cfr. (Ti+1 - Ti-1)/Ti-1).

>>> > > - 'K' equal absolute Tc spacing (in K, cfr. (Ti+1 - Ti-1).

>>> > > - '%-1': equal relative reciprocal Tc (MK-1 = mired).

>>> > > - 'K-1': equal absolute reciprocal Tc (MK-1 = mired).

>>> > If the 'increment' element is negative, it actually represents

---

the number of intervals between Tmin, Tmax (included).

> **cct_min**
>
>> _CCT_MIN, optional
>> Limit Tc's to a minimum value of cct_min
>
> **cct_max**
>
>> _CCT_MAX, optional
>> Limit Tc's to a maximum value of cct_max

Returns:

> **Tcs**
>
>> ndarray [N,1] of ccts.

`luxpy.color.cct.robertson1968.`**`calculate_lut`**(*ccts, cieobs, lut_vars=['T', 'uv', 'uvp', 'uvpp', 'iso-T-slope'], cct_min=450, cct_max=100000000000.0*)

Function that calculates a LUT for the specified calculation method for the input ccts. Calculation is performed for CMF set specified in cieobs and in the chromaticity diagram in cspace.

Args:

> **ccts**
>
>> ndarray [Nx1] or str or 4-element tuple
>> If ndarray: list of ccts for which to (re-)calculate the LUTs.
>> If str: path to file containing CCTs (no header; sep = ',')
>> If 4-element tuple: generate ccts from (Tmin, Tmax, increment, unit) specifier
>
> **cieobs**
>
>> None or str, optional
>> str specifying cmf set.
>
> **lut_vars**
>
>> ['T','uv','uvp','uvpp','iso-T-slope'], optional
>> Data the lut should contain. Must follow this order
>> and minimum should be ['T']
>
> **cct_min**
>
>> _CCT_MIN, optional
>> Limit Tc's to a minimum value of cct_min
>
> **cct_max**
>
>> _CCT_MAX, optional
>> Limit Tc's to a maximum value of cct_max

Returns:

> **returns**
>
>> **lut**
>>
>>> ndarray with T, u, v, u', v', u", v", slope (note ':1st deriv., ":2nd deriv.).

`luxpy.color.cct.robertson1968.`**`xyz_to_cct`**(*xyzw, is_uv_input=False, cieobs='1931_2', out='cct', lut=None, apply_newton_raphson=False, rtol=1e-10, atol=0.1, max_iter=10, split_calculation_at_N=25, use_fast_duv=True*)

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv(distance above (> 0) or below (< 0) the Planckian locus) using Robertson's 1968 search method.

Args:

> **xyzw**
>
>> ndarray of tristimulus values
>
> **is_uv_input**
>
>> False, optional
>> If True: xyzw contain uv input data, not xyz data!
>
> **cieobs**
>
>> _CCT_CIEOBS, optional
>> CMF set used to calculated xyzw.
>
> **out**
>
>> 'cct' (or 1), optional
>> Determines what to return.
>> Other options: 'duv' (or -1), 'cct,duv'(or 2), "[cct,duv]" (or -2)
>
> **rtol**
>
>> 1e-10, float, optional
>> Stop search when cct a relative tolerance is reached.
>> The relative tolerance is calculated as dCCT/CCT_est,
>> with CCT_est the current intermediate estimate in the
>> search and with dCCT the difference between
>> the present and former estimates.
>
> **atol**
>
>> 0.1, optional
>> Stop search when cct a absolute tolerance (K) is reached.
>
> **lut**
>
>> None, optional
>> Look-Up-Table with Ti, u,v,u',v',u'',v'',slope values of Planckians.
>> Options:
>>
>>> - None: defaults to the lut specified in _CCT_LUT['lut_type_def'].
>>> - tuple: new lut will be generated from scratch using the info in the tuple.
>>> - ndarray [Nx1]: list of luts for which to generate a lut
>>> - ndarray [Nxn] with n>3: pre-calculated lut (last col must contain slope of the isotemperature lines).
>
> **apply_newton_raphson**
>
>> False, optional
>> If False: use only the Robertson1968 base method.
>>
>>> Accuracy depends on CCT of test source and the location
>>> and spacing of the CCTs in the list.
>>
>> If True: improve estimate of base method using a follow-up
>>
>>> newton-raphson method.
>>> When the CCT for multiple source is calculated in one go,
>>> then the atol and rtol values have to be met for all!

max_iter

  _CCT_MAX_ITER, optional

  Maximum number of iterations used by the cascading-lut or newton-raphson methods.

split_calculation_at_N

  _CCT_SPLIT_CALC_AT_N, optional

  Split calculation when xyzw.shape[0] > split_calculation_at_N.

  Splitting speeds up the calculation. If None: no splitting is done.

use_fast_duv

  _CCT_FAST_DUV, optional

  If True: use a fast estimator of the Duv

    (one that avoids calculation of Planckians and uses the former

      best estimate's u,v coordinates. This method is accurate enough

      when the atol is small enough -> as long as abs(T-T_former)<=1K

      the Duv estimate should be ok.)

Returns:

 returns

  ndarray with:

   cct: out == 'cct' (or 1)

   duv: out == 'duv' (or -1)

   cct, duv: out == 'cct,duv' (or 2)

   [cct,duv]: out == "[cct,duv]" (or -2)

Note: 1. Out-of-lut CCTs are encoded as negative CCTs (with as absolute value the value of the closest CCT from the lut.)

References: 1. Robertson, A. R. (1968). Computation of Correlated Color Temperature and Distribution Temperature. Journal of the Optical Society of America, 58(11), 1528–1535.

2. Baxter D., Royer M., Smet K.A.G. (2022) Modifications of the Robertson Method for Calculating Correlated Color Temperature to Improve Accuracy and Speed (in preparation, LEUKOS?)

3. Li, C., Cui, G., Melgosa, M., Ruan, X., Zhang, Y., Ma, L., Xiao, K., & Luo, M. R. (2016). Accurate method for computing correlated color temperature. Optics Express, 24(13), 14066–14078.

luxpy.color.cct.robertson1968.**xyz_to_duv**(*xyzw*, *out='duv'*, ***kwargs*)

 Wraps xyz_to_cct, but with duv output. For kwargs info, see xyz_to_cct.

luxpy.color.cct.robertson1968.**cct_to_xyz**(*ccts*,   *duv=None*,   *cct_offset=None*,   *cieobs='1931_2'*)

 Convert correlated color temperature (550 K <= CCT <= 1e11 K) and Duv (distance above (>0) or below (<0) the Planckian locus) to XYZ tristimulus values.

Finds xyzw_estimated by determining the iso-temperature line

  (= line perpendicular to the Planckian locus):

  Option 1 (fastest):

    First, the angle between the coordinates corresponding to ccts

    and ccts-cct_offset are calculated, then 90° is added, and finally

    the new coordinates are determined, while taking sign of duv into account.

  Option 2 (slowest, about 55% slower):

    Calculate the slope of the iso-T-line directly using the Planckian

    spectrum and its derivative.

Args:

  ccts

    ndarray [N,1] of cct values

  duv

    None or ndarray [N,1] of duv values, optional
    Note that duv can be supplied together with cct values in :ccts:
    as ndarray with shape [N,2].

  cct_offset

    None, optional
    If None: use option 2 (direct iso-T slope calculation, more accurate,
      but slower: about 1.55 slower)
    else: use option 1 (estimate slope from 90° + angle of small cct_offset)

  cieobs

    _CCT_CIEOBS, optional
    CMF set used to calculated xyzw.

  wl

    None, optional
    Wavelengths used when calculating Planckian radiators.
    If None: use same wavelengths as CMFs in :cieobs:.

Returns:

  returns

    ndarray with estimated XYZ tristimulus values

Note: 1. If duv is not supplied (:ccts:.shape is (N,1) and :duv: is None), source is assumed to be on the Planckian locus. 2. Minimum CCT is 550 K (lower than 550 K, some negative Duv values will result in coordinates outside of the Spectrum Locus !!!)

4.4.5 cat/

py

    • __init__.py

    • chromaticadaptation.py

namespace luxpy.cat

cat: Module supporting chromatic adaptation transforms (corresponding colors)

  _WHITE_POINT default adopted white point

  _LA default luminance of the adaptation field

  _MCATS default chromatic adaptation sensor spaces

    • 'hpe': Hunt-Pointer-Estevez: R. W. G. Hunt, The Reproduction of Colour: Sixth Edition, 6th ed. Chichester, UK: John Wiley & Sons Ltd, 2004.

    • 'cat02': from ciecam02: CIE159-2004, "A Colour Apperance Model for Color Management System: CIECAM02," CIE, Vienna, 2004.

- 'cat02-bs': cat02 adjusted to solve yellow-blue problem (last line = [0 0 1]): Brill MH, Süsstrunk S. Repairing gamut problems in CIECAM02: A progress report. Color Res Appl 2008;33(5), 424–426.

- 'cat02-jiang': cat02 modified to solve yb-probem + purple problem: Jun Jiang, Zhifeng Wang,M. Ronnier Luo,Manuel Melgosa,Michael H. Brill,Changjun Li, Optimum solution of the CIECAM02 yellow–blue and purple problems, Color Res Appl 2015: 40(5), 491-503.

- 'kries'

- 'judd-1945': from CIE16-2004, Eq.4, a23 modified from 0.1 to 0.1020 for increased accuracy

- 'bfd': bradford transform : G. D. Finlayson and S. Susstrunk, "Spectral sharpening and the Bradford transform," 2000, vol. Proceeding, pp. 236–242.

- 'sharp': sharp transform: S. Süsstrunk, J. Holm, and G. D. Finlayson, "Chromatic adaptation performance of different RGB sensors," IS&T/SPIE Electronic Imaging 2001: Color Imaging, vol. 4300. San Jose, CA, January, pp. 172–183, 2001.

- 'cmc': C. Li, M. R. Luo, B. Rigg, and R. W. G. Hunt, "CMC 2000 chromatic adaptation transform: CMCCAT2000," Color Res. Appl., vol. 27, no. 1, pp. 49–58, 2002.

- 'ipt': F. Ebner and M. D. Fairchild, "Development and testing of a color space (IPT) with improved hue uniformity," in IS&T 6th Color Imaging Conference, 1998, pp. 8–13.

- 'lms':

- 'bianco': S. Bianco and R. Schettini, "Two new von Kries based chromatic adaptation transforms found by numerical optimization," Color Res. Appl., vol. 35, no. 3, pp. 184–192, 2010.

- 'bianco-pc': S. Bianco and R. Schettini, "Two new von Kries based chromatic adaptation transforms found by numerical optimization," Color Res. Appl., vol. 35, no. 3, pp. 184–192, 2010.

- 'cat16': C. Li, Z. Li, Z. Wang, Y. Xu, M. R. Luo, G. Cui, M. Melgosa, M. H. Brill, and M. Pointer, "Comprehensive color solutions: CAM16, CAT16, and CAM16-UCS," Color Res. Appl., p. n/a–n/a.

check_dimensions() Check if dimensions of data and xyzw match.

get_transfer_function()

Calculate the chromatic adaptation diagonal matrix transfer function Dt.
Default = 'vonkries' (others: 'rlab', see Fairchild 1990)

smet2017_D()

Calculate the degree of adaptation based on chromaticity.
Smet, K.A.G.*, Zhai, Q., Luo, M.R., Hanselaer, P., (2017), Study of chromatic adaptation using memory color matches, Part II: colored illuminants. Opt. Express, 25(7), pp. 8350-8365

get_degree_of_adaptation()

Calculates the degree of adaptation.

D passes either right through or D is calculated following some D-function (Dtype) published in literature (cat02, cat16, cmccat, smet2017) or set manually.

parse_x1x2_parameters() local helper function that parses input parameters and makes them the target_shape for easy calculation

apply() Calculate corresponding colors by applying a von Kries chromatic adaptation transform (CAT), i.e. independent rescaling of 'sensor sensitivity' to data to adapt from current adaptation conditions (1) to the new conditions (2).

---

luxpy.color.cat.**check_dimensions**(*data*, *xyzw*, *caller='cat.apply()'*)
Check if dimensions of data and xyzw match.

Does nothing when they do, but raises error if dimensions don't match.

Args:

> **data**
>
>> ndarray with color data.
>
> **xyzw**
>
>> ndarray with white point tristimulus values.
>
> **caller**
>
>> str with caller function for error handling, optional

Returns:

> **returns**
>
>> ndarray with input color data,
>> Raises error if dimensions don't match.

luxpy.color.cat.**get_transfer_function**(*cattype='vonkries'*, *catmode='1>0>2'*, *lmsw1=None*, *lmsw2=None*, *lmsw0=array([[100, 100, 100]])*, *D10=1.0*, *D20=1.0*, *La1=100.0*, *La2=100.0*, *La0=100.0*)
Calculate the chromatic adaptation diagonal matrix transfer function Dt.
Args:

> **cattype**
>
>> 'vonkries' (others: 'rlab', see Farchild 1990), optional
>
> **catmode**
>
>> '1>0>2, optional
>>> -'1>0>2': Two-step CAT
>>>> from illuminant 1 to baseline illuminant 0 to illuminant 2.
>>> -'1>0': One-step CAT
>>>> from illuminant 1 to baseline illuminant 0.
>>> -'0>2': One-step CAT
>>>> from baseline illuminant 0 to illuminant 2.
>
> **lmsw1**
>
>> None, depending on :catmode: optional
>
> **lmsw2**

> None, depending on :catmode: optional

lmsw0

> _WHITE_POINT, optional

D10

> 1.0, optional
> Degree of adaptation for ill. 1 to ill. 0

D20

> 1.0, optional
> Degree of adaptation for ill. 2 to ill. 0

La1

> luxpy._LA, optional
> Adapting luminance under ill. 1

La2

> luxpy._LA, optional
> Adapting luminance under ill. 2

La0

> luxpy._LA, optional
> Adapting luminance under baseline ill. 0

Returns:

> **Dt**
>
> ndarray (diagonal matrix)

luxpy.color.cat.**get_degree_of_adaptation**(*Dtype=None*, ***kwargs*)

> Calculates the degree of adaptation according to some function published in literature.

Args:

> **Dtype**
>
> > None, optional
> >
> > > If None: kwargs should contain 'D' with value.
> > >
> > > If 'manual: kwargs should contain 'D' with value.
> >
> > If 'cat02' or 'cat16': kwargs should contain keys 'F' and 'La'.
> >
> > > Calculate D according to CAT02 or CAT16 model:
> > >
> > > > D = F*(1-(1/3.6)*numpy.exp((-La-42)/92))
> >
> > If 'cmc': kwargs should contain 'La', 'La0'(or 'La2') and 'order'
> >
> > > for 'order' = '1>0': 'La' is set La1 and 'La0' to La0.
> > >
> > > for 'order' = '0>2': 'La' is set La0 and 'La0' to La1.
> > >
> > > for 'order' = '1>2': 'La' is set La1 and 'La2' to La0.
> > >
> > > D is calculated as follows:
> > >
> > > > D = 0.08*numpy.log10(La1+La0)+0.76-0.45*(La1-La0)/(La1+La0)
> >
> > If 'smet2017': kwargs should contain 'xyzw' and 'Dmax'
> >
> > > (see Smet2017_D for more details).
> >
> > If "? user defined", then D is calculated by:
> >
> > > D = ndarray(eval(:Dtype:))

Returns:

> **D**

---

ndarray with degree of adaptation values.

Notes:

1. D passes either right through or D is calculated following some D-function (Dtype) published in literature.
2. D is limited to values between zero and one
3. If kwargs do not contain the required parameters, an exception is raised.

luxpy.color.cat.**smet2017_D**(*xyzw*, *Dmax=None*)

Calculate the degree of adaptation based on chromaticity following Smet et al. (2017)

Args:

> **xyzw**
>
> > ndarray with white point data (CIE 1964 10° XYZs!!)
>
> **Dmax**
>
> > None or float, optional
> > Defaults to 0.6539 (max D obtained under experimental conditions,
> > but probably too low due to dark surround leading to incomplete
> > chromatic adaptation even for neutral illuminants
> > resulting in background luminance (fov~50Â°) of 760 cd/mÂ^2))

Returns:

> **D**
>
> > ndarray with degrees of adaptation

References: 1. Smet, K.A.G.*, Zhai, Q., Luo, M.R., Hanselaer, P., (2017), Study of chromatic adaptation using memory color matches, Part II: colored illuminants, Opt. Express, 25(7), pp. 8350-8365.

luxpy.color.cat.**parse_x1x2_parameters**(*x, target_shape, catmode, expand_2d_to_3d=None, default=[1.0, 1.0]*)

Parse input parameters x and make them the target_shape for easy calculation.

Input in main function can now be a single value valid for all xyzw or an array with a different value for each xyzw.

Args:

> **x**
>
> > list[float, float] or ndarray
>
> **target_shape**
>
> > tuple with shape information
>
> **catmode**
>
> > '1>0>2, optional
> > > -'1>0>2': Two-step CAT
> > > > from illuminant 1 to baseline illuminant 0 to illuminant 2.
> > > -'1>0': One-step CAT
> > > > from illuminant 1 to baseline illuminant 0.
> > > -'0>2': One-step CAT
> > > > from baseline illuminant 0 to illuminant 2.
>
> **expand_2d_to_3d**
>
> > None, optional

[will be removed in future, serves no purpose]

Expand :x: from 2 to 3 dimensions.

default

[1.0,1.0], optional

Default values for :x:

Returns:

returns

(ndarray, ndarray) for x10 and x20

luxpy.color.cat.**apply**(*data,   n_step=2,   catmode=None,   cattype='vonkries',   xyzw1=None,*
xyzw2=None,   xyzw0=None,   D=None,   mcat=['cat02'],   normxyz0=None,
outtype='xyz', La=None, F=None, Dtype=None)

Calculate corresponding colors by applying a von Kries chromatic adaptation transform (CAT), i.e. independent rescaling of 'sensor sensitivity' to data to adapt from current adaptation conditions (1) to the new conditions (2).

Args:

data

ndarray of tristimulus values (can be NxMx3)

n_step

2, optional

Number of step in CAT (1: 1-step, 2: 2-step)

catmode

None, optional

- None: use :n_step: to set mode: 1 = '1>2', 2:'1>0>2'

-'1>0>2': Two-step CAT

from illuminant 1 to baseline illuminant 0 to illuminant 2.

-'1>2': One-step CAT

from illuminant 1 to illuminant 2.

-'1>0': One-step CAT

from illuminant 1 to baseline illuminant 0.

-'0>2': One-step CAT

from baseline illuminant 0 to illuminant 2.

cattype

'vonkries' (others: 'rlab', see Farchild 1990), optional

xyzw1

None, depending on :catmode: optional (can be Mx3)

xyzw2

None, depending on :catmode: optional (can be Mx3)

xyzw0

None, depending on :catmode: optional (can be Mx3)

D

None, optional

Degrees of adaptation. Defaults to [1.0, 1.0].

La

None, optional
Adapting luminances.
If None: xyz values are absolute or relative.
If not None: xyz are relative.

F

None, optional
Surround parameter(s) for CAT02/CAT16 calculations
(:Dtype: == 'cat02' or 'cat16')
Defaults to [1.0, 1.0].

Dtype

None, optional
Type of degree of adaptation function from literature
See luxpy.cat.get_degree_of_adaptation()

mcat

[_MCAT_DEFAULT], optional
List[str] or List[ndarray] of sensor space matrices for each
condition pair. If len(:mcat:) == 1, the same matrix is used.

normxyz0

None, optional
Set of xyz tristimulus values to normalize the sensor space matrix to.

outtype

'xyz' or 'lms', optional
- 'xyz': return corresponding tristimulus values
- 'lms': return corresponding sensor space excitation values
(e.g. for further calculations)

Returns:

returns

ndarray with corresponding colors

Reference: 1. Smet, K. A. G., & Ma, S. (2020). Some concerns regarding the CAT16 chromatic adaptation
transform. Color Research & Application, 45(1), 172–177.

luxpy.color.cat.**apply_vonkries1**(*xyz*, *xyzw1*, *xyzw2*, *D=1*, *mcat=None*, *invmcat=None*,
in_type='xyz', *out_type='xyz'*, *use_Yw=False*)
Apply a 1-step von kries chromatic adaptation transform.
Args:

xyz

ndarray with sample tristimulus or cat-sensor values

xyzw1

ndarray with white point tristimulus or cat-sensor values of illuminant 1

xyzw2

ndarray with white point tristimulus or cat-sensor values of illuminant 2

D

1, optional

Degree of chromatic adaptation

mcat

None, optional
Specifies CAT sensor space.
- options:
- None defaults to luxpy.cat._MCAT_DEFAULT
- str: see see luxpy.cat._MCATS.keys() for options
(details on type, ?luxpy.cat)
- ndarray: matrix with sensor primaries

invmcat

None,optional
Pre-calculated inverse mcat.
If None: calculate inverse of mcat.

in_type

'xyz', optional
Input type ('xyz', 'rgb') of data in xyz, xyzw1, xyzw2

out_type

'xyz', optional
Output type ('xyz', 'rgb') of corresponding colors

use_Yw

False, optional
Use CAT version with Yw factors included (but this results in
potential wrong predictions, see Smet & Ma (2020)).

Returns:

xyzc

ndarray with corresponding colors.

Reference: 1. Smet, K. A. G., & Ma, S. (2020). Some concerns regarding the CAT16 chromatic adaptation
transform. Color Research & Application, 45(1), 172–177.

luxpy.color.cat.**apply_vonkries2**(*xyz*, *xyzw1*, *xyzw2*, *xyzw0=None*, *D=1*, *mcat=None*, *invm-
cat=None*, *in_type='xyz'*, *out_type='xyz'*, *use_Yw=False*)
Apply a 2-step von kries chromatic adaptation transform.

Args:

xyz

ndarray with sample tristimulus or cat-sensor values

xyzw1

ndarray with white point tristimulus or cat-sensor values of illuminant 1

xyzw2

ndarray with white point tristimulus or cat-sensor values of illuminant 2

xyzw0

None, optional
ndarray with white point tristimulus or cat-sensor values of baseline illuminant 0
None: defaults to EEW.

D

[1,1], optional

Degree of chromatic adaptations (Ill.1–>Ill.0, Ill.2.–>Ill.0)

mcat

None, optional

Specifies CAT sensor space.

- options:

- None defaults to luxpy.cat._MCAT_DEFAULT

- str: see see luxpy.cat._MCATS.keys() for options

(details on type, ?luxpy.cat)

- ndarray: matrix with sensor primaries

invmcat

None,optional

Pre-calculated inverse mcat.

If None: calculate inverse of mcat.

in_type

'xyz', optional

Input type ('xyz', 'rgb') of data in xyz, xyzw1, xyzw2

out_type

'xyz', optional

Output type ('xyz', 'rgb') of corresponding colors

use_Yw

False, optional

Use CAT version with Yw factors included (but this results in

potential wrong predictions, see Smet & Ma (2020)).

Returns:

xyzc

ndarray with corresponding colors.

Reference: 1. Smet, K. A. G., & Ma, S. (2020). Some concerns regarding the CAT16 chromatic adaptation
transform. Color Research & Application, 45(1), 172–177.

luxpy.color.cat.**apply_vonkries**(*xyz*, *xyzw1*, *xyzw2*, *xyzw0=None*, *D=1*, *n_step=2*, *cat-
mode='1>0>2'*, *mcat=None*, *invmcat=None*, *in_type='xyz'*,
out_type='xyz', *use_Yw=False*)

Apply a 1-step or 2-step von kries chromatic adaptation transform.

Args:

xyz

ndarray with sample tristimulus or cat-sensor values

xyzw1

ndarray with white point tristimulus or cat-sensor values of illuminant 1

xyzw2

ndarray with white point tristimulus or cat-sensor values of illuminant 2

xyzw0

> None, optional
> ndarray with white point tristimulus or cat-sensor values of baseline illuminant 0
> None: defaults to EEW.

D

> [1,1], optional
> Degree of chromatic adaptations (Ill.1–>Ill.0, Ill.2.–>Ill.0)

n_step

> 2, optional
> Number of step in CAT (1: 1-step, 2: 2-step)

catmode

> None, optional
>> - None: use :n_step: to set mode: 1 = '1>2', 2:'1>0>2'
>> -'1>0>2': Two-step CAT
>>> from illuminant 1 to baseline illuminant 0 to illuminant 2.
>> -'1>2': One-step CAT
>>> from illuminant 1 to illuminant 2.
>> -'1>0': One-step CAT
>>> from illuminant 1 to baseline illuminant 0.
>> -'0>2': One-step CAT
>>> from baseline illuminant 0 to illuminant 2.

mcat

> None, optional
> Specifies CAT sensor space.
> - options:
>> - None defaults to luxpy.cat._MCAT_DEFAULT
>> - str: see see luxpy.cat._MCATS.keys() for options
>>> (details on type, ?luxpy.cat)
>> - ndarray: matrix with sensor primaries

invmcat

> None,optional
> Pre-calculated inverse mcat.
> If None: calculate inverse of mcat.

in_type

> 'xyz', optional
> Input type ('xyz', 'rgb') of data in xyz, xyzw1, xyzw2

out_type

> 'xyz', optional
> Output type ('xyz', 'rgb') of corresponding colors

use_Yw

> False, optional
> Use CAT version with Yw factors included (but this results in
> potential wrong predictions, see Smet & Ma (2020)).

Returns:

> **xyzc**
>
>> ndarray with corresponding colors.

Reference: 1. Smet, K. A. G., & Ma, S. (2020). Some concerns regarding the CAT16 chromatic adaptation transform. Color Research & Application, 45(1), 172–177.

luxpy.color.cat.**apply_ciecat94**(*xyz*, *xyzw*, *xyzwr=None*, *E=1000*, *Er=1000*, *Yb=20*, *D=1*, *cat94_old=True*)

Calculate corresponding color tristimulus values using the CIECAT94 chromatic adaptation transform.

Args:

> **xyz**
>
>> ndarray with sample 1931 2° XYZ tristimulus values under the test illuminant
>
> **xyzw**
>
>> ndarray with white point tristimulus values of the test illuminant
>
> **xyzwr**
>
>> None, optional
>> ndarray with white point tristimulus values of the reference illuminant
>> None defaults to D65.
>
> **E**
>
>> 100, optional
>> Illuminance (lx) of test illumination
>
> **Er**
>
>> 63.66, optional
>> Illuminance (lx) of the reference illumination
>
> **Yb**
>
>> 20, optional
>> Relative luminance of the adaptation field (background)
>
> **D**
>
>> 1, optional
>> Degree of chromatic adaptation.
>> For object colours D = 1,
>> and for luminous colours (typically displays) D=0

Returns:

> **xyzc**
>
>> ndarray with corresponding tristimlus values.

Reference:

> 1. CIE160-2004. (2004). A review of chromatic adaptation transforms (Vols. CIE160-200). CIE.

4.4.6 cam/

> **py**
>
>> - __init__.py
>> - colorappearancemodels.py
>> - helpers.py

- utils.py

- ciecam02.py

- cam02ucs.py

- ciecam16.py

- cam16ucs.py

- cam15u

- sww2016.py

- cam18sl.py

- camjabz.py

- zcam.py

- cmf_translator_sww2021

namespace luxpy.cam

cam: sub-package with color appearance models

_UNIQUE_HUE_DATA

database of unique hues with corresponding
Hue quadratures and eccentricity factors
for ciecam02, ciecam16, ciecam97s, cam15u, cam18sl)

_SURROUND_PARAMETERS

database of surround param. c, Nc, F and FLL
for ciecam02, ciecam16, ciecam97s and cam15u.

_NAKA_RUSHTON_PARAMETERS

database with parameters (n, sig, scaling and noise)
for the Naka-Rushton function:
$NK(x) = sign(x) * scaling * ((abs(x)**n) / ((abs(x)**n) + (sig**n))) + noise$

_CAM_UCS_PARAMETERS

database with parameters specifying the conversion
> from ciecamX to:
>> camXucs (uniform color space),
>> camXlcd (large color diff.),
>> camXscd (small color diff).

_CAM15U_PARAMETERS database with CAM15u model parameters.

_CAM_SWW16_PARAMETERS cam_sww16 model parameters.

_CAM18SL_PARAMETERS database with CAM18sl model parameters

_CAM_DEFAULT_WHITE_POINT Default internal reference white point (xyz)

_CAM_DEFAULT_CONDITIONS Default CAM model parameters for model.

_CAM_AXES dict with list[str,str,str] containing axis labels of defined cspaces.

deltaH() Compute a hue difference, $dH = 2*C1*C2*sin(dh/2)$.

naka_rushton() applies a Naka-Rushton function to the input

hue_angle() calculates a positive hue angle

hue_quadrature() calculates the Hue quadrature from the hue.

ciecam02()

> calculates ciecam02 output
>
> N. Moroney, M. D. Fairchild, R. W. G. Hunt, C. Li, M. R. Luo, and T. Newman, "The CIECAM02 color appearance model," IS&T/SID Tenth Color Imaging Conference. p. 23, 2002.

cam16()

> calculates cam16 output
>
> C. Li, Z. Li, Z. Wang, Y. Xu, M. R. Luo, G. Cui, M. Melgosa, M. H. Brill, and M. Pointer, "Comprehensive color solutions: CAM16, CAT16, and CAM16-UCS," Color Res. Appl., p. n/a–n/a.

cam02ucs()

> calculates ucs (or lcd, scd) output based on ciecam02
>
> (forward + inverse available)
>
> M. R. Luo, G. Cui, and C. Li, "Uniform colour spaces based on CIECAM02 colour appearance model," Color Res. Appl., vol. 31, no. 4, pp. 320–330, 2006.

cam16ucs()

> calculates ucs (or lcd, scd) output based on cam16
>
> > (forward + inverse available)
>
> C. Li, Z. Li, Z. Wang, Y. Xu, M. R. Luo, G. Cui, M. Melgosa, M. H. Brill, and M. Pointer, "Comprehensive color solutions: CAM16, CAT16, and CAM16-UCS," Color Res. Appl.

cam15u()

> calculates the output for the CAM15u model for self-luminous unrelated stimuli.
>
> M. Withouck, K. A. G. Smet, W. R. Ryckaert, and P. Hanselaer, "Experimental driven modelling of the color appearance of unrelated self-luminous stimuli: CAM15u," Opt. Express, vol. 23, no. 9, pp. 12045–12064, 2015.
>
> M. Withouck, K. A. G. Smet, and P. Hanselaer, (2015), "Brightness prediction of different sized unrelated self-luminous stimuli," Opt. Express, vol. 23, no. 10, pp. 13455–13466.

cam_sww16()

> A simple principled color appearance model based on a mapping of the Munsell color system.
>
> Smet, K. A. G., Webster, M. A., & Whitehead, L. A. (2016). "A simple principled approach for modeling and understanding uniform color metrics." Journal of the Optical Society of America A, 33(3), A319–A331.

cam18sl()

> calculates the output for the CAM18sl model for self-luminous related stimuli.
>
> Hermans, S., Smet, K. A. G., & Hanselaer, P. (2018). "Color appearance model for self-luminous stimuli." Journal of the Optical Society of America A, 35(12), 2000–2009.

camXucs() Wraps ciecam02(), ciecam16(), cam02ucs(), cam16ucs().

specific wrappers in the 'xyz_to_cspace()' and 'cpsace_to_xyz()' format

> 'xyz_to_jabM_ciecam02', 'jabM_ciecam02_to_xyz',
> 'xyz_to_jabC_ciecam02', 'jabC_ciecam02_to_xyz',
> 'xyz_to_jabM_ciecam16', 'jabM_ciecam16_to_xyz',
> 'xyz_to_jabC_ciecam16', 'jabC_ciecam16_to_xyz',
> 'xyz_to_jabz', 'jabz_to_xyz',
> 'xyz_to_jabM_camjabz', 'jabM_camjabz_to_xyz',
> 'xyz_to_jabC_camjabz', 'jabC_camjabz_to_xyz',
> 'xyz_to_jab_cam02ucs', 'jab_cam02ucs_to_xyz',
> 'xyz_to_jab_cam02lcd', 'jab_cam02lcd_to_xyz',
> 'xyz_to_jab_cam02scd', 'jab_cam02scd_to_xyz',
> 'xyz_to_jab_cam16ucs', 'jab_cam16ucs_to_xyz',
> 'xyz_to_jab_cam16lcd', 'jab_cam16lcd_to_xyz',
> 'xyz_to_jab_cam16scd', 'jab_cam16scd_to_xyz',
> 'xyz_to_qabW_cam15u', 'qabW_cam15u_to_xyz',
> 'xyz_to_lab_cam_sww16','lab_cam_sww16_to_xyz',
> 'xyz_to_qabM_cam18sl', 'qabM_cam18sl_to_xyz',
> 'xyz_to_qabS_cam18sl', 'qabS_cam18sl_to_xyz',

_update_parameter_dict() Get parameter dict and update with values in args dict

_setup_default_adaptation_field() Setup a default illuminant adaptation field with Lw = 100 cd/m^2 for selected CIE observer.

_massage_input_and_init_output() Redimension input data to ensure most they have the appropriate sizes for easy and efficient looping.

_massage_output_data_to_original_shape() Massage output data to restore original shape of original CAM input.

_get_absolute_xyz_xyzw() Calculate absolute xyz tristimulus values of stimulus and white point from spectral input or convert relative xyz values to absolute ones.

_simple_cam() An example CAM illustration the usage of the functions in luxpy.cam.helpers

Module for CAM "front-end" cmf adaptation

translate_cmfI_to_cmfS()

> Using smooth RGB primaries, translate input data (spectral or tristimlus)
> for an indivual observer to the expected tristimulus values for a standard observer.

get_conversion_matrix()

> Using smooth RGB primaries, get the 'translator' matrix to convert
> tristimulus values calculated using an individual observer's
> color matching functions (cmfs) to those calculated using the cmfs of
> a standard observer.

get_rgb_smooth_prims() Get smooth R, G, B primaries with specified wavelength range

_R,_G,_B precalculated smooth primaries with [360,830,1] wavelength range.

luxpy.color.cam.**hue_angle**(*a*, *b*, *htype='deg'*)
    Calculate positive hue angle (0°-360° or 0 - 2*pi rad.) from opponent signals a and b.

Args:

> **a**
>
>> ndarray of a-coordinates
>
> **b**
>
>> ndarray of b-coordinates
>
> **htype**
>
>> 'deg' or 'rad', optional
>>> - 'deg': hue angle between 0° and 360°
>>> - 'rad': hue angle between 0 and 2pi radians

Returns:

> **returns**
>
>> ndarray of positive hue angles.

luxpy.color.cam.**naka_rushton**(*data*, *sig=2.0*, *n=0.73*, *scaling=1.0*, *noise=0.0*, *forward=True*)

> Apply a Naka-Rushton response compression (n) and an adaptive shift (sig).

> NK(x) = sign(x) * scaling * ((abs(x)**n) / ((abs(x)**n) + (sig**n))) + noise

> **Args:**
>
>> **data**
>>
>>> float or ndarray
>>
>> **sig**
>>
>>> 2.0, optional
>>> Semi-saturation constant. Value for which NK(:data:) is 1/2
>>
>> **n**
>>
>>> 0.73, optional
>>> Compression power.
>>
>> **scaling**
>>
>>> 1.0, optional
>>> Maximum value of NK-function.
>>
>> **noise**
>>
>>> 0.0, optional
>>> Cone excitation noise.
>>
>> **forward**
>>
>>> True, optional
>>> True: do NK(x)
>>> False: do NK(x)**(-1).
>
> **Returns:**
>
>> **returns**
>>
>>> float or ndarray with NK-(de)compressed input :x:

luxpy.color.cam.**deltaH**(*h1*, *C1*, *h2=None*, *C2=None*, *htype='deg'*)

> Compute a hue difference, dH = 2*C1*C2*sin(dh/2)

Args:

> **h1**
>
> > hue for sample 1 (or hue difference if h2 is None)
>
> **C1**
>
> > chroma of sample 1 (or prod C1*C2 if C2 is None)
>
> **h2**
>
> > hue angle of sample 2 (if None, then h1 contains a hue difference)
>
> **C2**
>
> > chroma of sample 2
>
> **htype**
>
> > 'deg' or 'rad', optional
> >
> > > - 'deg': hue angle between 0° and 360°
> > > - 'rad': hue angle between 0 and 2pi radians

Returns:

> **returns**
>
> > ndarray of deltaH values.

luxpy.color.cam.**hue_quadrature**(*h*, *unique_hue_data=None*, *forward=True*)

> Get hue quadrature H from hue h.
>
> **Args:**
>
> > **h**
> >
> > > float or ndarray [(N,) or (N,1)] with:
> > >
> > > > - hue angle data in degrees (!) if forward == True.
> > > > - Hue quadrature data if forward = False
> >
> > **unique_hue data**
> >
> > > None or dict, optional
> > >
> > > > - None: defaults to:
> > > >
> > > > > {'hues': 'red yellow green blue red'.split(),
> > > >
> > > > 'i': np.arange(5.0),
> > > >
> > > > 'hi':[20.14, 90.0, 164.25,237.53,380.14],
> > > >
> > > > 'ei':[0.8,0.7,1.0,1.2,0.8],
> > > >
> > > > 'Hi':[0.0,100.0,200.0,300.0,400.0]}
> > > >
> > > > - dict: user specified unique hue data
> > > >
> > > > > (same structure as above)
> >
> > **forward**
> >
> > > True, optional
> > >
> > > If true: input h is hue angle, else it is Hue quadrature
>
> **Returns:**
>
> > **H**
> >
> > > ndarray of Hue quadrature value(s) (forward == True) or of hue angle values(s)
> > > (foward == False).

luxpy.color.cam.**_update_parameter_dict**(*args*, *parameters={}*, *cieobs='2006_10'*, *match_conversionmatrix_to_cieobs=False*, *Mxyz2lms_whitepoint=None*)

Get parameter dict and update with values in args dict.

Also replace the xyz-to-lms conversion matrix with the one corresponding
to cieobs and normalize it to illuminant E.

Args:

> **args**
>
> > dictionary with updated values.
> > (get by placing 'args = locals().copy()' immediately after the start
> > of the function from which the update is called,
> > see _simple_cam() code for an example.)
>
> **parameters**
>
> > dictionary with all (adjustable) parameter values used by the model
>
> **cieobs**
>
> > String with the CIE observer CMFs (one of _CMF['types'] of the input data
> > Is used to get the Mxyz2lms matrix when match_conversionmatrix_to_cieobs ==
> > True)
>
> **match_conversionmatrix_to_cieobs**
>
> > False, optional
> > If False: keep the Mxyz2lms in the parameters dict
>
> **Mxyz2lms_whitepoint**
>
> > None, optional
> > If not None: update the Mxyz2lms key in the parameters dict
> > so that the conversion matrix is the one in _CMF[cieobs]['M'],
> > in other such that it matches the cieobs of the input data.

Returns:

> **parameters**
>
> > updated dictionary with model parameters for further use in the CAM.

Notes: For an example on the use, see code _simple_cam() (type: _simple_cam??)

luxpy.color.cam.**_setup_default_adaptation_field**(*dataw=None,* *Lw=100,*
cie_illuminant='D65', *input-*
type='xyz', *relative=True,*
cieobs='2006_10')

Setup a default illuminant adaptation field with Lw = 100 cd/m^2 for selected CIE observer.

Args:

> **dataw**
>
> > None or ndarray, optional
> > Input tristimulus values or spectral data of white point.
> > None defaults to the use of the illuminant specified in :cie_illuminant:.
>
> **cie_illuminant**
>
> > 'D65', optional
> > String corresponding to one of the illuminants (keys)
> > in luxpy._CIE_ILLUMINANT
> > If ndarray, then use this one.
> > This is ONLY USED WHEN dataw is NONE !!!
>
> **Lw**

100.0, optional

Luminance (cd/m^2) of white point.

inputtype

'xyz' or 'spd', optional

Specifies the type of input:

tristimulus values or spectral data for the forward mode.

relative

True or False, optional

True: xyz tristimulus values are relative (Yw = 100)

cieobs

_CAM_DEFAULT_CIEOBS, optional

CMF set to use to perform calculations where spectral data

is involved (inputtype == 'spd'; dataw = None)

Other options: see luxpy._CMF['types']

Returns:

dataw

Ndarray with default adaptation field data (spectral or xyz)

Notes: For an example on the use, see code _simple_cam() (type: _simple_cam??)

luxpy.color.cam.**_massage_input_and_init_output**(*data*, *dataw*, *inputtype='xyz'*, *direction='forward'*, *n_out=3*)

Redimension input data to ensure most they have the appropriate sizes for easy and efficient looping. | | 1.
Convert data and dataw to atleast_2d ndarrays | 2. Make axis 1 of dataw have 'same' dimensions as data | 3.
Make dataw have same lights source axis size as data | 4. Flip light source axis to axis=0 for efficient looping |
5. Initialize output array camout to 'same' shape as data but with camout.shape[-1] == n_out

Args:

data

ndarray with input tristimulus values

or spectral data

or input color appearance correlates

Can be of shape: (N [, xM], x 3), whereby:

N refers to samples and M refers to light sources.

Note that for spectral input shape is (N x (M+1) x wl)

dataw

None or ndarray, optional

Input tristimulus values or spectral data of white point.

None defaults to the use of CIE illuminant C.

inputtype

'xyz' or 'spd', optional

Specifies the type of input:

tristimulus values or spectral data for the forward mode.

direction

'forward' or 'inverse', optional

-'forward': xyz -> cam

-'inverse': cam -> xyz

n_out

    3, optional

    output size of last dimension of camout

    (e.g. n_out=3 for j,a,b output or n_out = 5 for J,M,h,a,b output)

Returns:

data

    ndarray with reshaped data

dataw

    ndarray with reshaped dataw

camout

    NaN filled ndarray for output of CAMv (camout.shape[-1] == Nout)

originalshape

    original shape of data

Notes: For an example on the use, see code _simple_cam() (type: _simple_cam??)

luxpy.color.cam.**_massage_output_data_to_original_shape**(*data*, *originalshape*)

    Massage output data to restore original shape of original CAM input.

    Notes: For an example on the use, see code _simple_cam() (type: _simple_cam??)

luxpy.color.cam.**_get_absolute_xyz_xyzw**(*data*, *dataw*, *i=0*, *Lw=100*, *direction='forward'*,
                                  cieobs='2006_10', *inputtype='xyz'*, *relative=True*)

    Calculate absolute xyz tristimulus values of stimulus and white point from spectral input or convert relative xyz values to absolute ones.

    Args:

data

    ndarray with input tristimulus values

    or spectral data

    or input color appearance correlates

    Can be of shape: (N [, xM], x 3), whereby:

    N refers to samples and M refers to light sources.

    Note that for spectral input shape is (N x (M+1) x wl)

dataw

    None or ndarray, optional

    Input tristimulus values or spectral data of white point.

    None defaults to the use of CIE illuminant C.

i

    0, optional

    row number in data and dataw ndarrays

    (for loops across illuminant dimension after dimension reshape

    with _massage_output_data_to_original_shape).

Lw

    100.0, optional

    Luminance (cd/m^2) of white point.

inputtype

    'xyz' or 'spd', optional

Specifies the type of input:

tristimulus values or spectral data for the forward mode.

direction

'forward' or 'inverse', optional
- 'forward': xyz -> cam
- 'inverse': cam -> xyz

relative

True or False, optional
True: xyz tristimulus values are relative (Yw = 100)

cieobs

_CAM_DEFAULT_CIEOBS, optional
CMF set to use to perform calculations where spectral data is involved (inputtype == 'spd'; dataw = None)
Other options: see luxpy._CMF['types']

Returns:

xyzti

in forward mode : ndarray with relative or absolute sample xyz for data[i]
in inverse mode: None

xyzwi

ndarray with relative or absolute white point for dataw[i]

xyzw_abs

ndarray with absolute xyz for white point for dataw[i]

Notes: For an example on the use, see code _simple_cam() (type: _simple_cam??)

luxpy.color.cam.**_simple_cam**(*data, dataw=None, Lw=100.0, relative=True, inputtype='xyz', direction='forward', cie_illuminant='D65', parameters={'Mxyz2lms': array([[3.8971e-01, 6.8898e-01, -7.8680e-02], [-2.2981e-01, 1.1834e+00, 4.6410e-02], [0.0000e+00, 0.0000e+00, 1.0000e+00]]), 'cA': 1, 'ca': array([1, -1, 0]), 'cb': array([1.6667e-01, 1.6667e-01, -3.3333e-01]), 'n': 0.3333333333333333}, cieobs='2006_10', match_to_conversionmatrix_to_cieobs=True*)

An example CAM illustration the usage of the functions in luxpy.cam.helpers

Note that this example uses NO chromatic adaptation
and SIMPLE compression, opponent and correlate processing.
THIS IS ONLY FOR ILLUSTRATION PURPOSES !!!

Args:

data

ndarray with input:
- tristimulus values

or
- spectral data

or
- input color appearance correlates

Can be of shape: (N [, xM], x 3), whereby:

N refers to samples and M refers to light sources.

Note that for spectral input shape is (N x (M+1) x wl)

dataw

None or ndarray, optional

Input tristimulus values or spectral data of white point.

None defaults to the use of :cie_illuminant:

cie_illuminant

'D65', optional

String corresponding to one of the illuminants (keys)

in luxpy._CIE_ILLUMINANT

If ndarray, then use this one.

This is ONLY USED WHEN dataw is NONE !!!

Lw

100.0, optional

Luminance (cd/m^2) of white point.

relative

True or False, optional

True: data and dataw input is relative (i.e. Yw = 100)

parameters

{'cA': 1, 'ca':np.array([1,-1,0]), 'cb':(1/3)*np.array([0.5,0.5,-1]),

'n': 1/3, 'Mxyz2lms': _CMF['1931_2']['M'].copy()}

Dict with model parameters

(For illustration purposes of match_conversionmatrix_to_cieobs,

the conversion matrix luxpy._CMF['1931_2']['M'] does NOT match

the default observer specification of the input data in :cieobs: !!!)

inputtype

'xyz' or 'spd', optional

Specifies the type of input:

tristimulus values or spectral data for the forward mode.

direction

'forward' or 'inverse', optional

-'forward': xyz -> cam

-'inverse': cam -> xyz

cieobs

'2006_10', optional

CMF set to use to perform calculations where spectral data

is involved (inputtype == 'spd'; dataw = None)

Other options: see luxpy._CMF['types']

match_conversionmatrix_to_cieobs

True, optional

> When changing to a different CIE observer, change the xyz_to_lms
> matrix to the one corresponding to that observer.
> Set to False to keep the one in the parameter dict!

Returns:

> **returns**
>
> > ndarray with:
> > - color appearance correlates (:direction: == 'forward')
> >   or
> > - XYZ tristimulus values (:direction: == 'inverse')

luxpy.color.cam.**ciecam02**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *outin='J, aM, bM'*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *forward=True*, *yellowbluepurplecorrect=False*, *mcat='cat02'*)

Run CIECAM02 color appearance model in forward or backward modes.

Args:

> **data**
>
> > ndarray with relative sample xyz values (forward mode) or J'a'b' coordinates (inverse mode)
>
> **xyzw**
>
> > ndarray with relative white point tristimulus values
>
> **Yw**
>
> > None, optional
> > Luminance factor of white point.
> > If None: xyz (in data) and xyzw are entered as relative tristimulus values
> >   (normalized to Yw = 100).
> > If not None: input tristimulus are absolute and Yw is used to
> >   rescale the absolute values to relative ones
> >   (relative to a reference perfect white diffuser
> >   with Ywr = 100).
> > Yw can be < 100 for e.g. paper as white point. If Yw is None, it
> > is assumed that the relative Y-tristimulus value in xyzw
> > represents the luminance factor Yw.
>
> **conditions**
>
> > None, optional
> > Dictionary with viewing condition parameters for:
> >   La, Yb, D and surround.
> >   surround can contain:
> >   - str (options: 'avg','dim','dark') or
> >   - dict with keys c, Nc, F.
> > None results in:
> >   {'La':100, 'Yb':20, 'D':1, 'surround':'avg'}
>
> **naka_rushton_parameters**
>
> > None, optional
> > If None: use _NAKA_RUSHTON_PARAMETERS
>
> **unique_hue_data**

None, optional

If None: use _UNIQUE_HUE_DATA

forward

True, optional

If True: run in CAM in forward mode, else: inverse mode.

outin

'J,aM,bM', optional

String with requested output (e.g. "J,aM,bM,M,h") [Forward mode]

- attributes: 'J': lightness,'Q': brightness,

'M': colorfulness,'C': chroma, 's': saturation,

'h': hue angle, 'H': hue quadrature/composition,

String with inputs in data [inverse mode].

Input must have data.shape[-1]==3 and last dim of data must have

the following structure for inverse mode:

* data[. . . ,0] = J or Q,

* data[. . . ,1:] = (aM,bM) or (aC,bC) or (aS,bS) or (M,h) or (C, h), . . .

yellowbluepurplecorrect

False, optional

If False: don't correct for yellow-blue and purple problems in ciecam02.

If 'brill-suss':

for yellow-blue problem, see:

- Brill [Color Res Appl, 2006; 31, 142-145] and

- Brill and Süsstrunk [Color Res Appl, 2008; 33, 424-426]

If 'jiang-luo':

for yellow-blue problem + purple line problem, see:

- Jiang, Jun et al. [Color Res Appl 2015: 40(5), 491-503]

mcat

'cat02', optional

Specifies CAT sensor space.

- options:

- None defaults to 'cat02'

(others e.g. 'cat02-bs', 'cat02-jiang',

all trying to correct gamut problems of original cat02 matrix)

- str: see see luxpy.cat._MCATS.keys() for options

(details on type, ?luxpy.cat)

- ndarray: matrix with sensor primaries

Returns:

camout

ndarray with color appearance correlates (forward mode)

or

XYZ tristimulus values (inverse mode)

References: 1. N. Moroney, M. D. Fairchild, R. W. G. Hunt, C. Li, M. R. Luo, and T. Newman, (2002), "The CIECAM02 color appearance model," IS&T/SID Tenth Color Imaging Conference. p. 23, 2002.

luxpy.color.cam.**xyz_to_jabM_ciecam02**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=False*, *mcat='cat02'*, ***kwargs*)

> Wrapper function for ciecam02 forward mode with J,aM,bM output.

> For help on parameter details: ?luxpy.cam.ciecam02

luxpy.color.cam.**jabM_ciecam02_to_xyz**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=False*, *mcat='cat02'*, ***kwargs*)

> Wrapper function for ciecam02 inverse mode with J,aM,bM input.

> For help on parameter details: ?luxpy.cam.ciecam02

luxpy.color.cam.**xyz_to_jabC_ciecam02**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=False*, *mcat='cat02'*, ***kwargs*)

> Wrapper function for ciecam02 forward mode with J,aC,bC output.

> For help on parameter details: ?luxpy.cam.ciecam02

luxpy.color.cam.**jabC_ciecam02_to_xyz**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=False*, *mcat='cat02'*, ***kwargs*)

> Wrapper function for ciecam02 inverse mode with J,aC,bC input.

> For help on parameter details: ?luxpy.cam.ciecam02

luxpy.color.cam.**cam02ucs**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *ucstype='ucs'*, *forward=True*, *yellowbluepurplecorrect=False*, *mcat='cat02'*)

> Run the CAM02-UCS[,-LCD,-SDC] color appearance difference model in forward or backward modes.
> **Args:**
> > **data**
> > > ndarray with sample xyz values (forward mode) or J'a'b' coordinates (inverse mode)
> > **xyzw**

ndarray with white point tristimulus values

conditions

None, optional

Dictionary with viewing conditions.

None results in:

{'La':100, 'Yb':20, 'D':1, 'surround':'avg'}

For more info see luxpy.cam.ciecam02()?

naka_rushton_parameters

None, optional

If None: use _NAKA_RUSHTON_PARAMETERS

unique_hue_data

None, optional

If None: use _UNIQUE_HUE_DATA

ucstype

'ucs', optional

String with type of color difference appearance space

options: 'ucs', 'scd', 'lcd'

forward

True, optional

If True: run in CAM in forward mode, else: inverse mode.

yellowbluepurplecorrect

False, optional

If False: don't correct for yellow-blue and purple problems in ciecam02.

If 'brill-suss':

for yellow-blue problem, see:

- Brill [Color Res Appl, 2006; 31, 142-145] and

- Brill and Süsstrunk [Color Res Appl, 2008; 33, 424-426]

If 'jiang-luo':

for yellow-blue problem + purple line problem, see:

- Jiang, Jun et al. [Color Res Appl 2015: 40(5), 491-503]

mcat

'cat02', optional

Specifies CAT sensor space.

- options:

- None defaults to 'cat02'

(others e.g. 'cat02-bs', 'cat02-jiang',

all trying to correct gamut problems of original cat02 matrix)

- str: see see luxpy.cat._MCATS.keys() for options

(details on type, ?luxpy.cat)

- ndarray: matrix with sensor primaries

Returns:

camout

ndarray with J'a'b' coordinates (forward mode)

or

XYZ tristimulus values (inverse mode)

References: 1. M.R. Luo, G. Cui, and C. Li, 'Uniform colour spaces based on CIECAM02 colour appearance model,' Color Res. Appl., vol. 31, no. 4, pp. 320–330, 2006.

luxpy.color.cam.**xyz_to_jab_cam02ucs**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=None*, *mcat='cat02'*, ***kwargs*)

Wrapper function for cam02ucs forward mode with J,aM,bM output.

For help on parameter details: ?luxpy.cam.cam02ucs

luxpy.color.cam.**jab_cam02ucs_to_xyz**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=None*, *mcat='cat02'*, ***kwargs*)

Wrapper function for cam02ucs inverse mode with J,aM,bM input.

For help on parameter details: ?luxpy.cam.cam02ucs

luxpy.color.cam.**xyz_to_jab_cam02lcd**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=None*, *mcat='cat02'*, ***kwargs*)

Wrapper function for cam02ucs forward mode with J,aMp,bMp output and ucstype = lcd.

For help on parameter details: ?luxpy.cam.cam02ucs

luxpy.color.cam.**jab_cam02lcd_to_xyz**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=None*, *mcat='cat02'*, ***kwargs*)

Wrapper function for cam02ucs inverse mode with J,aMp,bMp input and ucstype = lcd.

For help on parameter details: ?luxpy.cam.cam02ucs

luxpy.color.cam.**xyz_to_jab_cam02scd**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=None*, *mcat='cat02'*, ***kwargs*)

Wrapper function for cam02ucs forward mode with J,aMp,bMp output and ucstype = scd.

For help on parameter details: ?luxpy.cam.cam02ucs

luxpy.color.cam.**jab_cam02scd_to_xyz**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *yellowbluepurplecorrect=None*, *mcat='cat02'*, ***kwargs*)

Wrapper function for cam02ucs inverse mode with J,aMp,bMp input and ucstype = scd.

For help on parameter details: ?luxpy.cam.cam02ucs

luxpy.color.cam.**ciecam16**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *outin='J, aM, bM'*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *forward=True*, *mcat='cat16'*)

Run CIECAM16 color appearance model in forward or backward modes.

Args:

> **data**
>
>> ndarray with relative sample xyz values (forward mode) or J'a'b' coordinates (inverse mode)
>
> **xyzw**
>
>> ndarray with relative white point tristimulus values
>
> **Yw**
>
>> None, optional
>> Luminance factor of white point.
>> If None: xyz (in data) and xyzw are entered as relative tristimulus values
>>> (normalized to Yw = 100).
>> If not None: input tristimulus are absolute and Yw is used to
>>> rescale the absolute values to relative ones
>>> (relative to a reference perfect white diffuser
>>>> with Ywr = 100).
>> Yw can be < 100 for e.g. paper as white point. If Yw is None, it
>> is assumed that the relative Y-tristimulus value in xyzw
>> represents the luminance factor Yw.
>
> **conditions**
>
>> None, optional
>> Dictionary with viewing condition parameters for:
>>> La, Yb, D and surround.
>> surround can contain:
>>> - str (options: 'avg','dim','dark') or
>>> - dict with keys c, Nc, F.
>> None results in:
>>> {'La':100, 'Yb':20, 'D':1, 'surround':'avg'}
>
> **naka_rushton_parameters**

None, optional

If None: use _NAKA_RUSHTON_PARAMETERS

unique_hue_data

None, optional

If None: use _UNIQUE_HUE_DATA

forward

True, optional

If True: run in CAM in forward mode, else: inverse mode.

outin

'J,aM,bM', optional

String with requested output (e.g. "J,aM,bM,M,h") [Forward mode]

- attributes: 'J': lightness,'Q': brightness,

  'M': colorfulness,'C': chroma, 's': saturation,

  'h': hue angle, 'H': hue quadrature/composition,

String with inputs in data [inverse mode].

Input must have data.shape[-1]==3 and last dim of data must have

the following structure for inverse mode:

* data[...,0] = J or Q,

* data[...,1:] = (aM,bM) or (aC,bC) or (aS,bS) or (M,h) or (C, h), ...

mcat

'cat16', optional

Specifies CAT sensor space.

- options:

  - None defaults to 'cat16'

  - str: see see luxpy.cat._MCATS.keys() for options

    (details on type, ?luxpy.cat)

  - ndarray: matrix with sensor primaries

Returns:

camout

ndarray with color appearance correlates (forward mode)

or

XYZ tristimulus values (inverse mode)

References: 1. C. Li, Z. Li, Z. Wang, Y. Xu, M. R. Luo, G. Cui, M. Melgosa, M. H. Brill, and M. Pointer, (2017), "Comprehensive color solutions: CAM16, CAT16, and CAM16-UCS," Color Res. Appl., p. n/a–n/a.

luxpy.color.cam.**xyz_to_jabM_ciecam16**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *mcat='cat16'*, ***kwargs*)

Wrapper function for ciecam16 forward mode with J,aM,bM output.

For help on parameter details: ?luxpy.cam.ciecam16

luxpy.color.cam.**jabM_ciecam16_to_xyz**(*data,*   *xyzw=array([[1.0000e+02,*   *1.0000e+02,*
1.0000e+02]]),   *Yw=None,*   *condi-*
tions=None,   *naka_rushton_parameters=None,*
unique_hue_data=None, mcat='cat16', **kwargs)
    Wrapper function for ciecam16 inverse mode with J,aM,bM input.

    For help on parameter details: ?luxpy.cam.ciecam16

luxpy.color.cam.**xyz_to_jabC_ciecam16**(*data,*   *xyzw=array([[1.0000e+02,*   *1.0000e+02,*
1.0000e+02]]),   *Yw=None,*   *condi-*
tions=None,   *naka_rushton_parameters=None,*
unique_hue_data=None, mcat='cat16', **kwargs)
    Wrapper function for ciecam16 forward mode with J,aC,bC output.

    For help on parameter details: ?luxpy.cam.ciecam16

luxpy.color.cam.**jabC_ciecam16_to_xyz**(*data,*   *xyzw=array([[1.0000e+02,*   *1.0000e+02,*
1.0000e+02]]),   *Yw=None,*   *condi-*
tions=None,   *naka_rushton_parameters=None,*
unique_hue_data=None, mcat='cat16', **kwargs)
    Wrapper function for ciecam16 inverse mode with J,aC,bC input.

    For help on parameter details: ?luxpy.cam.ciecam16

luxpy.color.cam.**cam16ucs**(*data,*   *xyzw=array([[1.0000e+02,   1.0000e+02,   1.0000e+02]]),*
Yw=None,   *conditions=None,*   *naka_rushton_parameters=None,*
unique_hue_data=None, ucstype='ucs', forward=True, mcat='cat16')
    Run the CAM16-UCS[,-LCD,-SDC] color appearance difference model in forward or backward modes.
    Args:

        data

            ndarray with sample xyz values (forward mode) or J'a'b' coordinates (inverse mode)

        xyzw

            ndarray with white point tristimulus values

        conditions

            None, optional
            Dictionary with viewing conditions.
            None results in:
                {'La':100, 'Yb':20, 'D':1, 'surround':'avg'}
            For more info see luxpy.cam.ciecam16()?

        naka_rushton_parameters

            None, optional
             If None: use _NAKA_RUSHTON_PARAMETERS

        unique_hue_data

> None, optional
> If None: use _UNIQUE_HUE_DATA

ucstype

> 'ucs', optional
> String with type of color difference appearance space
> options: 'ucs', 'scd', 'lcd'

forward

> True, optional
> If True: run in CAM in forward mode, else: inverse mode.

mcat

> 'cat16', optional
> Specifies CAT sensor space.
> - options:
>> - None defaults to 'cat16'
>> - str: see see luxpy.cat._MCATS.keys() for options
>>> (details on type, ?luxpy.cat)
>> - ndarray: matrix with sensor primaries

Returns:

> **camout**

>> ndarray with J'a'b' coordinates (forward mode)
>>> or
>> XYZ tristimulus values (inverse mode)

References: 1. M.R. Luo, G. Cui, and C. Li, 'Uniform colour spaces based on CIECAM02 colour appearance model,' Color Res. Appl., vol. 31, no. 4, pp. 320–330, 2006.

luxpy.color.cam.**xyz_to_jab_cam16ucs**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *mcat='cat16'*, ***kwargs*)
Wrapper function for cam16ucs forward mode with J,aM,bM output.

For help on parameter details: ?luxpy.cam.cam16ucs

luxpy.color.cam.**jab_cam16ucs_to_xyz**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *mcat='cat16'*, ***kwargs*)
Wrapper function for cam16ucs inverse mode with J,aM,bM input.

For help on parameter details: ?luxpy.cam.cam16ucs

luxpy.color.cam.**xyz_to_jab_cam16lcd**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *mcat='cat16'*, ***kwargs*)
Wrapper function for cam16ucs forward mode with J,aMp,bMp output and ucstype = lcd.

For help on parameter details: ?luxpy.cam.cam16ucs

luxpy.color.cam.**jab_cam16lcd_to_xyz**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *mcat='cat16'*, ***kwargs*)
> Wrapper function for cam16ucs inverse mode with J,aMp,bMp input and ucstype = lcd.

For help on parameter details: ?luxpy.cam.cam16ucs

luxpy.color.cam.**xyz_to_jab_cam16scd**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *mcat='cat16'*, ***kwargs*)
> Wrapper function for cam16ucs forward mode with J,aMp,bMp output and ucstype = scd.

For help on parameter details: ?luxpy.cam.cam16ucs

luxpy.color.cam.**jab_cam16scd_to_xyz**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions=None*, *naka_rushton_parameters=None*, *unique_hue_data=None*, *mcat='cat16'*, ***kwargs*)
> Wrapper function for cam16ucs inverse mode with J,aMp,bMp input and ucstype = scd.

For help on parameter details: ?luxpy.cam.cam16ucs

luxpy.color.cam.**zcam**(*data*, *xyzw=None*, *outin='J, aM, bM'*, *cieobs='1931_2'*, *conditions=None*, *forward=True*, *mcat='cat02'*, *apply_cat_to_whitepoint=False*, ***kwargs*)
Run the Jz,az,bz based color appearance model in forward or backward modes.
Args:
> **data**
>> ndarray with relative sample xyz values (forward mode) or J'a'b' coordinates (inverse mode)

> **xyzw**
>> ndarray with relative white point tristimulus values
>> None defaults to D65

> **cieobs**
>> _CIEOBS, optional
>> CMF set to use when calculating :xyzw: if this is None.

> **conditions**
>> None, optional
>> Dictionary with viewing condition parameters for:
>>> La, Yb, D and surround.

surround can contain:

> - str (options: 'avg','dim','dark') or
> - dict with keys c, Nc, F.

None results in:

> {'La':100, 'Yb':20, 'D':1, 'surround':'avg'}

forward

True, optional

If True: run in CAM in forward mode, else: inverse mode.

outin

'J,aM,bM', optional

String with requested output (e.g. "J,aM,bM,M,h") [Forward mode]

- attributes: 'J': lightness,'Q': brightness,

> 'M': colorfulness,'C': chroma, 's': saturation,
>
> 'h': hue angle, 'H': hue quadrature/composition,
>
> 'Wz': whiteness, 'Kz':blackness, 'Sz': saturation, 'V': vividness

String with inputs in data [inverse mode].

Input must have data.shape[-1]==3 and last dim of data must have

the following structure for inverse mode:

> * data[...,0] = J or Q,
> * data[...,1:] = (aM,bM) or (aC,bC) or (aS,bS) or (M,h) or (C, h), ...

mcat

'cat02', optional

Specifies CAT sensor space.

- options:

> - None defaults to 'cat02'
> - str: see see luxpy.cat._MCATS.keys() for options
>
> > (details on type, ?luxpy.cat)
>
> - ndarray: matrix with sensor primaries

apply_cat_to_whitepoint

False, optional

Apply a CAT to the white point.

However, ZCAM as published doesn't do this for some reason.

Returns:

camout

ndarray with color appearance correlates (forward mode)

> or

XYZ tristimulus values (inverse mode)

References: 1. Safdar, M., Cui, G., Kim,Y. J., and Luo, M. R.(2017). Perceptually uniform color space for image signals including high dynamic range and wide gamut. Opt. Express, vol. 25, no. 13, pp. 15131–15151, Jun. 2017.

2. Safdar, M., Hardeberg, J., Cui, G., Kim, Y. J., and Luo, M. R.(2018). A Colour Appearance Model based on Jzazbz Colour Space, 26th Color and Imaging Conference (2018), Vancouver, Canada, November 12-16, 2018, pp96-101.

3. Safdar, M., Hardeberg, J.Y., Luo, M.R. (2021) ZCAM, a psychophysical model for colour appearance prediction, Optics Express. 29(4), 6036-6052, <https://doi.org/10.1364/OE.413659>'_

---

luxpy.color.cam.**xyz_to_jabz**(*xyz*, *ztype='jabz'*, *use_zcam_parameters=False*, ***kwargs*)

> Convert XYZ tristimulus values to Jz,az,bz color coordinates.
>
> **Args:**
>> **xyz**
>>
>>> ndarray with absolute tristimulus values (Y in cd/m^2!)
>>
>> **ztype**
>>
>>> 'jabz', optional
>>> String with requested return:
>>> Options: 'jabz', 'iabz'
>>
>> **use_zcam_parameters**
>>
>>> False, optional
>>> ZCAM uses a slightly different values (see notes)
>
> **Returns:**
>> **jabz**
>>
>>> ndarray with Jz (or Iz), az, bz color coordinates
>
> **Notes:**
>
>> 1. :xyz: is assumed to be under D65 viewing conditions! If necessary perform chromatic adaptation!
>>
>> 2a. Jz represents the 'lightness' relative to a D65 white with luminance = 10000 cd/m^2
>>> (note that Jz that not exactly equal 1 for this high value, but rather for 102900 cd/m2)
>> 2b. az, bz represent respectively a red-green and a yellow-blue opponent axis
>>> (but note that a D65 shows a small offset from (0,0))
>> 3. ZCAM: calculates Iz as M' - epsilon (instead L'/2 + M'/2 as in Iz,az,bz color space!).
>
> **Reference:** 1. Safdar, M., Cui, G., Kim,Y. J., and Luo, M. R. (2017). Perceptually uniform color space for image signals including high dynamic range and wide gamut. Opt. Express, vol. 25, no. 13, pp. 15131–15151, June 2017.
>
>> 2. Safdar, M., Hardeberg, J.Y., Luo, M.R. (2021) ZCAM, a psychophysical model for colour appearance prediction, Optics Express. 29(4), 6036-6052, <https://doi.org/10.1364/OE.413659>'_

luxpy.color.cam.**jabz_to_xyz**(*jabz*, *ztype='jabz'*, *use_zcam_parameters=False*, ***kwargs*)

> Convert Jz,az,bz color coordinates to XYZ tristimulus values.
>
> **Args:**
>> **jabz**
>>
>>> ndarray with Jz,az,bz color coordinates
>>
>> **ztype**
>>
>>> 'jabz', optional
>>> String with requested return:
>>> Options: 'jabz', 'iabz'
>>
>> **use_zcam_parameters**
>>
>>> False, optional
>>> ZCAM uses a slightly different values (see notes)
>
> **Returns:**
>> **xyz**
>>
>>> ndarray with tristimulus values
>
> **Note:**

1. :xyz: is assumed to be under D65 viewing conditions! If necessary perform chromatic adaptation!

2a. Jz represents the 'lightness' relative to a D65 white with luminance = 10000 cd/m^2

(note that Jz that not exactly equal 1 for this high value, but rather for 102900 cd/m2)

2b. az, bz represent respectively a red-green and a yellow-blue opponent axis

(but note that a D65 shows a small offset from (0,0))

3. ZCAM: calculates Iz as M' - epsilon (instead L'/2 + M'/2 as in Iz,az,bz color space!).

Reference: 1. Safdar, M., Cui, G., Kim,Y. J., and Luo, M. R. (2017). Perceptually uniform color space for image signals including high dynamic range and wide gamut. Opt. Express, vol. 25, no. 13, pp. 15131–15151, June, 2017.

2. Safdar, M., Hardeberg, J.Y., Luo, M.R. (2021) ZCAM, a psychophysical model for colour appearance prediction, Optics Express. 29(4), 6036-6052, <https://doi.org/10.1364/OE.413659>'_

luxpy.color.cam.**xyz_to_jabM_zcam**(*data*, *xyzw='_CIE_D65'*, *cieobs='1931_2'*, *conditions=None*, *mcat='cat02'*, *apply_cat_to_whitepoint=False*, ***kwargs*)
Wrapper function for zcam forward mode with J,aM,bM output.

For help on parameter details: ?luxpy.cam.zcam

luxpy.color.cam.**jabM_zcam_to_xyz**(*data*, *xyzw='_CIE_D65'*, *cieobs='1931_2'*, *conditions=None*, *mcat='cat02'*, *apply_cat_to_whitepoint=False*, ***kwargs*)
Wrapper function for zcam inverse mode with J,aM,bM input.

For help on parameter details: ?luxpy.cam.zcam

luxpy.color.cam.**xyz_to_jabC_zcam**(*data*, *xyzw='_CIE_D65'*, *cieobs='1931_2'*, *conditions=None*, *mcat='cat02'*, *apply_cat_to_whitepoint=False*, ***kwargs*)
Wrapper function for zcam forward mode with J,aC,bC output.

For help on parameter details: ?luxpy.cam.zcam

luxpy.color.cam.**jabC_zcam_to_xyz**(*data*, *xyzw='_CIE_D65'*, *cieobs='1931_2'*, *conditions=None*, *mcat='cat02'*, *apply_cat_to_whitepoint=False*, ***kwargs*)
Wrapper function for zcam inverse mode with J,aC,bC input.

For help on parameter details: ?luxpy.cam.zcam

luxpy.color.cam.**cam15u**(*data*, *fov=10.0*, *inputtype='xyz'*, *direction='forward'*, *outin='Q, aW, bW'*, *parameters=None*)
Convert between CIE 2006 10° XYZ tristimulus values (or spectral data) and CAM15u color appearance correlates.
Args:

> **data**
>
>> ndarray of CIE 2006 10° XYZ tristimulus values or spectral data
>>> or color appearance attributes
>
> **fov**
>
>> 10.0, optional
>> Field-of-view of stimulus (for size effect on brightness)
>
> **inputtpe**
>
>> 'xyz' or 'spd', optional
>> Specifies the type of input:
>>> tristimulus values or spectral data for the forward mode.
>
> **direction**
>
>> 'forward' or 'inverse', optional
>>> -'forward': xyz -> cam15u
>>> -'inverse': cam15u -> xyz
>
> **outin**
>
>> 'Q,aW,bW' or str, optional
>> 'Q,aW,bW' (brightness and opponent signals for amount-of-neutral)
>>> other options: 'Q,aM,bM' (colorfulness) and 'Q,aS,bS' (saturation)
>> Str specifying the type of
>>> input (:direction: == 'inverse') and
>>> output (:direction: == 'forward')
>
> **parameters**
>
>> None or dict, optional
>> Set of model parameters.
>>> - None: defaults to luxpy.cam._CAM15U_PARAMETERS
>>>> (see references below)

Returns:

> **returns**
>
>> ndarray with color appearance correlates (:direction: == 'forward')
>>> or
>> XYZ tristimulus values (:direction: == 'inverse')

References: 1. M. Withouck, K. A. G. Smet, W. R. Ryckaert, and P. Hanselaer, "Experimental driven modelling of the color appearance of unrelated self-luminous stimuli: CAM15u," Opt. Express, vol. 23, no. 9, pp. 12045–12064, 2015. 2. M. Withouck, K. A. G. Smet, and P. Hanselaer, (2015), "Brightness prediction of different sized unrelated self-luminous stimuli," Opt. Express, vol. 23, no. 10, pp. 13455–13466.

luxpy.color.cam.**xyz_to_qabW_cam15u**(*xyz*, *fov=10.0*, *parameters=None*, ***kwargs*)
> Wrapper function for cam15u forward mode with 'Q,aW,bW' output.

> For help on parameter details: ?luxpy.cam.cam15u

luxpy.color.cam.**qabW_cam15u_to_xyz**(*qab*, *fov=10.0*, *parameters=None*, ***kwargs*)
> Wrapper function for cam15u inverse mode with 'Q,aW,bW' input.

For help on parameter details: ?luxpy.cam.cam15u

luxpy.color.cam.**cam_sww16**(*data*, *dataw=None*, *Yb=20.0*, *Lw=400.0*, *Ccwb=None*, *relative=True*, *inputtype='xyz'*, *direction='forward'*, *parameters='JOSA'*, *cieobs='2006_10'*, *match_conversionmatrix_to_cieobs=True*)
A simple principled color appearance model based on a mapping of the Munsell color system.

This function implements the JOSA A (parameters = 'JOSA') published model.

Args:

> **data**
>
>> ndarray with input tristimulus values
>> or spectral data
>> or input color appearance correlates
>> Can be of shape: (N [, xM], x 3), whereby:
>> N refers to samples and M refers to light sources.
>> Note that for spectral input shape is (N x (M+1) x wl)
>
> **dataw**
>
>> None or ndarray, optional
>> Input tristimulus values or spectral data of white point.
>> None defaults to the use of CIE illuminant C.
>
> **Yb**
>
>> 20.0, optional
>> Luminance factor of background (perfect white diffuser, Yw = 100)
>
> **Lw**
>
>> 400.0, optional
>> Luminance (cd/m^2) of white point.
>
> **Ccwb**
>
>> None, optional
>> Degree of cognitive adaptation (white point balancing)
>> If None: use [..,..] from parameters dict.
>
> **relative**
>
>> True or False, optional
>> True: xyz tristimulus values are relative (Yw = 100)
>
> **parameters**
>
>> 'JOSA' or str or dict, optional
>> Dict with model parameters.
>>> - str: 'JOSA','best-fit-JOSA' or 'best-fit-all-Munsell'
>>> - dict: user defined model parameters
>>>> (dict should have same structure)
>
> **inputtype**
>
>> 'xyz' or 'spd', optional

> Specifies the type of input:
>> tristimulus values or spectral data for the forward mode.

> **direction**

>> 'forward' or 'inverse', optional
>>> -'forward': xyz -> cam_sww_2016
>>> -'inverse': cam_sww_2016 -> xyz

> **cieobs**

>> '2006_10', optional
>> CMF set to use to perform calculations where spectral data
>> is involved (inputtype == 'spd'; dataw = None)
>> Other options: see luxpy._CMF['types']

> **match_conversionmatrix_to_cieobs**

>> When changing to a different CIE observer, change the xyz_to_lms
>> matrix to the one corresponding to that observer. If False: use
>> the one set in parameters or _CAM_SWW16_PARAMETERS

Returns:
> **returns**

>> ndarray with color appearance correlates (:direction: == 'forward')
>>> or
>> XYZ tristimulus values (:direction: == 'inverse')

Notes:

> This function implements the JOSA A (parameters = 'JOSA')
> published model.
> With:

>> 1. A correction for the parameter
>>> in Eq.4 of Fig. 11: 0.952 –> -0.952

>> 2. The delta_ac and delta_bc white-balance shifts in Eq. 5e & 5f
>>> should be: -0.028 & 0.821

>> (cfr. Ccwb = 0.66 in:
>>> ab_test_out = ab_test_int - Ccwb*ab_gray_adaptation_field_int))

References: 1. Smet, K. A. G., Webster, M. A., & Whitehead, L. A. (2016). A simple principled approach for modeling and understanding uniform color metrics. Journal of the Optical Society of America A, 33(3), A319–A331.

luxpy.color.cam.**xyz_to_lab_cam_sww16**(*xyz*, *xyzw=None*, *Yb=20.0*, *Lw=400.0*, *Ccwb=None*, *relative=True*, *parameters='JOSA'*, *inputtype='xyz'*, *cieobs='2006_10'*, ***kwargs*)
> Wrapper function for cam_sww16 forward mode with 'xyz' input.

> For help on parameter details: ?luxpy.cam.cam_sww16

luxpy.color.cam.**lab_cam_sww16_to_xyz**(*lab*, *xyzw=None*, *Yb=20.0*, *Lw=400.0*, *Ccwb=None*, *relative=True*, *parameters='JOSA'*, *inputtype='xyz'*, *cieobs='2006_10'*, ***kwargs*)

Wrapper function for cam_sww16 inverse mode with 'xyz' input.

For help on parameter details: ?luxpy.cam.cam_sww16

luxpy.color.cam.**cam18sl**(*data, datab=None, Lb=[100], fov=10.0, inputtype='xyz', direction='forward', outin='Q,aS,bS', parameters=None*)

Convert between CIE 2006 10° XYZ tristimulus values (or spectral data) and CAM18sl color appearance correlates.

Args:

> **data**
>
>> ndarray of CIE 2006 10° absolute XYZ tristimulus values or spectral data
>>> or color appearance attributes of stimulus
>
> **datab**
>
>> ndarray of CIE 2006 10° absolute XYZ tristimulus values or spectral data
>>> of stimulus background
>
> **Lb**
>
>> [100], optional
>> Luminance (cd/m^2) value(s) of background(s) calculated using the CIE 2006 10°
>> CMFs
>> (only used in case datab == None and the background is assumed to be an
>> Equal-Energy-White)
>
> **fov**
>
>> 10.0, optional
>> Field-of-view of stimulus (for size effect on brightness)
>
> **inputtpe**
>
>> 'xyz' or 'spd', optional
>> Specifies the type of input:
>>> tristimulus values or spectral data for the forward mode.
>
> **direction**
>
>> 'forward' or 'inverse', optional
>>> -'forward': xyz -> cam18sl
>>> -'inverse': cam18sl -> xyz
>
> **outin**
>
>> 'Q,aS,bS' or str, optional
>> 'Q,aS,bS' (brightness and opponent signals for saturation)
>>> other options: 'Q,aM,bM' (colorfulness)
>>>> (Note that 'Q,aW,bW' would lead to a Cartesian
>>>>> a,b-coordinate system centered at (1,0))
>> Str specifying the type of
>>> input (:direction: == 'inverse') and
>>> output (:direction: == 'forward')
>
> **parameters**

None or dict, optional

Set of model parameters.
- None: defaults to luxpy.cam._CAM18SL_PARAMETERS
(see references below)

Returns:

returns

ndarray with color appearance correlates (:direction: == 'forward')
or
XYZ tristimulus values (:direction: == 'inverse')

Notes:

* Instead of using the CIE 1964 10° CMFs in some places of the model,
the CIE 2006 10° CMFs are used througout, making it more self_consistent.
This has an effect on the k scaling factors (now different those in CAM15u)
and the illuminant E normalization for use in the chromatic adaptation transform.
(see future erratum to Hermans et al., 2018)

* The paper also used an equation for the amount of white W, which is
based on a Q value not expressed in 'bright' ('cA' = 0.937 instead of 123).
This has been corrected for in the luxpy version of the model, i.e.
_CAM18SL_PARAMETERS['cW'][0] has been changed from 2.29 to 1/11672.
(see future erratum to Hermans et al., 2018)

* Default output was 'Q,aW,bW' prior to March 2020, but since this
is an a,b Cartesian system centered on (1,0), the default output
has been changed to 'Q,aS,bS'.

References: 1. Hermans, S., Smet, K. A. G., & Hanselaer, P. (2018). "Color appearance model for self-luminous stimuli." Journal of the Optical Society of America A, 35(12), 2000–2009.

luxpy.color.cam.**xyz_to_qabM_cam18sl** (*xyz, xyzb=None, Lb=[100], fov=10.0, parameters=None, **kwargs*)
Wrapper function for cam18sl forward mode with 'Q,aM,bM' output.

For help on parameter details: ?luxpy.cam.cam18sl

luxpy.color.cam.**qabM_cam18sl_to_xyz** (*qab, xyzb=None, Lb=[100], fov=10.0, parameters=None, **kwargs*)
Wrapper function for cam18sl inverse mode with 'Q,aM,bM' input.

For help on parameter details: ?luxpy.cam.cam18sl

luxpy.color.cam.**xyz_to_qabS_cam18sl** (*xyz, xyzb=None, Lb=[100], fov=10.0, parameters=None, **kwargs*)
Wrapper function for cam18sl forward mode with 'Q,aS,bS' output.

For help on parameter details: ?luxpy.cam.cam18sl

`luxpy.color.cam.`**`qabS_cam18sl_to_xyz`**(*qab, xyzb=None, Lb=[100], fov=10.0, parameters=None, **kwargs*)

    Wrapper function for cam18sl inverse mode with 'Q,aS,bS' input.

    For help on parameter details: ?luxpy.cam.cam18sl

`luxpy.color.cam.`**`camXucs`**(*data, xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=None, outin='J, aM, bM', conditions=None, forward=True, ucstype='ucs', yellowbluepurplecorrect=False, mcat=None, camtype='ciecam02'*)

    Wraps ciecam02(), ciecam16(), cam02ucs(), cam16ucs().

    Args:

            camtype

                _DEFAULT_TYPE, optional

                String specifying the cam-model.

    Notes:

        1. To call ciecam02() or ciecam16(): set ucstype to None !!!

        2. For more info on other input arguments, see doc-strings of those functions.

4.4.7 deltaE/

    py

            • __init__.py

            • colordifferences.py

            • discriminationellipses.py

            • frieleellipses.py

            • macadamellipses.py

    namespace luxpy.deltaE

Module for color difference calculations

    process_DEi() Process color difference input DEi for output (helper fnc).

    DE_camucs() Calculate color appearance difference DE using camucs type model.

    DE_2000() Calculate DE2000 color difference.

    DE_cspace() Calculate color difference DE in specific color space.

    get_macadam_ellipse() Estimate n-step MacAdam ellipse at CIE x,y coordinates

    get_brown1957_ellipse() Estimate n-step Brown (1957) ellipse at CIE x,y coordinates.

    get_gij_fmc() Get gij matrices describing the discrimination ellipses for Yxy using FMC-1 or FMC-2.

    get_fmc_discrimination_ellipse() Get n-step discrimination ellipse(s) in v-format (R,r, xc, yc, theta) for Yxy using FMC-1 or FMC-2.

`luxpy.color.deltaE.`**`deltaH`**(*h1, C1, h2=None, C2=None, htype='deg'*)

    Compute a hue difference, $dH = 2*C1*C2*\sin(dh/2)$

    Args:

h1

hue for sample 1 (or hue difference if h2 is None)

C1

chroma of sample 1 (or prod C1*C2 if C2 is None)

h2

hue angle of sample 2 (if None, then h1 contains a hue difference)

C2

chroma of sample 2

htype

'deg' or 'rad', optional
- 'deg': hue angle between 0° and 360°
- 'rad': hue angle between 0 and 2pi radians

Returns:

returns

ndarray of deltaH values.

luxpy.color.deltaE.**DE_camucs**(*xyzt*, *xyzr*, *DEtype='jab'*, *avg=None*, *avg_axis=0*, *out='DEi'*, *xyzwt=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *xyzwr=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Ywt=None*, *conditionst={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *Ywr=None*, *conditionsr={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *camtype='ciecam02'*, *ucstype='ucs'*, *mcat=None*, *outin='J, aM, bM'*, *yellowbluepurplecorrect=False*, ***kwargs*)
Calculate color appearance difference DE using camucs type model.

Args:

xyzt

ndarray with tristimulus values of test data.

xyzr

ndarray with tristimulus values of reference data.

DEtype

'jab' or str, optional
Options:
- 'jab' : calculates full color difference over all 3 dimensions.
- 'ab' : calculates chromaticity difference.
- 'j' : calculates lightness or brightness difference
    (depending on :outin:).
- 'j,ab': calculates both 'j' and 'ab' options
    and returns them as a tuple.

avg

None, optional
None: don't calculate average DE,
    otherwise use function handle in :avg:.

avg_axis

axis to calculate average over, optional

out

'DEi' or str, optional
Requested output.

camtype

luxpy.cam._CAM_DEFAULT_TYPE, optional
Str specifier for CAM type to use, options: 'ciecam02' or 'ciecam16'.

ucstype

'ucs' or 'lcd' or 'scd', optional
Str specifier for which type of color attribute compression

parameters to use:

-'ucs': uniform color space,
-'lcd': large color differences,
-'scd': small color differences

Note: For the other input arguments, see ?luxpy.cam.camucs_structure.

Returns:

returns

ndarray with DEi [, DEa] or other as specified by :out:

luxpy.color.deltaE.**DE2000**(*xyzt*, *xyzr*, *dtype='xyz'*, *DEtype='jab'*, *avg=None*, *avg_axis=0*, *out='DEi'*, *xyzwt=None*, *xyzwr=None*, *KLCH=None*)

Calculate DE2000 color difference.

Args:

xyzt

ndarray with tristimulus values of test data.

xyzr

ndarray with tristimulus values of reference data.

dtype

'xyz' or 'lab', optional
Specifies data type in :xyzt: and :xyzr:.

xyzwt

None or ndarray, optional
White point tristimulus values of test data
None defaults to the one set in lx.xyz_to_lab()

xyzwr

None or ndarray, optional
Whitepoint tristimulus values of reference data
None defaults to the one set in lx.xyz_to_lab()

DEtype

'jab' or str, optional
Options:
- 'jab' : calculates full color difference over all 3 dimensions.
- 'ab' : calculates chromaticity difference.

> - 'j' : calculates lightness or brightness difference
>   (depending on :outin:).
> - 'j,ab': calculates both 'j' and 'ab' options
>   and returns them as a tuple.

KLCH

> None, optional
> Weigths for L, C, H
> None: default to [1,1,1]

avg

> None, optional
> None: don't calculate average DE,
>   otherwise use function handle in :avg:.

avg_axis

> axis to calculate average over, optional

out

> 'DEi' or str, optional
> Requested output.

Note: For the other input arguments, see specific color space used.

Returns:

returns

> ndarray with DEi [, DEa] or other as specified by :out:

References: 1. Sharma, G., Wu, W., & Dalal, E. N. (2005). The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. Color Research & Application, 30(1), 21–30.

luxpy.color.deltaE.**DE_cspace**(*xyzt*, *xyzr*, *dtype='xyz'*, *tf='Yuv'*, *DEtype='jab'*, *avg=None*, *avg_axis=0*, *out='DEi'*, *xyzwt=None*, *xyzwr=None*, *fwtft={}*, *fwtfr={}*, *KLCH=None*, *camtype='ciecam02'*, *ucstype='ucs'*)

Calculate color difference DE in specific color space.

Args:

xyzt

> ndarray with tristimulus values of test data.

xyzr

> ndarray with tristimulus values of reference data.

dtype

> 'xyz' or 'jab', optional
> Specifies data type in :xyzt: and :xyzr:.

xyzwt

> None or ndarray, optional
> White point tristimulus values of test data
> None defaults to the one set in :fwtft:
> or else to the default of cspace.

xyzwr

> None or ndarray, optional

> > Whitepoint tristimulus values of reference data
> > > None defaults to the one set in non-empty :fwtfr:
> > > or else to default of cspace.

tf

> _CSPACE, optional
> Color space to use for color difference calculation.

fwtft

> {}, optional
> Dict with parameters for forward transform from xyz to cspace for test data.

fwtfr

> {}, optional
> Dict with parameters for forward transform
> from xyz to cspace for reference data.

KLCH

> None, optional
> Weigths for L, C, H
> None: default to [1,1,1]
> KLCH is not used when tf == 'camucs'.

DEtype

> 'jab' or str, optional
> Options:
> > - 'jab' : calculates full color difference over all 3 dimensions.
> > - 'ab' : calculates chromaticity difference.
> > - 'j' : calculates lightness or brightness difference
> > > (depending on :outin:).
> > - 'j,ab': calculates both 'j' and 'ab' options
> > > and returns them as a tuple.

avg

> None, optional
> None: don't calculate average DE,
> > otherwise use function handle in :avg:.

avg_axis

> axis to calculate average over, optional

out

> 'DEi' or str, optional
> Requested output.

camtype

> luxpy.cam._CAM_DEFAULT_TYPE, optional
> Str specifier for CAM type to use, options: 'ciecam02' or 'ciecam16'.
> Only when DEtype == 'camucs'.

ucstype

'ucs' or 'lcd' or 'scd', optional

Str specifier for which type of color attribute compression

parameters to use:

-'ucs': uniform color space,

-'lcd', large color differences,

-'scd': small color differences

Only when DEtype == 'camucs'.

Note: For the other input arguments, see specific color space used.

Returns:

returns

ndarray with DEi [, DEa] or other as specified by :out:

luxpy.color.deltaE.**get_discrimination_ellipse**(*Yxy=array([[1.0000e+02, 3.3333e-01, 3.3333e-01]]), etype='fmc2', nsteps=10, k_neighbours=3, average_cik=True, Y=None, brown1957_weighted=True*)

Get discrimination ellipse(s) in v-format (R,r, xc, yc, theta) for Yxy using an interpolation of the MacAdam ellipses or using FMC-1 or FMC-2.

Args:

Yxy

2D ndarray with [Y,]x,y coordinate centers.

If Yxy.shape[-1]==2: Y is added using the value from the Y-input argument.

etype

'fmc2', optional

Type color discrimination ellipse estimation to use.

options: 'macadam', 'fmc1', 'fmc2'

- 'macadam': interpolate covariance matrices of closest MacAdam ellipses (see: get_macadam_ellipse?).

- 'fmc1': use FMC-1 from ref 2 (see get_fmc_discrimination_ellipse?).

- 'fmc2': use FMC-1 from ref 3 (see get_fmc_discrimination_ellipse?).

- 'brown1957': interpolate covariance matrices of closest Brown1957 ellipses (see: get_brown1957_ellipse?).

nsteps

10, optional

Set multiplication factor for ellipses

(nsteps=1 corresponds to approximately 1 MacAdam step,

for FMC-2, Y also has to be 10.69, see note below).

brown1957_weighted

True, optional

If True: use weighted averages from Table III in Brown 1957 paper, else use the straight averages.

k_neighbours

3, optional

Only for option 'macadam'.

Number of nearest ellipses to use to calculate ellipse at xy

average_cik

True, optional

Only for option 'macadam'.

If True: take distance weighted average of inverse

'covariance ellipse' elements cik.

If False: average major & minor axis lengths and

ellipse orientation angles directly.

Y

None, optional

Only for option 'fmc2'(see note below).

If not None: Y = 10.69 and overrides values in Yxy.

Note:

1. FMC-2 is almost identical to FMC-1 is Y = 10.69!; see [3]

References:

1. MacAdam DL. Visual Sensitivities to Color Differences in Daylight*. J Opt Soc Am. 1942;32(5):247-274.
2. Chickering, K.D. (1967), Optimization of the MacAdam-Modified 1965 Friele Color-Difference Formula, 57(4):537-541
3. Chickering, K.D. (1971), FMC Color-Difference Formulas: Clarification Concerning Usage, 61(1):118-122
4. Brown, WRJ. (1957). Color Discrimination of Twelve Observers*. Journal of the Optical Society of America, 47(2), 137–143.

luxpy.color.deltaE.**get_macadam_ellipse**(*xy=None*, *k_neighbours=3*, *nsteps=10*, *average_cik=True*)

Estimate n-step MacAdam ellipse at CIE x,y coordinates xy by calculating average inverse covariance ellipse of the k_neighbours closest ellipses.

Args:

xy

None or ndarray, optional

If None: output Macadam ellipses, if not None: xy are the

CIE xy coordinates for which ellipses will be estimated.

k_neighbours

3, optional

Number of nearest ellipses to use to calculate ellipse at xy

nsteps

10, optional

Set number of MacAdam steps of ellipse.

average_cik

True, optional

If True: take distance weighted average of inverse

'covariance ellipse' elements cik.

If False: average major & minor axis lengths and

ellipse orientation angles directly.

Returns:

v_mac_est

estimated MacAdam ellipse(s) in v-format [Rmax,Rmin,xc,yc,theta]

References:

1. MacAdam DL. Visual Sensitivities to Color Differences in Daylight*. J Opt Soc Am. 1942;32(5):247-274.

`luxpy.color.deltaE.`**`get_brown1957_ellipse`**(*xy=None*, *weighted=True*, *k_neighbours=3*, *nsteps=10*, *average_cik=True*)

Estimate n-step Brown1957 ellipse at CIE x,y coordinates xy by calculating average inverse covariance ellipse of the k_neighbours closest ellipses.

Args:

> **xy**
>
>> None or ndarray, optional
>>
>> If None: output Brown1957 ellipses, if not None: xy are the
>>
>> CIE xy coordinates for which ellipses will be estimated.
>
> **weighted**
>
>> True, optional
>>
>> If True: use weighted averages from Table III in Brown 1957 paper, else use the straight averages.
>
> **k_neighbours**
>
>> 3, optional
>>
>> Number of nearest ellipses to use to calculate ellipse at xy
>
> **nsteps**
>
>> 10, optional
>>
>> Set number of steps of ellipse.
>
> **average_cik**
>
>> True, optional
>>
>> If True: take distance weighted average of inverse
>>
>>> 'covariance ellipse' elements cik.
>>
>> If False: average major & minor axis lengths and
>>
>>> ellipse orientation angles directly.

Returns:

> **v_brown_est**
>
>> estimated Brown1957 ellipse(s) in v-format [Rmax,Rmin,xc,yc,theta]

References:

1. Brown, W.R.J. (1957). Color Discrimination of Twelve Observers*. Journal of the Optical Society of America, 47(2), 137–143. https://doi.org/10.1364/JOSA.47.000137

`luxpy.color.deltaE.`**`get_gij_fmc`**(*Yxy*, *etype='fmc2'*, *ellipsoid=True*, *Y=None*, *cspace='Yxy'*)

Get gij matrices describing the discrimination ellipses/ellipsoids for Yxy or xyz using FMC-1 or FMC-2.

Args:

> **Yxy**
>
>> 2D ndarray with [Y,]x,y coordinate centers.
>>
>> If Yxy.shape[-1]==2: Y is added using the value from the Y-input argument.
>
> **etype**
>
>> 'fmc2', optional
>>
>> Type of FMC color discrimination equations to use (see references below).
>>
>> options: 'fmc1', fmc2'
>
> **Y**

> None, optional
>
> Only affects FMC-2 (see note below).
>
> If not None: Y = 10.69 and overrides values in Yxy.

ellipsoid

> True, optional
>
> If True: return ellipsoids, else return ellipses (only if cspace == 'Yxy')!

cspace

> 'Yxy', optional
>
> Return coefficients for Yxy-ellipses/ellipsoids ('Yxy') or XYZ ellipsoids ('xyz')

Note:

1. FMC-2 is almost identical to FMC-1 is Y = 10.69!; see [2]

References:

1. Chickering, K.D. (1967), Optimization of the MacAdam-Modified 1965 Friele Color-Difference Formula, 57(4), p.537-541
2. Chickering, K.D. (1971), FMC Color-Difference Formulas: Clarification Concerning Usage, 61(1), p.118-122

luxpy.color.deltaE.**get_fmc_discrimination_ellipse**(*Yxy=array([[1.0000e+02, 3.3333e-01, 3.3333e-01]])*, *etype='fmc2'*, *Y=None*, *nsteps=10*)

Get discrimination ellipse(s) in v-format (R,r, xc, yc, theta) for Yxy using FMC-1 or FMC-2.

Args:

Yxy

> 2D ndarray with [Y,]x,y coordinate centers.
>
> If Yxy.shape[-1]==2: Y is added using the value from the Y-input argument.

etype

> 'fmc2', optional
>
> Type of FMC color discrimination equations to use (see references below).
>
> options: 'fmc1', fmc2'

Y

> None, optional
>
> Only affects FMC-2 (see note below).
>
> If not None: Y = 10.69 and overrides values in Yxy.

nsteps

> 10, optional
>
> Set multiplication factor for ellipses
>
> (nsteps=1 corresponds to approximately 1 MacAdam step,
>
> for FMC-2, Y also has to be 10.69, see note below).

Note:

1. FMC-2 is almost identical to FMC-1 is Y = 10.69!; see [2]

References:

1. Chickering, K.D. (1967), Optimization of the MacAdam-Modified 1965 Friele Color-Difference Formula, 57(4), p.537-541
2. Chickering, K.D. (1971), FMC Color-Difference Formulas: Clarification Concerning Usage, 61(1), p.118-122

luxpy.color.deltaE.**discrimination_hotelling_t2**(*Yxy1*, *Yxy2*, *etype='fmc2'*, *ellipsoid=True*, *Y1=None*, *Y2=None*, *cspace='Yxy'*)

Check 'significance' of difference using Hotelling's T2 test on the centers Yxy1 and Yxy2 and their associate FMC-1/2 discrimination ellipses.

Args:

> **Yxy1, Yxy2**
>
> > 2D ndarrays with [Y,]x,y coordinate centers.
> >
> > If Yxy.shape[-1]==2: Y is added using the value from the Y-input argument.
>
> **etype**
>
> > 'fmc2', optional
> >
> > Type of FMC color discrimination equations to use (see references below).
> >
> > options: 'fmc1', fmc2'
>
> **Y1, Y2**
>
> > None, optional
> >
> > Only affects FMC-2 (see note below).
> >
> > If not None: Yi = 10.69 and overrides values in Yxyi.
>
> **ellipsoid**
>
> > True, optional
> >
> > If True: return ellipsoids, else return ellipses (only if cspace == 'Yxy')!
>
> **cspace**
>
> > 'Yxy', optional
> >
> > Return coefficients for Yxy-ellipses/ellipsoids ('Yxy') or XYZ ellipsoids ('xyz')

Returns:

> **p**
>
> > Chi-square based p-value
>
> **T2**
>
> > T2 test statistic (= mahalanobis distance on summed standard error cov. matrices)

Steps: 1. For each center coordinate, the standard error covariance matrix $gij^{-1} = Si/ni$ is determined using the FMC-1 or FMC-2 equations (see refs. 1 & 2). 2. Calculate sum of covariance matrices: SIG = S1/n1 + S2/n2 = $gij1^{-1}$ + $gij2^{-1}$ 3. These are then used in Hotelling's T2 test: T2 = (xy1 - xy2).T*(SIG^-1)*(xy1_xy2) 4. The T2 statistic is then tested against a Chi-square distribution with 2 or 3 degrees of freedom.

References:

> 1. Chickering, K.D. (1967), Optimization of the MacAdam-Modified 1965 Friele Color-Difference Formula, 57(4):537-541
> 2. Chickering, K.D. (1971), FMC Color-Difference Formulas: Clarification Concerning Usage, 61(1):118-122

4.4.8 whiteness/

> **py**
>
> > • __init__.py
> >
> > • smet_white_loci.py
>
> **namespace** luxpy

Module with Smet et al. (2018) neutral white loci

_UW_NEUTRALITY_PARAMETERS_SMET2014 dict with parameters of the unique white models in Smet et al. (2014)

xyz_to_neutrality_smet2018() Calculate degree of neutrality using the unique white model in Smet et al. (2014) or the normalized (max = 1) degree of chromatic adaptation model from Smet et al. (2017).

cct_to_neutral_loci_smet2018() Calculate the most neutral appearing Duv10 in and the degree of neutrality for a specified CCT using the models in Smet et al. (2018).

References

1. Smet, K. A. G. (2018). Two Neutral White Illumination Loci Based on Unique White Rating and Degree of Chromatic Adaptation. LEUKOS, 14(2), 55–67.

2. Smet, K., Deconinck, G., & Hanselaer, P., (2014), Chromaticity of unique white in object mode. Optics Express, 22(21), 25830–25841.

3. Smet, K.A.G.*, Zhai, Q., Luo, M.R., Hanselaer, P., (2017), Study of chromatic adaptation using memory color matches, Part II: colored illuminants, Opt. Express, 25(7), pp. 8350-8365.

Added August 02, 2019.

luxpy.color.whiteness.**xyz_to_neutrality_smet2018**(*xyz10*, *nlocitype='uw'*, *uw_model='Linvar'*)
Calculate degree of neutrality using the unique white model in Smet et al. (2014) or the normalized (max = 1) degree of chromatic adaptation model from Smet et al. (2017).

Args:

> **xyz10**
>
>> ndarray with CIE 1964 10° xyz tristimulus values.
>
> **nlocitype**
>
>> 'uw', optional
>> 'uw': use unique white models published in Smet et al. (2014).
>> 'ca': use degree of chromatic adaptation model from Smet et al. (2017).
>
> **uw_model**
>
>> 'Linvar', optional
>> Use Luminance invariant unique white model from Smet et al. (2014).
>> Other options: 'L200' (200 cd/m^2), 'L1000' (1000 cd/m^2) and 'L2000' (2000 cd/m^2).

Returns:

> **N**
>
>> ndarray with calculated neutrality

References: 1. Smet, K., Deconinck, G., & Hanselaer, P., (2014), Chromaticity of unique white in object mode. Optics Express, 22(21), 25830–25841.

> 2. Smet, K.A.G., Zhai, Q., Luo, M.R., Hanselaer, P., (2017), Study of chromatic adaptation using memory color matches, Part II: colored illuminants, Opt. Express, 25(7), pp. 8350-8365.

luxpy.color.whiteness.**cct_to_neutral_loci_smet2018**(*cct*, *nlocitype='uw'*, *out='duv, D'*)
Calculate the most neutral appearing Duv10 in and the degree of neutrality for a specified CCT using the models in Smet et al. (2018).

Args:

cct10

> ndarray CCT

nlocitype

> 'uw', optional
>
> 'uw': use unique white models published in Smet et al. (2014).
>
> 'ca': use degree of chromatic adaptation model from Smet et al. (2017).

out

> 'duv,D', optional
>
> Specifies requested output (other options: 'duv', 'D').

Returns:

duv

> ndarray with most neutral Duv10 value corresponding to the cct input.

D

> ndarray with the degree of neutrality at (cct, duv).

References: 1. Smet, K.A.G., (2018), Two Neutral White Illumination Loci Based on Unique White Rating and Degree of Chromatic Adaptation. LEUKOS, 14(2), 55–67.

Notes:

1. Duv is specified in the CIE 1960 u10v10 chromatity diagram as the models were developed using CIE 1964 10° tristimulus, chromaticity and CCT values.
2. The parameter +0.0172 in Eq. 4b should be -0.0172.

4.4.9 cri/

py

- __init__.py
- colorrendition.py
- /utils/
  - __init__.py
  - init_cri_defaults_database.py
  - DE_scalers.py
  - helpers.py
  - graphics.py
- /indices/
  - __init__.py
  - indices.py
  - cie_wrappers.py
  - iestm30_wrappers.py
  - cri2012.py
  - mcri.py
  - cqs.py

- – fci.py

- – thorntoncpi.py

- **/iestm30/**

  - – __init__.py

  - – metrics.py

  - – graphics.py

  - – metrics_fast.py

- **/VFPX/**

  - – __inint__.py

  - – vectorshiftmodel.py

  - – pixelshiftmodel.py

  - – VF_PX_models.py

namespace luxpy.cri

cri: sub-package suppporting color rendition calculations (colorrendition.py)

utils/init_cri_defaults_database.py

_CRI_TYPE_DEFAULT Default cri_type.

_CRI_DEFAULTS

default parameters for color fidelity and gamut area metrics (major dict has 9 keys (04-Jul-2017): sampleset [str/dict], ref_type [str], cieobs [str], avg [fcn handle], scale [dict], cspace [dict], catf [dict], rg_pars [dict], cri_specific_pars [dict])

- **Supported cri-types:**

  - – 'ciera','ciera-8','ciera-14','cierf',

  - – 'iesrf','iesrf-tm30-15','iesrf-tm30-18','iesrf-tm30-20',

  - – 'cri2012','cri2012-hl17','cri2012-hl1000','cri2012-real210',

  - – 'mcri',

  - – 'cqs-v7.5','cqs-v9.0'

  - – 'fci'

  - – 'thornton_cpi'

process_cri_type_input() load a cri_type dict but overwrites any keys that have a non-None input in calling function.

utils/DE_scalers.py

linear_scale()

Linear color rendering index scale from CIE13.3-1974/1995:

$R_{fi,a} = 100 - c_1 * DE_{i,a}$. ($c_1 = 4.6$)

log_scale()

> Log-based color rendering index scale from Davis & Ohno (2009):
> Rfi,a = 10 * ln(exp((100 - c1*DEi,a)/10) + 1)

psy_scale()

> Psychometric based color rendering index scale from Smet et al. (2013):
> Rfi,a = 100 * (2 / (exp(c1*abs(DEi,a)**(c2) + 1))) ** c3

utils/helpers.py

_get_hue_bin_data() Slice gamut spanned by the sample jabt, jabr and calculate hue-bin data.

_hue_bin_data_to_rxhj() Calculate hue bin measures: Rcshj, Rhshj, Rfhj, DEhj

_hue_bin_data_to_rfi() Get sample color differences DEi and calculate color fidelity values Rfi.

_hue_bin_data_to_rg() Calculates gamut area index, Rg.

spd_to_jab_t_r() Calculates jab color values for a sample set illuminated with test source and its reference illuminant.

spd_to_rg() Calculates the color gamut index of spectral data for a sample set illuminated with test source (data) with respect to some reference illuminant.

spd_to_DEi() Calculates color difference (~fidelity) of spectral data between sample set illuminated with test source (data) and some reference illuminant.

optimize_scale_factor() Optimize scale_factor of cri-model in cri_type such that average Rf for a set of light sources is the same as that of a target-cri (default: 'ciera')

spd_to_cri() Calculates the color rendering fidelity index (CIE Ra, CIE Rf, IES Rf, CRI2012 Rf) of spectral data. Can also output Rg, Rfhi, Rcshi, Rhshi, cct, duv, . . .

utils/graphics.py

plot_hue_bins() Makes basis plot for Color Vector Graphic (CVG).

plot_ColorVectorGraphic() Plots Color Vector Graphic (see IES TM30).

indices/indices.py

wrapper_functions_for_fidelity_type_metrics

> spd_to_ciera(): CIE 13.3 1995 version
> spd_to_ciera_133_1995(): CIE 13.3 1995 version
> spd_to_cierf(): latest version
> spd_to_cierf_224_2017(): CIE224-2017 version

> spd_to_iesrf(): latest version
> spd_to_iesrf_tm30(): latest version
> spd_to_iesrf_tm30_15(): TM30-15 version

spd_to_iesrf_tm30_18(): TM30-18 version

spd_to_iesrf_tm30_20(): TM30-20 version (= TM30-18)

spd_to_cri2012()

spd_to_cri2012_hl17()

spd_to_cri2012_hl1000()

spd_to_cri2012_real210()

wrapper_functions_for_gamut_area_metrics

spd_to_iesrg(): latest version

spd_to_iesrg_tm30(): latest version

spd_to_iesrg_tm30_15(): TM30-15 version

spd_to_iesrg_tm30_18(): TM30-18 version

spd_to_iesrg_tm30_20(): TM30-20 version (= TM30-18)

indices/mcri.py

spd_to_mcri()

Calculates the memory color rendition index, Rm:

K. A. G. Smet, W. R. Ryckaert, M. R. Pointer, G. Deconinck, and P. Hanselaer, (2012)

"A memory colour quality metric for white light sources,"

Energy Build., vol. 49, no. C, pp. 216–225.

indices/cqs.py

spd_to_cqs()

versions 7.5 and 9.0 are supported.

W. Davis and Y. Ohno,

"Color quality scale," (2010),

Opt. Eng., vol. 49, no. 3, pp. 33602–33616.

iestm30/graphics.py

spd_to_ies_tm30_metrics() Calculates IES TM30 metrics from spectral data

plot_cri_graphics() Plots graphical information on color rendition properties based on spectral data input or dict with pre-calculated measures.

_tm30_process_spd() Calculate all required parameters for plotting from spd using cri.spd_to_cri()

plot_tm30_cvg() Plot TM30 Color Vector Graphic (CVG).

plot_tm30_Rfi() Plot Sample Color Fidelity values (Rfi).

plot_tm30_Rxhj() Plot Local Chroma Shifts (Rcshj), Local Hue Shifts (Rhshj) and Local Color Fidelity values (Rfhj).

plot_tm30_Rcshj() Plot Local Chroma Shifts (Rcshj).

plot_tm30_Rhshj() Plot Local Hue Shifts (Rhshj).

plot_tm30_Rfhj() Plot Local Color Fidelity values (Rfhj).

plot_tm30_spd() Plot test SPD and reference illuminant, both normalized to the same luminous power.

plot_tm30_report() Plot a figure with an ANSI/IES-TM30 color rendition report.

plot_cri_graphics() Plots graphical information on color rendition properties based on spectral data input or dict with pre-calculated measures (cusom design). Includes Metameric uncertainty index Rt and vector-fields of color rendition shifts.

iestm30/metrics.py

spd_to_ies_tm30_metrics() Calculates IES TM30 metrics from spectral data + Metameric Uncertainty + Vector Fields

iestm30/metrics_fast.py

_cri_ref() Calculate multiple reference illuminant spectra based on ccts for color rendering index calculations.

_xyz_to_jab_cam02ucs() Calculate CAM02-UCS J'a'b' coordinates from xyz tristimulus values of sample and white point.

spd_to_tm30() Calculate tm30 measures from spd.

- Created for faster spectral optimization based on ANSI/IES-TM30 measures

VFPX

:Module_for_VectorField_and_Pixelation_CRI models.

- see ?luxpy.cri.VFPX

luxpy.color.cri.**linear_scale**(*data, scale_factor=[4.6], scale_max=100.0*)
  Linear color rendering index scale from CIE13.3-1974/1995:

  $Rf_{i,a} = 100 - c_1 * DE_{i,a}$. ($c_1 = 4.6$)

  Args:
  - **data**

      float or list[floats] or ndarray

  - **scale_factor**

      [4.6] or list[float] or ndarray, optional
      Rescales color differences before subtracting them from :scale_max:

  - **scale_max**

      100.0, optional
      Maximum value of linear scale

Returns:

> **returns**
>
>> float or list[floats] or ndarray

References: 1. CIE13.3-1995, "Method of Measuring and Specifying Colour Rendering Properties of Light Sources," CIE, Vienna, Austria, 1995.,ISBN 978 3 900734 57 2

luxpy.color.cri.**log_scale**(*data, scale_factor=[6.73], scale_max=100.0*)

Log-based color rendering index scale from Davis & Ohno (2009):

Rfi,a = 10 * ln(exp((100 - c1*DEi,a)/10) + 1).

Args:

> **data**
>
>> float or list[floats] or ndarray

> **scale_factor**
>
>> [6.73] or list[float] or ndarray, optional
>>
>> Rescales color differences before subtracting them from :scale_max:
>>
>> Note that the default value is the one from cie-224-2017.

> **scale_max**
>
>> 100.0, optional
>>
>> Maximum value of linear scale

Returns:

> **returns**
>
>> float or list[floats] or ndarray

References: 1. W. Davis and Y. Ohno, "Color quality scale," (2010), Opt. Eng., vol. 49, no. 3, pp. 33602–33616. 2. CIE224:2017. CIE 2017 Colour Fidelity Index for accurate scientific use. Vienna, Austria: CIE. (2017).

luxpy.color.cri.**psy_scale**(*data, scale_factor=[0.01818181818181818, 1.5, 2.0], scale_max=100.0*)

Psychometric based color rendering index scale from CRI2012:

Rfi,a = 100 * (2 / (exp(c1*abs(DEi,a)**(c2) + 1))) ** c3.

Args:

> **data**
>
>> float or list[floats] or ndarray

> **scale_factor**
>
>> [1/55, 3/2, 2.0] or list[float] or ndarray, optional
>>
>> Rescales color differences before subtracting them from :scale_max:
>>
>> Note that the default value is the one from (Smet et al. 2013, LRT).

> **scale_max**
>
>> 100.0, optional
>>
>> Maximum value of linear scale

Returns:

> **returns**
>
> > float or list[floats] or ndarray

References: 1. Smet, K., Schanda, J., Whitehead, L., & Luo, R. (2013). CRI2012: A proposal for updating the CIE colour rendering index. Lighting Research and Technology, 45, 689–709.

luxpy.color.cri.**_get_hue_bin_data**(*jabt*, *jabr*, *start_hue=0*, *nhbins=16*, *normalized_chroma_ref=100*)

Slice gamut spanned by the sample jabt, jabr and calculate hue-bin data.

Args:

> **jabt**
>
> > ndarray with jab sample data under test illuminant
>
> **jabr**
>
> > ndarray with jab sample data under reference illuminant
>
> **start_hue**
>
> > 0.0 or float, optional
> > Hue angle to start bin slicing
>
> **nhbins**
>
> > None or int, optional
> > > - None: defaults to using the sample hues themselves as 'bins'.
> > > > In other words, the number of bins will be equal to the number of samples.
> > > - float: number of bins to slice the sample gamut in.
>
> **normalized_chroma_ref**
>
> > 100.0 or float, optional
> > Controls the size (chroma/radius) of the normalization circle/gamut.

Returns:

> **dict**
>
> > Dictionary with keys:
> >
> > - 'jabt', 'jabr': ndarrays with jab sample data under test & ref. illuminants
> > - 'DEi': ndarray with sample jab color difference between test and ref.
> > - 'Ct', 'Cr': chroma for each sample under test and ref.
> > - 'ht', 'hr': hue angles (rad.) for each sample under test and ref.
> > - 'ht_idx', 'hr_idx': hue bin indices for each sample under test and ref.
> > - 'jabt_hj', 'jabr_hj': ndarrays with hue-bin averaged jab's under test & ref. illuminants
> > - 'DE_hj' : ndarray with average sample DE in each hue bin
> > - 'jabt_hj_closed', 'jabr_hj_closed': ndarrays with hue-bin averaged jab's under test & ref. illuminants (closed gamut: 1st == last)
> > - 'jabtn_hj', 'jabrn_hj': ndarrays with hue-bin averaged and normalized jab's under test & ref. illuminants
> > - 'jabtn_hj_closed', 'jabrn_hj_closed': ndarrays with hue-bin and normalized averaged jab's under test & ref. illuminants (closed gamut: 1st == last)
> > - 'ht_hj', 'hr_hj': hues (rad.) for each hue bin for test and ref.
> > - 'Ct_hj', 'Cr_hj': chroma for each hue bin for test and ref.

- 'Ctn_hj' : normalized chroma for each hue bin for test (ref = normalized_chroma_ref)
- 'nhbins': number of hue bins
- 'start_hue' : start hue for bin slicing
- 'normalized_chroma_ref': normalized chroma value for ref.
- 'dh': hue-angle arcs (°)
- 'hue_bin_edges': hue bin edge (rad)
- 'hbinnrs': hue bin indices for each sample under ref. (= hr_idx)

luxpy.color.cri.**spd_to_jab_t_r**(*St, cri_type='ies-tm30', out='jabt, jabr', wl=None, sampleset=None, ref_type=None, cieobs=None, cspace=None, catf=None, cri_specific_pars=None*)

Calculates jab color values for a sample set illuminated with test source SPD and its reference illuminant.

Args:

St

ndarray with spectral data
(can be multiple SPDs, first axis are the wavelengths)

out

'jabt,jabr' or str, optional
Specifies requested output (e.g.'jabt,jabr' or 'jabt,jabr,cct,duv')

wl

None, optional
Wavelengths (or [start, end, spacing]) to interpolate the spds in St to.
None: default to no interpolation

cri_type

_CRI_TYPE_DEFAULT or str or dict, optional
-'str: specifies dict with default cri model parameters
(for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
- dict: user defined model parameters
(see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
for required structure)
Note that any non-None input arguments to the function will
override default values in cri_type dict.

sampleset

None or ndarray or str, optional
Specifies set of spectral reflectance samples for cri calculations.
- None defaults to standard set for metric in cri_type.
- ndarray: user defined set of spectral reflectance functions
(.shape = (N+1, number of wavelengths);
first axis are wavelengths)

ref_type

None or str or ndarray, optional
Specifies type of reference illuminant type.
- None: defaults to metric_specific reference illuminant in
accordance with cri_type.

>> - str: 'BB' : Blackbody radiatiors,
>>> 'DL': daylightphase,

>>> 'ciera': used in CIE CRI-13.3-1995,

>>> 'cierf': used in CIE 224-2017,

>>> 'iesrf': used in TM30-15, . . .

>> - ndarray: user defined reference SPD

> **cieobs**

>> None or dict, optional

>> Specifies which CMF sets to use for the calculation of the sample

>> XYZs and the CCT (for reference illuminant calculation).

>> None defaults to the one specified in :cri_type: dict.

>>> - key: 'xyz': str specifying CMF set for calculating xyz
>>>> of samples and white

>>> - key: 'cct': str specifying CMF set for calculating cct

> **cspace**

>> None or dict, optional

>> Specifies which color space to use.

>> None defaults to the one specified in :cri_type: dict.

>>> - key: 'type': str specifying color space used to calculate
>>>> color differences in.

>>> - key: 'xyzw': None or ndarray with white point of color space
>>>> If None: use xyzw of test / reference (after chromatic
>>>>> adaptation, if specified)

>>> - other keys specify other possible parameters needed for color
>>>> space calculation,

>>>> see lx.cri._CRI_DEFAULTS['iesrf']['cspace'] for details.

> **catf**

>> None or dict, optional

>> Perform explicit CAT before converting to color space coordinates.

>>> - None: don't apply a cat (other than perhaps the one built
>>>> into the colorspace)

>>> - dict: with CAT parameters:

>>>> - key: 'D': ndarray with degree of adaptation

>>>> - key: 'mcat': ndarray with sensor matrix specification

>>>> - key: 'xyzw': None or ndarray with white point
>>>>> None: use xyzw of reference otherwise transform both
>>>>>> test and ref to xyzw

> **cri_specific_pars**

>> None or dict, optional

>> Specifies other parameters specific to type of cri

>> (e.g. maxC for CQS calculations)

>>> - None: default to the one specified in :cri_type: dict.

>>> - dict: user specified parameters.

>>>> For its use, see for example:

luxpy.cri._CRI_DEFAULTS['mcri']['cri_specific_pars']

Returns:

> **returns**
>
>> (ndarray, ndarray)
>> with jabt and jabr data for :out: 'jabt,jabr'
>>
>> Other output is also possible by changing the :out: str value.

luxpy.color.cri.**spd_to_rg**(*St*, *cri_type='ies-tm30'*, *out='Rg'*, *wl=None*, *sampleset=None*, *ref_type=None*, *cieobs=None*, *avg=None*, *cspace=None*, *catf=None*, *cri_specific_pars=None*, *rg_pars=None*, *fit_gamut_ellipse=False*)

Calculates the color gamut index, Rg, of spectral data.

Args:

> **St**
>
>> ndarray with spectral data
>> (can be multiple SPDs, first axis are the wavelengths)
>
> **out**
>
>> 'Rg' or str, optional
>> Specifies requested output (e.g. 'Rg,cct,duv')
>
> **wl**
>
>> None, optional
>> Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
>> None: default to no interpolation
>
> **cri_type**
>
>> _CRI_TYPE_DEFAULT or str or dict, optional
>>> -'str: specifies dict with default cri model parameters
>>>> (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
>>> - dict: user defined model parameters
>>>> (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
>>>> for required structure)
>> Note that any non-None input arguments to the function will
>> override default values in cri_type dict.
>
> **sampleset**
>
>> None or ndarray or str, optional
>> Specifies set of spectral reflectance samples for cri calculations.
>>> - None defaults to standard set for metric in cri_type.
>>> - ndarray: user defined set of spectral reflectance functions
>>>> (.shape = (N+1, number of wavelengths);
>>>>> first axis are wavelengths)
>
> **ref_type**
>
>> None or str or ndarray, optional
>> Specifies type of reference illuminant type.
>>> - None: defaults to metric_specific reference illuminant in
>>>> accordance with cri_type.
>>> - str: 'BB' : Blackbody radiatiors,

> 'DL': daylightphase,
>
> 'ciera': used in CIE CRI-13.3-1995,
>
> 'cierf': used in CIE 224-2017,
>
> 'iesrf': used in TM30-15, . . .
>
> - ndarray: user defined reference SPD

cieobs

> None or dict, optional
>
> Specifies which CMF sets to use for the calculation of the sample
> XYZs and the CCT (for reference illuminant calculation).
> None defaults to the one specified in :cri_type: dict.
>
> > - key: 'xyz': str specifying CMF set for calculating xyz
> > > of samples and white
> > - key: 'cct': str specifying CMF set for calculating cct

cspace

> None or dict, optional
>
> Specifies which color space to use.
> None defaults to the one specified in :cri_type: dict.
>
> > - key: 'type': str specifying color space used to calculate
> > > color differences in.
> > - key: 'xyzw': None or ndarray with white point of color space
> > > If None: use xyzw of test / reference (after chromatic
> > > > adaptation, if specified)
> > - other keys specify other possible parameters needed for color
> > > space calculation,
> > > see lx.cri._CRI_DEFAULTS['iesrf']['cspace'] for details.

catf

> None or dict, optional
>
> Perform explicit CAT before converting to color space coordinates.
>
> > - None: don't apply a cat (other than perhaps the one built
> > > into the colorspace)
> > - dict: with CAT parameters:
> > > - key: 'D': ndarray with degree of adaptation
> > > - key: 'mcat': ndarray with sensor matrix specification
> > > - key: 'xyzw': None or ndarray with white point
> > > > None: use xyzw of reference otherwise transform both
> > > > > test and ref to xyzw

cri_specific_pars

> None or dict, optional
>
> Specifies other parameters specific to type of cri
> (e.g. maxC for CQS calculations)
>
> > - None: default to the one specified in :cri_type: dict.
> > - dict: user specified parameters.
> > > For its use, see for example:
> > > > luxpy.cri._CRI_DEFAULTS['mcri']['cri_specific_pars']

rg_pars

> None or dict, optional
> Dict containing specifying parameters for slicing the gamut.
> Dict structure:
>> {'nhbins' : None, 'start_hue' : 0,
>>> 'normalize_gamut' : False, 'normalized_chroma_ref': 100.0}
>
> - key: 'nhbins': int, number of hue bins to slice gamut
>   (None use the one specified in :cri_type: dict).
> - key: 'start_hue': float (°), hue at which to start slicing
> - key: 'normalize_gamut': True or False:
>   normalize gamut or not before calculating a gamut
>   area index Rg.
> - key: 'normalized_chroma_ref': 100.0 or float, optional
>   Controls the size (chroma/radius)
>   of the normalization circle/gamut.
> - key 'use_bin_avg_DEi': True or False
>   Note that following IES-TM30 DEhj from gamut_slicer()
>   is obtained by averaging the DEi per hue bin (True),
>   and NOT by averaging the jabt and jabr per hue bin
>   and then calculating the DEhj (False).

avg

> None or fcn handle, optional
> Averaging function (handle) for color differences, DEi
> (e.g. numpy.mean, .math.rms, .math.geomean)
> None use the one specified in :cri_type: dict.

scale

> None or dict, optional
> Specifies scaling of color differences to obtain CRI.
> - None use the one specified in :cri_type: dict.
> - dict: user specified dict with scaling parameters.
>> - key: 'fcn': function handle to type of cri scale,
>>> e.g.
>>> * linear()_scale –> (100 - scale_factor*DEi),
>>> * log_scale –> (cfr. Ohno's CQS),
>>> * psy_scale (Smet et al.'s cri2012,See: LRT 2013)
>> - key: 'cfactor': factors used in scaling function,
>>> If None:
>>>> Scaling factor value(s) will be optimized to
>>>> minimize the rms between the Rf's of the
>>>> requested metric and the target metric specified
>>>> in:
>>>>
>>>>> - key: 'opt_cri_type': str
>>>>>> * str: one of the preset _CRI_DEFAULTS
>>>>>> * dict: user speciied

(dict must contain all keys as normal)

Note that if key not in :scale: dict,

then 'opt_cri_type' is added with default

setting = 'ciera'.

- key: 'opt_spd_set': ndarray with set of light

source spds used to optimize cfactor.

Note that if key not in :scale: dict,

then default = 'F1-F12'.

fit_gamut_ellipse

fit ellipse to normalized color gamut

(extract from function using out; also stored in hue_bin_data['gamut_ellipse_fit'])

Returns:

returns

float or ndarray with Rg for :out: 'Rg'

Other output is also possible by changing the :out: str value.

E.g. out == 'Rg,data' would output an ndarray with Rg values

and a dictionary :data: with keys:

'St', 'Sr', 'cct', 'duv', 'hue_bin_data'

'xyzti', xyzti, 'xyztw', 'xyzri', 'xyzrw'

References: 1. IES TM30, Method for Evaluating Light Source Color Rendition. New York, NY: The Illuminating Engineering Society of North America.

2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.

luxpy.color.cri.**spd_to_DEi**(*St*, *cri_type='ies-tm30'*, *out='DEi'*, *wl=None*, *sampleset=None*, *ref_type=None*, *cieobs=None*, *avg=None*, *cspace=None*, *catf=None*, *cri_specific_pars=None*)

Calculates color differences (~fidelity), DEi, of spectral data.

Args:

St

ndarray with spectral data

(can be multiple SPDs, first axis are the wavelengths)

out

'DEi' or str, optional

Specifies requested output (e.g. 'DEi,DEa,cct,duv')

wl

None, optional

Wavelengths (or [start, end, spacing]) to interpolate the spds in St to.

None: default to no interpolation

cri_type

_CRI_TYPE_DEFAULT or str or dict, optional

- 'str: specifies dict with default cri model parameters

(for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])

- dict: user defined model parameters

(see e.g. luxpy.cri._CRI_DEFAULTS['cierf']

for required structure)

Note that any non-None input arguments to the function will override default values in cri_type dict.

sampleset

None or ndarray or str, optional

Specifies set of spectral reflectance samples for cri calculations.
- None defaults to standard set for metric in cri_type.
- ndarray: user defined set of spectral reflectance functions
    (.shape = (N+1, number of wavelengths);
        first axis are wavelengths)

ref_type

None or str or ndarray, optional

Specifies type of reference illuminant type.
- None: defaults to metric_specific reference illuminant in
    accordance with cri_type.
- str: 'BB' : Blackbody radiatiors,
    'DL': daylightphase,
    'ciera': used in CIE CRI-13.3-1995,
    'cierf': used in CIE 224-2017,
    'iesrf': used in TM30-15, . . .
- ndarray: user defined reference SPD

cieobs

None or dict, optional

Specifies which CMF sets to use for the calculation of the sample

XYZs and the CCT (for reference illuminant calculation).

None defaults to the one specified in :cri_type: dict.
- key: 'xyz': str specifying CMF set for calculating xyz
    of samples and white
- key: 'cct': str specifying CMF set for calculating cct

cspace

None or dict, optional

Specifies which color space to use.

None defaults to the one specified in :cri_type: dict.
- key: 'type': str specifying color space used to calculate
    color differences in.
- key: 'xyzw': None or ndarray with white point of color space
    If None: use xyzw of test / reference (after chromatic
        adaptation, if specified)
- other keys specify other possible parameters needed for color
    space calculation,
    see lx.cri._CRI_DEFAULTS['iesrf']['cspace'] for details.

catf

None or dict, optional

Perform explicit CAT before converting to color space coordinates.
- None: don't apply a cat (other than perhaps the one built
    into the colorspace)
- dict: with CAT parameters:

> - key: 'D': ndarray with degree of adaptation
> - key: 'mcat': ndarray with sensor matrix specification
> - key: 'xyzw': None or ndarray with white point
>> None: use xyzw of reference otherwise transform both
>>> test and ref to xyzw

cri_specific_pars

> None or dict, optional
>
> Specifies other parameters specific to type of cri
>
> (e.g. maxC for CQS calculations)
>> - None: default to the one specified in :cri_type: dict.
>> - dict: user specified parameters.
>>> For its use, see for example:
>>>> luxpy.cri._CRI_DEFAULTS['mcri']['cri_specific_pars']

Returns:

returns

> float or ndarray with DEi for :out: 'DEi'

> Other output is also possible by changing the :out: str value.

luxpy.color.cri.**optimize_scale_factor**(*cri_type*,   *opt_scale_factor*,   *scale_fcn*,   *avg*,   *rf_from_avg_rounded_rfi*)

Optimize scale_factor of cri-model in cri_type such that average Rf for a set of light sources is the same as that of a target-cri (default: 'ciera').

Args:

cri_type

> str or dict
>> -'str: specifies dict with default cri model parameters
>>> (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
>> - dict: user defined model parameters
>>> (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
>>> for required structure)

opt_scale

> True or False
>
> True: optimize scaling-factor, else do nothing and use value of
>
> scaling-factor in :scale: dict.

scale_fcn

> function handle to type of cri scale,
>
> e.g.
>> * linear()_scale –> (100 - scale_factor*DEi),
>> * log_scale –> (cfr. Ohno's CQS),
>> * psy_scale (Smet et al.'s cri2012,See: LRT 2013)

avg

> None or fcn handle
>
> Averaging function (handle) for color differences, DEi
>
> (e.g. numpy.mean, .math.rms, .math.geomean)
>
> None use the one specified in :cri_type: dict.

Returns:

scaling_factor

---

ndarray

luxpy.color.cri.**spd_to_cri**(*St*, *cri_type='ies-tm30'*, *out='Rf'*, *wl=None*, *sample-set=None*, *ref_type=None*, *cieobs=None*, *avg=None*, *rf_from_avg_rounded_rfi=None*, *scale=None*, *opt_scale_factor=False*, *cspace=None*, *catf=None*, *cri_specific_pars=None*, *rg_pars=None*, *fit_gamut_ellipse=False*)

Calculates the color rendering fidelity index, Rf, of spectral data.

Args:

    St

        ndarray with spectral data

        (can be multiple SPDs, first axis are the wavelengths)

    out

        'Rf' or str, optional

        Specifies requested output (e.g. 'Rf,cct,duv')

    wl

        None, optional

        Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.

        None: default to no interpolation

    cri_type

        _CRI_TYPE_DEFAULT or str or dict, optional

            -'str: specifies dict with default cri model parameters

                (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])

            - dict: user defined model parameters

                (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']

                for required structure)

        Note that any non-None input arguments to the function will

        override default values in cri_type dict.

    sampleset

        None or ndarray or str, optional

        Specifies set of spectral reflectance samples for cri calculations.

            - None defaults to standard set for metric in cri_type.

            - ndarray: user defined set of spectral reflectance functions

                (.shape = (N+1, number of wavelengths);

                    first axis are wavelengths)

    ref_type

        None or str or ndarray, optional

        Specifies type of reference illuminant type.

            - None: defaults to metric_specific reference illuminant in

                accordance with cri_type.

            - str: 'BB' : Blackbody radiatiors,

                'DL': daylightphase,

                'ciera': used in CIE CRI-13.3-1995,

                'cierf': used in CIE 224-2017,

                'iesrf': used in TM30-15, . . .

            - ndarray: user defined reference SPD

    cieobs

> None or dict, optional
> Specifies which CMF sets to use for the calculation of the sample
> XYZs and the CCT (for reference illuminant calculation).
> None defaults to the one specified in :cri_type: dict.
>> - key: 'xyz': str specifying CMF set for calculating xyz
>>   of samples and white
>> - key: 'cct': str specifying CMF set for calculating cct

cspace

> None or dict, optional
> Specifies which color space to use.
> None defaults to the one specified in :cri_type: dict.
>> - key: 'type': str specifying color space used to calculate
>>   color differences in.
>> - key: 'xyzw': None or ndarray with white point of color space
>>   If None: use xyzw of test / reference (after chromatic
>>   adaptation, if specified)
>> - other keys specify other possible parameters needed for color
>>   space calculation,
>>   see lx.cri._CRI_DEFAULTS['iesrf']['cspace'] for details.

catf

> None or dict, optional
> Perform explicit CAT before converting to color space coordinates.
>> - None: don't apply a cat (other than perhaps the one built
>>   into the colorspace)
>> - dict: with CAT parameters:
>>   - key: 'D': ndarray with degree of adaptation
>>   - key: 'mcat': ndarray with sensor matrix specification
>>   - key: 'xyzw': None or ndarray with white point
>>   None: use xyzw of reference otherwise transform both
>>   test and ref to xyzw

cri_specific_pars

> None or dict, optional
> Specifies other parameters specific to type of cri
> (e.g. maxC for CQS calculations)
>> - None: default to the one specified in :cri_type: dict.
>> - dict: user specified parameters.
>>   For its use, see for example:
>>   luxpy.cri._CRI_DEFAULTS['mcri']['cri_specific_pars']

rg_pars

> None or dict, optional
> Dict containing specifying parameters for slicing the gamut
> and calculating hue bin specific indices.
> Dict structure:
>> {'nhbins' : None, 'start_hue' : 0,
>>   'normalize_gamut' : False, 'normalized_chroma_ref': 100.0}
>> - key: 'nhbins': int, number of hue bins to slice gamut

(None use the one specified in :cri_type: dict).

- key: 'start_hue': float (°), hue at which to start slicing

- key: 'normalize_gamut': True or False:

> normalize gamut or not before calculating a gamut
>
> area index Rg.

- key: 'normalized_chroma_ref': 100.0 or float, optional

> Controls the size (chroma/radius)
>
> of the normalization circle/gamut.

- key 'use_bin_avg_DEi': True or False

> Note that following IES-TM30 DEhj from gamut_slicer()
>
> is obtained by averaging the DEi per hue bin (True),
>
> and NOT by averaging the jabt and jabr per hue bin
>
> and then calculating the DEhj (False).

avg

> None or fcn handle, optional
>
> Averaging function (handle) for color differences, DEi
>
> (e.g. numpy.mean, .math.rms, .math.geomean)
>
> None use the one specified in :cri_type: dict.

rf_from_avg_rounded_rfi

> None, optional
>
> If None: use as specified in the :cri_type: dict
>
> If False: calculate Rf directly from DEa.
>
> If True: round Rfi to integer numbers and average them to Rf
>
> > (method used in CIE-13.3-1995 Ra calculation)

scale

> None or dict, optional
>
> Specifies scaling of color differences to obtain CRI.
>
> > - None use the one specified in :cri_type: dict.
> >
> > - dict: user specified dict with scaling parameters.
> >
> > > - key: 'fcn': function handle to type of cri scale,
> > >
> > > > e.g.
> > > >
> > > > * linear_scale –> (100 - scale_factor*DEi),
> > > >
> > > > * log_scale –> (cfr. Ohno's CQS),
> > > >
> > > > * psy_scale (Smet et al.'s cri2012,See: LRT 2013)
> > >
> > > - key: 'cfactor': factors used in scaling function,
> > >
> > > > If None:
> > > >
> > > > > Scaling factor value(s) will be optimized to
> > > > > minimize the rms between the Rf's of the
> > > > > requested metric and the target metric specified
> > > > > in:
> > > > >
> > > > > > - key: 'opt_cri_type': str
> > > > > >
> > > > > > > * str: one of the preset _CRI_DEFAULTS
> > > > > > >
> > > > > > > * dict: user speciied
> > > > > > >
> > > > > > > (dict must contain all keys as normal)
> > > > > > >
> > > > > > > Note that if key not in :scale: dict,

> then 'opt_cri_type' is added with default
> setting = 'ciera'.
> - key: 'opt_spd_set': ndarray with set of light
> source spds used to optimize cfactor.
> Note that if key not in :scale: dict,
> then default = 'F1-F12'.

opt_scale_factor

> True or False, optional
> True: optimize scaling-factor, else do nothing and use value of
> scaling-factor in :scale: dict.

fit_gamut_ellipse

> fit ellipse to normalized color gamut
> (extract from function using out; also stored in hue_bin_data['gamut_ellipse_fit'])

Returns:

returns

> float or ndarray with Rf for :out: 'Rf'
> Other output is also possible by changing the :out: str value.
> E.g. out == 'Rg,data' would output an ndarray with Rf values
>
> and a dictionary :data: with keys:
>
> - 'St, Sr' : ndarray of test SPDs and corresponding ref. illuminants.
> - 'xyz_cct': xyz of white point calculate with cieobs defined for cct calculations in cri_type['cieobs']
> - 'cct, duv': CCT and Duv obtained with cieobs in cri_type['cieobs']['cct']
> - 'xyzti, xyzri': ndarray tristimulus values of test and ref. samples (obtained with with cieobs in cri_type['cieobs']['xyz'])
> - 'xyztw, xyzrw': ndarray tristimulus values of test and ref. white points (obtained with with cieobs in cri_type['cieobs']['xyz'])
> - 'DEi, DEa': ndarray with individual sample color differences DEi and average DEa between test and ref.
> - 'Rf' : ndarray with general color fidelity index values
> - 'Rg' : ndarray with color gamut area index values
> - 'Rfi' : ndarray with specific (sample) color fidelity indices
> - 'Rfhj' : ndarray with local (hue binned) fidelity indices
> - 'DEhj' : ndarray with local (hue binned) color differences
> - 'Rcshj': ndarray with local chroma shifts indices
> - 'Rhshj': ndarray with local hue shifts indices
> - 'hue_bin_data': dict with output from _get_hue_bin_data() [see its help for more info]
> - 'cri_type': same as input (for reference purposes)

References: 1. IES TM30, Method for Evaluating Light Source Color Rendition. New York, NY: The Illuminating Engineering Society of North America.

2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.

3. CIE224:2017. CIE 2017 Colour Fidelity Index for accurate scientific use. Vienna, Austria: CIE.

(2017).

4. Smet, K., Schanda, J., Whitehead, L., & Luo, R. (2013). CRI2012: A proposal for updating the CIE colour rendering index. Lighting Research and Technology, 45, 689–709.

5. CIE13.3-1995. Method of Measuring and Specifying Colour Rendering Properties of Light Sources (Vol. CIE13.3-19). Vienna, Austria: CIE. (1995).

luxpy.color.cri.**_hue_bin_data_to_rxhj**(*hue_bin_data*, *cri_type='ies-tm30'*, *scale_factor=None*, *scale_fcn=None*, *use_bin_avg_DEi=True*)

Calculate hue bin measures: Rcshj, Rhshj, Rfhj, DEhj.

Rcshj: local chroma shift
Rhshj: local hue shift
Rfhj: local (hue bin) color fidelity
DEhj: local (hue bin) color differences

(See IES TM30)

Args:

>   **hue_bin_data**
>
>>   Dict with hue bin data obtained with _get_hue_bin_data().
>
>   **use_bin_avg_DEi**
>
>>   True, optional
>>   Note that following IES-TM30 DEhj from gamut_slicer() is obtained by
>>   averaging the DEi per hue bin (True), and NOT by averaging the
>>   jabt and jabr per hue bin and then calculating the DEhj (False).
>>   If None: use value in rg_pars dict in cri_type dict!
>
>   **scale_fcn**
>
>>   function handle to type of cri scale,
>>   e.g.
>>> * linear()_scale –> (100 - scale_factor*DEi),
>>> * log_scale –> (cfr. Ohno's CQS),
>>> * psy_scale (Smet et al.'s cri2012,See: LRT 2013)
>
>   **scale_factor**
>
>>   factors used in scaling function

Returns:

>   **returns**
>
>>   ndarrays of Rcshj, Rhshj, Rfhj, DEhj

References: 1. IES TM30, Method for Evaluating Light Source Color Rendition. New York, NY: The Illuminating Engineering Society of North America.

luxpy.color.cri.**_hue_bin_data_to_rfi**(*hue_bin_data=None*, *cri_type='ies-tm30'*, *scale_factor=None*, *scale_fcn=None*)

Get sample color differences DEi and calculate color fidelity values Rfi.

Rfi: Sample color fidelity
DEi: Sample color differences

(See IES TM30)

Args:

>**hue_bin_data**

>>Dict with hue bin data obtained with _get_hue_bin_data().

>**scale_fcn**

>>function handle to type of cri scale,

>>e.g.

>>>* linear()_scale –> (100 - scale_factor*DEi),
>>>* log_scale –> (cfr. Ohno's CQS),
>>>* psy_scale (Smet et al.'s cri2012,See: LRT 2013)

>**scale_factor**

>>factors used in scaling function

Returns:

>**returns**

>>ndarrays of Rfi, DEi

References: 1. IES TM30, Method for Evaluating Light Source Color Rendition. New York, NY: The Illuminating Engineering Society of North America.

luxpy.color.cri.**_hue_bin_data_to_rg**(*hue_bin_data*, *max_scale=100*, *normalize_gamut=False*)

Calculates gamut area index, Rg.

Args:

>**hue_bin_data**

>>Dict with hue bin data obtained with _get_hue_bin_data().

>**max_scale**

>>100.0, optional
>>Value of Rg when Rf = max_scale (i.e. DEavg = 0)

>**normalize_gamut**

>>False, optional
>>True normalizes the gamut of test to that of ref.
>>(perfect agreement results in circle).

>**out**

>>'Rg', optional
>>Specifies which variables to output as ndarray

Returns:

>**Rg**

>>float or ndarray with gamut area indices Rg.

luxpy.color.cri.**spd_to_ciera**(*SPD*, *out='Rf'*, *wl=None*)

Wrapper function the 'ciera' color rendition (fidelity) metric (CIE 13.3-1995).

Args:

>**SPD**

>>ndarray with spectral data
>>(can be multiple SPDs, first axis are the wavelengths)

>**wl**

>>None, optional

Wavelengths (or [start, end, spacing]) to interpolate :SPD: to.

None: default to no interpolation

out

'Rf' or str, optional

Specifies requested output (e.g. 'Rf,Rfi,cct,duv')

Returns:

returns

float or ndarray with CIE13.3 Ra for :out: 'Rf'

Other output is also possible by changing the :out: str value.

References: 1. CIE13.3-1995. Method of Measuring and Specifying Colour Rendering Properties of Light Sources (Vol. CIE13.3-19). Vienna, Austria: CIE. (1995).

luxpy.color.cri.**spd_to_cierf**(*SPD*, *out='Rf'*, *wl=None*)

Wrapper function the 'cierf' color rendition (fidelity) metric (CIE224-2017).

Args:

SPD

ndarray with spectral data (can be multiple SPDs,

first axis are the wavelengths)

wl

None, optional

Wavelengths (or [start, end, spacing]) to interpolate :SPD: to.

None: default to no interpolation

out

'Rf' or str, optional

Specifies requested output (e.g. 'Rf,Rfi,cct,duv')

Returns:

returns

float or ndarray with CIE224-2017 Rf for :out: 'Rf'

Other output is also possible by changing the :out: str value.

References: 1. CIE224:2017. CIE 2017 Colour Fidelity Index for accurate scientific use. Vienna, Austria: CIE. (2017).

luxpy.color.cri.**spd_to_ciera_133_1995**(*SPD*, *out='Rf'*, *wl=None*)

Wrapper function the 'ciera' color rendition (fidelity) metric (CIE 13.3-1995).

Args:

SPD

ndarray with spectral data

(can be multiple SPDs, first axis are the wavelengths)

wl

None, optional

Wavelengths (or [start, end, spacing]) to interpolate :SPD: to.

None: default to no interpolation

out

'Rf' or str, optional

Specifies requested output (e.g. 'Rf,Rfi,cct,duv')

Returns:

returns

float or ndarray with CIE13.3 Ra for :out: 'Rf'

Other output is also possible by changing the :out: str value.

References: 1. CIE13.3-1995. Method of Measuring and Specifying Colour Rendering Properties of Light Sources (Vol. CIE13.3-19). Vienna, Austria: CIE. (1995).

luxpy.color.cri.**spd_to_cierf_224_2017**(*SPD*, *out='Rf'*, *wl=None*)

Wrapper function the 'cierf' color rendition (fidelity) metric (CIE224-2017).

Args:

    SPD

        ndarray with spectral data (can be multiple SPDs,

        first axis are the wavelengths)

    wl

        None, optional

        Wavelengths (or [start, end, spacing]) to interpolate :SPD: to.

        None: default to no interpolation

    out

        'Rf' or str, optional

        Specifies requested output (e.g. 'Rf,Rfi,cct,duv')

Returns:

    returns

        float or ndarray with CIE224-2017 Rf for :out: 'Rf'

        Other output is also possible by changing the :out: str value.

References: 1. CIE224:2017. CIE 2017 Colour Fidelity Index for accurate scientific use. Vienna, Austria: CIE. (2017).

luxpy.color.cri.**spd_to_iesrf**(*SPD*, *out='Rf'*, *wl=None*, *cri_type='iesrf-tm30-20'*)

Wrapper function for the 'iesrf' color fidelity index (IES TM30-20 = TM30-18).

Args:

    SPD

        ndarray with spectral data (can be multiple SPDs,

        first axis are the wavelengths)

    wl

        None, optional

        Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.

        None: default to no interpolation

    out

        'Rf' or str, optional

        Specifies requested output (e.g. 'Rf,Rfi,cct,duv')

Returns:

    returns

        float or ndarray with IES TM30-20 Rf for :out: 'Rf'

        Other output is also possible by changing the :out: str value.

References: 1. IES TM30 (99, 4880 spectrally uniform samples)

2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.

3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

luxpy.color.cri.**spd_to_iesrg**(*SPD*, *out='Rg'*, *wl=None*, *cri_type='iesrf-tm30-20'*)
> Wrapper function for the 'spd_to_rg' color gamut area index (IES TM30-18 = TM30-20).
> **Args:**
> > **SPD**
> > > ndarray with spectral data (can be multiple SPDs,
> > > first axis are the wavelengths)
> > **wl**
> > > None, optional
> > > Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
> > > None: default to no interpolation
> > **out**
> > > 'Rg' or str, optional
> > > Specifies requested output (e.g. 'Rg,Rf,Rfi,cct,duv')
> **Returns:**
> > **returns**
> > > float or ndarray with IES TM30-20 Rg for :out: 'Rg'
> > > Other output is also possible by changing the :out: str value.
> **References:** 1. IES TM30 (99, 4880 spectrally uniform samples)
>
> > 2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.
> >
> > 3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

luxpy.color.cri.**spd_to_iesrf_tm30**(*SPD*, *out='Rf'*, *wl=None*, *cri_type='iesrf-tm30-20'*)
> Wrapper function for the 'iesrf' color fidelity index (IES TM30-20 = TM30-18).
> **Args:**
> > **SPD**
> > > ndarray with spectral data (can be multiple SPDs,
> > > first axis are the wavelengths)
> > **wl**
> > > None, optional
> > > Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
> > > None: default to no interpolation
> > **out**
> > > 'Rf' or str, optional
> > > Specifies requested output (e.g. 'Rf,Rfi,cct,duv')
> **Returns:**
> > **returns**
> > > float or ndarray with IES TM30-20 Rf for :out: 'Rf'
> > > Other output is also possible by changing the :out: str value.
> **References:** 1. IES TM30 (99, 4880 spectrally uniform samples)
>
> > 2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.
> >
> > 3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

`luxpy.color.cri.`**`spd_to_iesrg_tm30`**(*SPD*, *out='Rg'*, *wl=None*, *cri_type='iesrf-tm30-20'*)

Wrapper function for the 'spd_to_rg' color gamut area index (IES TM30-18 = TM30-20).

Args:

        SPD

            ndarray with spectral data (can be multiple SPDs,

            first axis are the wavelengths)

        wl

            None, optional

            Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.

            None: default to no interpolation

        out

            'Rg' or str, optional

            Specifies requested output (e.g. 'Rg,Rf,Rfi,cct,duv')

Returns:

        returns

            float or ndarray with IES TM30-20 Rg for :out: 'Rg'

            Other output is also possible by changing the :out: str value.

References: 1. IES TM30 (99, 4880 spectrally uniform samples)

    2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.

    3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

`luxpy.color.cri.`**`spd_to_iesrf_tm30_15`**(*SPD*, *out='Rf'*, *wl=None*, *cri_type='iesrf-tm30-15'*)

Wrapper function for the 'iesrf' color fidelity index (IES TM30-15).

Args:

        SPD

            ndarray with spectral data (can be multiple SPDs,

            first axis are the wavelengths)

        wl

            None, optional

            Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.

            None: default to no interpolation

        out

            'Rf' or str, optional

            Specifies requested output (e.g. 'Rf,Rfi,cct,duv')

Returns:

        returns

            float or ndarray with IES TM30-15 Rf for :out: 'Rf'

            Other output is also possible by changing the :out: str value.

References: 1. IES TM30 (99, 4880 spectrally uniform samples)

    2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.

    3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

`luxpy.color.cri.`**`spd_to_iesrg_tm30_15`**(*SPD*, *out='Rg'*, *wl=None*, *cri_type='iesrf-tm30-15'*)

> Wrapper function for the 'spd_to_rg' color gamut area index (IES TM30-15).
>
> **Args:**
>> **SPD**
>>
>>> ndarray with spectral data (can be multiple SPDs,
>>>
>>> first axis are the wavelengths)
>>
>> **wl**
>>
>>> None, optional
>>>
>>> Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
>>>
>>> None: default to no interpolation
>>
>> **out**
>>
>>> 'Rg' or str, optional
>>>
>>> Specifies requested output (e.g. 'RgRf,Rfi,cct,duv')
>
> **Returns:**
>> **returns**
>>
>>> float or ndarray with IES TM30-15 Rg for :out: 'Rg'
>>>
>>> Other output is also possible by changing the :out: str value.
>
> **References:** 1. IES TM30 (99, 4880 spectrally uniform samples)
>
>> 2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.
>>
>> 3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

`luxpy.color.cri.`**`spd_to_iesrf_tm30_18`**(*SPD*, *out='Rf'*, *wl=None*, *cri_type='iesrf-tm30-18'*)

> Wrapper function for the 'iesrf' color fidelity index (IES TM30-18).
>
> **Args:**
>> **SPD**
>>
>>> ndarray with spectral data (can be multiple SPDs,
>>>
>>> first axis are the wavelengths)
>>
>> **wl**
>>
>>> None, optional
>>>
>>> Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
>>>
>>> None: default to no interpolation
>>
>> **out**
>>
>>> 'Rf' or str, optional
>>>
>>> Specifies requested output (e.g. 'Rf,Rfi,cct,duv')
>
> **Returns:**
>> **returns**
>>
>>> float or ndarray with IES TM30-18 Rf for :out: 'Rf'
>>>
>>> Other output is also possible by changing the :out: str value.
>
> **References:** 1. IES TM30 (99, 4880 spectrally uniform samples)
>
>> 2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.
>>
>> 3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

luxpy.color.cri.**spd_to_iesrg_tm30_18**(*SPD*, *out='Rg'*, *wl=None*, *cri_type='iesrf-tm30-18'*)

> Wrapper function for the 'spd_to_rg' color gamut area index (IES TM30-18).
>
> > **Args:**
> >
> > > **SPD**
> > >
> > > > ndarray with spectral data (can be multiple SPDs,
> > > >
> > > > first axis are the wavelengths)
> > >
> > > **wl**
> > >
> > > > None, optional
> > > >
> > > > Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
> > > >
> > > > None: default to no interpolation
> > >
> > > **out**
> > >
> > > > 'Rg' or str, optional
> > > >
> > > > Specifies requested output (e.g. 'Rg,Rf,Rfi,cct,duv')
> >
> > **Returns:**
> >
> > > **returns**
> > >
> > > > float or ndarray with IES TM30-18 Rg for :out: 'Rg'
> > > >
> > > > Other output is also possible by changing the :out: str value.
>
> **References:** 1. IES TM30 (99, 4880 spectrally uniform samples)
>
> > 2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.
> >
> > 3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

luxpy.color.cri.**spd_to_iesrf_tm30_20**(*SPD*, *out='Rf'*, *wl=None*, *cri_type='iesrf-tm30-20'*)

> Wrapper function for the 'iesrf' color fidelity index (IES TM30-20 = TM30-18).
>
> > **Args:**
> >
> > > **SPD**
> > >
> > > > ndarray with spectral data (can be multiple SPDs,
> > > >
> > > > first axis are the wavelengths)
> > >
> > > **wl**
> > >
> > > > None, optional
> > > >
> > > > Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
> > > >
> > > > None: default to no interpolation
> > >
> > > **out**
> > >
> > > > 'Rf' or str, optional
> > > >
> > > > Specifies requested output (e.g. 'Rf,Rfi,cct,duv')
> >
> > **Returns:**
> >
> > > **returns**
> > >
> > > > float or ndarray with IES TM30-20 Rf for :out: 'Rf'
> > > >
> > > > Other output is also possible by changing the :out: str value.
>
> **References:** 1. IES TM30 (99, 4880 spectrally uniform samples)
>
> > 2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.
> >
> > 3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

luxpy.color.cri.**spd_to_iesrg_tm30_20**(*SPD*, *out='Rg'*, *wl=None*, *cri_type='iesrf-tm30-20'*)

Wrapper function for the 'spd_to_rg' color gamut area index (IES TM30-18 = TM30-20).

Args:

> **SPD**
>
>> ndarray with spectral data (can be multiple SPDs,
>>
>> first axis are the wavelengths)
>
> **wl**
>
>> None, optional
>>
>> Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
>>
>> None: default to no interpolation
>
> **out**
>
>> 'Rg' or str, optional
>>
>> Specifies requested output (e.g. 'Rg,Rf,Rfi,cct,duv')

Returns:

> **returns**
>
>> float or ndarray with IES TM30-20 Rg for :out: 'Rg'
>>
>> Other output is also possible by changing the :out: str value.

References: 1. IES TM30 (99, 4880 spectrally uniform samples)

> 2. A. David, P. T. Fini, K. W. Houser, Y. Ohno, M. P. Royer, K. A. G. Smet, M. Wei, and L. Whitehead, "Development of the IES method for evaluating the color rendition of light sources," Opt. Express, vol. 23, no. 12, pp. 15888–15906, 2015.
>
> 3. K. A. G. Smet, A. David, and L. Whitehead, "Why color space uniformity and sample set spectral uniformity are essential for color rendering measures," LEUKOS, vol. 12, no. 1–2, pp. 39–50, 2016

luxpy.color.cri.**spd_to_cri2012**(*SPD*, *out='Rf'*, *wl=None*)

Wrapper function for the 'cri2012' color rendition (fidelity) metric with the spectally uniform HL17 mathematical sampleset.

Args:

> **SPD**
>
>> ndarray with spectral data (can be multiple SPDs,
>>
>> first axis are the wavelengths)
>
> **wl**
>
>> None, optional
>>
>> Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
>>
>> None: default to no interpolation
>
> **out**
>
>> 'Rf' or str, optional
>>
>> Specifies requested output (e.g. 'Rf,Rfi,cct,duv')

Returns:

> **returns**
>
>> float or ndarray with CRI2012 Rf for :out: 'Rf'
>>
>> Other output is also possible by changing the :out: str value.

References:

> ..[1] Smet, K., Schanda, J., Whitehead, L., & Luo, R. (2013). CRI2012: A proposal for updating the CIE colour rendering index. Lighting Research and Technology, 45, 689–709. Retrieved from http://lrt.sagepub.com/content/45/6/689

luxpy.color.cri.**spd_to_cri2012_hl17**(*SPD*, *out='Rf'*, *wl=None*)

> Wrapper function for the 'cri2012' color rendition (fidelity) metric with the spectally uniform HL17 mathematical sampleset.
>
> **Args:**
>> **SPD**
>>
>>> ndarray with spectral data (can be multiple SPDs,
>>>
>>> first axis are the wavelengths)
>>
>> **wl**
>>
>>> None, optional
>>>
>>> Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
>>>
>>> None: default to no interpolation
>>
>> **out**
>>
>>> 'Rf' or str, optional
>>>
>>> Specifies requested output (e.g. 'Rf,Rfi,cct,duv')
>
> **Returns:**
>> **returns**
>>
>>> float or ndarray with CRI2012 Rf for :out: 'Rf'
>>>
>>> Other output is also possible by changing the :out: str value.
>
> **Reference:** 1. Smet, K., Schanda, J., Whitehead, L., & Luo, R. (2013). CRI2012: A proposal for updating the CIE colour rendering index. Lighting Research and Technology, 45, 689–709.

luxpy.color.cri.**spd_to_cri2012_hl1000**(*SPD*, *out='Rf'*, *wl=None*)

> Wrapper function for the 'cri2012' color rendition (fidelity) metric with the spectally uniform Hybrid HL1000 sampleset.
>
> **Args:**
>> **SPD**
>>
>>> ndarray with spectral data (can be multiple SPDs,
>>>
>>> first axis are the wavelengths)
>>
>> **wl**
>>
>>> None, optional
>>>
>>> Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
>>>
>>> None: default to no interpolation
>>
>> **out**
>>
>>> 'Rf' or str, optional
>>>
>>> Specifies requested output (e.g. 'Rf,Rfi,cct,duv')
>
> **Returns:**
>> **returns**
>>
>>> float or ndarray with CRI2012 Rf for :out: 'Rf'
>>>
>>> Other output is also possible by changing the :out: str value.
>
> **Reference:** 1. Smet, K., Schanda, J., Whitehead, L., & Luo, R. (2013). CRI2012: A proposal for updating the CIE colour rendering index. Lighting Research and Technology, 45, 689–709.

luxpy.color.cri.**spd_to_cri2012_real210**(*SPD*, *out='Rf'*, *wl=None*)

> Wrapper function the 'cri2012' color rendition (fidelity) metric with the Real-210 sampleset (normally for special color rendering indices).
>
> **Args:**
>> **SPD**
>>
>>> ndarray with spectral data (can be multiple SPDs,
>>>
>>> first axis are the wavelengths)

wl

> None, optional
>
> Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
>
> None: default to no interpolation

out

> 'Rf' or str, optional
>
> Specifies requested output (e.g. 'Rf,Rfi,cct,duv')

Returns:

returns

> float or ndarray with CRI2012 Rf for :out: 'Rf'
>
> Other output is also possible by changing the :out: str value.

Reference: 1. Smet, K., Schanda, J., Whitehead, L., & Luo, R. (2013). CRI2012: A proposal for updating the CIE colour rendering index. Lighting Research and Technology, 45, 689–709.

luxpy.color.cri.**spd_to_mcri**(*SPD*, *D=0.9*, *E=None*, *Yb=20.0*, *out='Rm'*, *wl=None*, *mcri_defaults=None*)

Calculates the MCRI or Memory Color Rendition Index, Rm

Args:

SPD

> ndarray with spectral data (can be multiple SPDs,
>
> > first axis are the wavelengths)

D

> 0.9, optional
>
> Degree of adaptation.

E

> None, optional
>
> Illuminance in lux
>
> > (used to calculate La = (Yb/100)*(E/pi) to then calculate D
> > following the 'cat02' model).
>
> If None: the degree is determined by :D:
>
> > If (:E: is not None) & (:Yb: is None): :E: is assumed to contain
> > the adapting field luminance La (cd/m^2).

Yb

> 20.0, optional
>
> Luminance factor of background. (used when calculating La from E)
>
> If None, E contains La (cd/m^2).

out

> 'Rm' or str, optional
>
> Specifies requested output (e.g. 'Rm,Rmi,cct,duv')

wl

> None, optional
>
> Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
>
> None: default to no interpolation

mcri_defaults

> None, optional
>
> Dictionary with structure of _MCRI_DEFAULTS containing everything
>
> needed to calculate MCRI.

If None: _MCRI_DEFAULTS is used.

Returns:

returns

float or ndarray with MCRI Rm for :out: 'Rm'

Other output is also possible by changing the :out: str value.

References: 1. K.A.G. Smet, W.R. Ryckaert, M.R. Pointer, G. Deconinck, P. Hanselaer,(2012) "A memory colour quality metric for white light sources," Energy Build., vol. 49, no. C, pp. 216–225.

luxpy.color.cri.**spd_to_cqs**(*SPD*, *version='v9.0'*, *out='Qa'*, *wl=None*)

Calculates CQS Qa (Qai) or Qf (Qfi) or Qp (Qpi) for versions v9.0 or v7.5.

Args:

SPD

ndarray with spectral data (can be multiple SPDs,

first axis are the wavelengths)

version

'v9.0' or 'v7.5', optional

out

'Qa' or str, optional

Specifies requested output (e.g. 'Qa,Qai,Qf,cct,duv')

wl

None, optional

Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.

None: default to no interpolation

Returns:

returns

float or ndarray with CQS Qa for :out: 'Qa'

Other output is also possible by changing the :out: str value.

References: 1. W. Davis and Y. Ohno, "Color quality scale," (2010), Opt. Eng., vol. 49, no. 3, pp. 33602–33616.

luxpy.color.cri.**spd_to_fci**(*spd*, *use_cielab=True*)

Calculate Feeling of Contrast Index (FCI).

Args:

spd

ndarray with spectral power distribution(s) of the test light source(s).

use_cielab

True, optional

True: use original formulation of FCI, which adopts a CIECAT94

chromatic adaptation transform followed by a conversion to

CIELAB coordinates before calculating the gamuts.

False: use CIECAM02 coordinates and embedded CAT02 transform.

Returns:

fci

ndarray with FCI values.

References: 1. Hashimoto, K., Yano, T., Shimizu, M., & Nayatani, Y. (2007). New method for specifying color-rendering properties of light sources based on feeling of contrast. Color Research and Application, 32(5), 361–371.

luxpy.color.cri.**spd_to_thornton_cpi**(*spd*)

    Calculate Thornton's Color Preference Index (CPI).

    Args:

        spd

            nd array with spectral power distribution(s) of the test light source(s).

    Returns:

        cpi

            ndarray with CPI values.

    Reference: 1. Thornton, W. A. (1974). A Validation of the Color-Preference Index. Journal of the Illuminating Engineering Society, 4(1), 48–52.

luxpy.color.cri.**plot_hue_bins**(*hbins=16*, *start_hue=0.0*, *scalef=100*, *plot_axis_labels=False*, *bin_labels='#'*, *plot_edge_lines=True*, *plot_center_lines=False*, *plot_bin_colors=True*, *plot_10_20_circles=False*, *axtype='polar'*, *ax=None*, *force_CVG_layout=False*, *hbin_color_map=None*)

    Makes basis plot for Color Vector Graphic (CVG).

    Args:

        hbins

            16 or ndarray with sorted hue bin centers (°), optional

        start_hue

            0.0, optional

        scalef

            100, optional

            Scale factor for graphic.

        plot_axis_labels

            False, optional

            Turns axis ticks on/off (True/False).

        bin_labels

            None or list[str] or '#', optional

            Plots labels at the bin center hues.

                 - None: don't plot.

                - list[str]: list with str for each bin.

                      (len(:bin_labels:) = :nhbins:)

                - '#': plots number.

        plot_edge_lines

            True or False, optional

            Plot grey bin edge lines with '–'.

        plot_center_lines

            False or True, optional

            Plot colored lines at 'center' of hue bin.

        plot_bin_colors

            True, optional

            Colorize hue bins.

        plot_10_20_circles

            False, optional

            If True and :axtype: == 'cart': Plot white circles at

> 80%, 90%, 100%, 110% and 120% of :scalef:

axtype

> 'polar' or 'cart', optional
> Make polar or Cartesian plot.

ax

> None or 'new' or 'same', optional
>> - None or 'new' creates new plot
>> - 'same': continue plot on same axes.
>> - axes handle: plot on specified axes.

force_CVG_layout

> False or True, optional
> True: Force plot of basis of CVG on first encounter.

hbin_color_map

> ndarray with predefined RGB color map
> If None or hbin_color_map.shape[0]<nhbins: cmap will be created, else use values in ndarray.

> Returns:

returns

> gcf(), gca(), list with rgb colors for hue bins (for use in other plotting fcns)

luxpy.color.cri.**plot_ColorVectorGraphic**(*jabt*, *jabr*, *hbins=16*, *start_hue=0.0*, *scalef=100*, *plot_axis_labels=False*, *bin_labels=None*, *plot_edge_lines=True*, *plot_center_lines=False*, *plot_bin_colors=True*, *plot_10_20_circles=True*, *plot_vectors=True*, *gamut_line_color=None*, *gamut_line_style='-'*, *gamut_line_marker='o'*, *gamut_line_label=None*, *axtype='polar'*, *ax=None*, *force_CVG_layout=False*, *hbin_color_map=None*, *hvector_color_map=None*, *jabti=None*, *jabri=None*, *hbinnr=None*)

Plot Color Vector Graphic (CVG).

> Args:

jabt

> ndarray with jab data under test SPD

jabr

> ndarray with jab data under reference SPD

hbins

> 16 or ndarray with sorted hue bin centers (°), optional

start_hue

> 0.0, optional

scalef

> 100, optional
> Scale factor for graphic.

plot_axis_labels

> False, optional
> Turns axis ticks on/off (True/False).

bin_labels

> None or list[str] or '#', optional
> Plots labels at the bin center hues.
>> - None: don't plot.
>> - list[str]: list with str for each bin.
>>> (len(:bin_labels:) = :nhbins:)
>> - '#': plots number.

plot_edge_lines

> True or False, optional
> Plot grey bin edge lines with '–'.

plot_center_lines

> False or True, optional
> Plot colored lines at 'center' of hue bin.

plot_bin_colors

> True, optional
> Colorize hue-bins.

plot_10_20_circles

> True, optional
> If True and :axtype: == 'cart': Plot white circles at
> 80%, 90%, 100%, 110% and 120% of :scalef:

plot_vectors

> True, optional
> True: plot vectors from reference to test colors.

gamut_line_color

> 'grey', optional
> Color to plot the test color gamut in.

gamut_line_style

> '-', optional
> Line style to plot the test color gamut in.

gamut_line_marker

> 'o', optional
> Markers to plot the test color gamut points for each hue bin in
> (only used when plot_vectors = False).

gamut_line_label

> None, optional
> Label for gamut line. (only used when plot_vectors = False).

axtype

> 'polar' or 'cart', optional
> Make polar or Cartesian plot.

ax

> None or 'new' or 'same', optional
>> - None or 'new' creates new plot
>> - 'same': continue plot on same axes.
>> - axes handle: plot on specified axes.

force_CVG_layout

> False or True, optional

> True: Force plot of basis of CVG.

hbin_color_map

> ndarray with predefined RGB color map for the hue bins
>
> If None or hbin_color_map.shape[0]<nhbins: cmap will be created, else use values in ndarray.

hvector_color_map

> ndarray with predefined RGB color map for the color shift vectors in each hue bin.
>
> If None or hvector_color_map.shape[0]<hbins: cmap will be created, else use values in ndarray.

jabti

> None, optional
>
> ndarray with jab data of all samples under test SPD (scaled to 'unit' circle)
>
> If not None: plot chromaticity coordinates of test samples relative to
>
> the mean chromaticity of the samples under the reference illuminant.

jabri

> None, optional
>
> ndarray with jab data of all samples under reference SPD (scaled to 'unit' circle)
>
> Must be supplied when jabti is not None!

hbinnr

> None, optional
>
> ndarray with hue bin number of each sample.
>
> Must be supplied when jabti is not None!

> **Returns:**

returns

> gcf(), gca(), list with rgb colors for hue bins (for use in
>
> other plotting fcns)

luxpy.color.cri.**spd_to_ies_tm30_metrics**(*St, cri_type=None, hbins=16, start_hue=0.0, scalef=100, vf_model_type='M6', vf_pcolorshift={'Cref': 40, 'href': array([3.7835e+00, 3.3161e+00, 2.8272e+00, 1.9093e+00, 5.2787e+00, 4.3081e+00, 3.7762e-01, 6.2055e+00, 1.4564e+00, 8.8927e-01]), 'labels': array(['5B', '5BG', '5G', '5GY', '5P', '5PB', '5R', '5RP', '5Y', '5YR'], dtype=object), 'sig': 0.3}, scale_vf_chroma_to_sample_chroma=False*)

Calculates IES TM30 metrics from spectral data.

> **Args:**

St

> numpy.ndarray with spectral data

cri_type

> None, optional
>
> If None: defaults to cri_type = 'iesrf'.
>
> Not none values of :hbins:, :start_hue: and :scalef: overwrite
>
> input in cri_type['rg_pars']

hbins

> None or numpy.ndarray with sorted hue bin centers (°), optional

---

> **start_hue**
>
>> None, optional
>
> **scalef**
>
>> None, optional
>> Scale factor for reference circle.
>
> **vf_pcolorshift**
>
>> _VF_PCOLORSHIFT or user defined dict, optional
>> The polynomial models of degree 5 and 6 can be fully specified or
>> summarized by the model parameters themselved OR by calculating the
>> dCoverC and dH at resp. 5 and 6 hues. :VF_pcolorshift: specifies
>> these hues and chroma level.
>
> **scale_vf_chroma_to_sample_chroma**
>
>> False, optional
>> Scale chroma of reference and test vf fields such that average of
>> binned reference chroma equals that of the binned sample chroma
>> before calculating hue bin metrics.

Returns:

> **data**
>
>> Dictionary with color rendering data:
>>
>> - 'St, Sr' : ndarray of test SPDs and corresponding ref. illuminants.
>> - 'xyz_cct': xyz of white point calculate with cieobs defined for cct calculations in cri_type['cieobs']
>> - 'cct, duv': CCT and Duv obtained with cieobs in cri_type['cieobs']['cct']
>> - 'xyzti, xyzri': ndarray tristimulus values of test and ref. samples (obtained with with cieobs in cri_type['cieobs']['xyz'])
>> - 'xyztw, xyzrw': ndarray tristimulus values of test and ref. white points (obtained with with cieobs in cri_type['cieobs']['xyz'])
>> - 'DEi, DEa': ndarray with individual sample color differences DEi and average DEa between test and ref.
>> - 'Rf' : ndarray with general color fidelity index values
>> - 'Rg' : ndarray with color gamut area index values
>> - 'Rfi' : ndarray with specific (sample) color fidelity indices
>> - 'Rfhj' : ndarray with local (hue binned) fidelity indices
>> - 'DEhj' : ndarray with local (hue binned) color differences
>> - 'Rcshj': ndarray with local chroma shifts indices
>> - 'Rhshj': ndarray with local hue shifts indices
>> - 'hue_bin_data': dict with output from _get_hue_bin_data() [see its help for more info]
>> - 'cri_type': same as input (for reference purposes)
>> - 'vf' : dictionary with vector field measures and data.
>>> Keys:
>>>> - 'Rt' : ndarray with general metameric uncertainty index Rt
>>>> - 'Rti' : ndarray with specific metameric uncertainty indices Rti
>>>> - 'Rfhj' : ndarray with local (hue binned) fidelity indices
>>>>> obtained from VF model predictions at color space

> pixel coordinates
>
> - 'DEhj' : ndarray with local (hue binned) color differences
>
>> (same as above)
>
> - 'Rcshj': ndarray with local chroma shifts indices for vectorfield coordinates
>
>> (same as above)
>
> - 'Rhshj': ndarray with local hue shifts indicesfor vectorfield coordinates
>
>> (same as above)
>
> - 'Rfi': ndarray with sample fidelity indices for vectorfield coordinates
>
>> (same as above)
>
> - 'DEi': ndarray with sample color differences for vectorfield coordinates
>
>> (same as above)
>
> - 'hue_bin_data': dict with output from _get_hue_bin_data() for vectorfield coordinates
>
> - 'dataVF': dictionary with output of cri.VFPX.VF_colorshift_model()

luxpy.color.cri.**_tm30_process_spd**(*spd*, *cri_type='ies-tm30'*, ***kwargs*)

> Calculate all required parameters for plotting from spd using cri.spd_to_cri()
>
> **Args:**
>
>> **spd**
>>
>>> ndarray or dict
>>>
>>> If ndarray: single spectral power distribution.
>>>
>>> If dict: dictionary with pre-computed parameters.
>>>
>>>> required keys:
>>>>
>>>>> dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',
>>>>>
>>>>>> 'xyzti', 'xyztw', 'xyzri', 'xyzrw',
>>>>>>
>>>>>> 'DEi', 'DEa', 'Rf', 'Rg',
>>>>>>
>>>>>> 'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])
>>>
>>> see cri.spd_to_cri() for more info on parameters.
>>
>> **cri_type**
>>
>>> _CRI_TYPE_DEFAULT or str or dict, optional
>>>
>>>> -'str: specifies dict with default cri model parameters
>>>>
>>>>> (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
>>>>
>>>> - dict: user defined model parameters
>>>>
>>>>> (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
>>>>>
>>>>> for required structure)
>>>
>>> Note that any non-None input arguments (in kwargs)
>>>
>>> to the function will override default values in cri_type dict.
>>
>> **kwargs**
>>
>>> Additional optional keyword arguments,
>>>
>>> the same as in cri.spd_to_cri()
>
> **Returns:**
>
>> **data**
>>
>>> dictionary with required parameters for plotting functions.

luxpy.color.cri.**plot_tm30_cvg**(*spd, cri_type='ies-tm30', gamut_line_color=None, gamut_line_style='-', gamut_line_marker='o', gamut_line_label=None, plot_vectors=True, plot_index_values=True, axh=None, axtype='cart', show_annexE_priority=True, show_Rcsh1_Rfh1=True, **kwargs*)

> Plot TM30 Color Vector Graphic (CVG).
> **Args:**
> > **spd**
> > > ndarray or dict
> > > If ndarray: single spectral power distribution.
> > > If dict: dictionary with pre-computed parameters (using _tm30_process_spd()).
> > > > required keys:
> > > > > dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',
> > > > > > 'xyzti', 'xyztw', 'xyzri', 'xyzrw',
> > > > > > 'DEi', 'DEa', 'Rf', 'Rg',
> > > > > > 'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])
> > > see cri.spd_to_cri() for more info on parameters.
> > **cri_type**
> > > _CRI_TYPE_DEFAULT or str or dict, optional
> > > > -'str: specifies dict with default cri model parameters
> > > > > (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
> > > > - dict: user defined model parameters
> > > > > (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
> > > > > for required structure)
> > > Note that any non-None input arguments (in kwargs)
> > > to the function will override default values in cri_type dict.
> > **gamut_line_color**
> > > 'r', optional
> > > Plotting line style for the line connecting the
> > > average test chromaticity in the hue bins.
> > > None defaults to red (240,80,70)/255 (IES-TM30-20 recommended).
> > **gamut_line_style**
> > > '-', optional
> > > Plotting color for the line connecting the
> > > average test chromaticity in the hue bins.
> > **gamut_line_marker**
> > > '-', optional
> > > Markers to plot the test color gamut points for each hue bin in
> > > (only used when plot_vectors = False).
> > **gamut_line_label**
> > > None, optional
> > > Label for gamut line. (only used when plot_vectors = False).
> > **plot_vectors**
> > > True, optional
> > > Plot color shift vectors in CVG (True) or not (False).
> > **plot_index_values**

True, optional

Print Rf, Rg, CCT and Duv in corners of CVG (True) or not (False).

If False: turns of potential prints of Rcsh1, Rfh1

and annexE_priority levelels as well. This way this argument can be

easily used to turn off all plotting and printing when graphs are

to be generated with gamuts of multiple sources.

axh

None, optional

If None: create new figure with single axes, else plot on specified axes.

axtype

'cart' (or 'polar'), optional

Make Cartesian (default) or polar plot.

show_annexE_priority

True, optional

Add Annex E priority levels for source.

show_Rcsh1_Rfh1

True, optional

Add the local chroma shift (%) and the local color fidelity index

for hue bin 1 at the bottom of the graph.

kwargs

Additional optional keyword arguments,

the same as in cri.spd_to_cri()

Returns:

axh

handle to figure axes.

data

dictionary with required parameters for plotting functions.

luxpy.color.cri.**plot_tm30_Rfi**(*spd*, *cri_type='ies-tm30'*, *axh=None*, *font_size=11*, ***kwargs*)

Plot Sample Color Fidelity values (Rfi).

Args:

spd

ndarray or dict

If ndarray: single spectral power distribution.

If dict: dictionary with pre-computed parameters (using _tm30_process_spd()).

required keys:

dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',

'xyzti', 'xyztw', 'xyzri', 'xyzrw',

'DEi', 'DEa', 'Rf', 'Rg',

'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])

see cri.spd_to_cri() for more info on parameters.

cri_type

_CRI_TYPE_DEFAULT or str or dict, optional

-'str: specifies dict with default cri model parameters

(for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])

- dict: user defined model parameters

> > (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
> >
> > for required structure)
>
> Note that any non-None input arguments (in kwargs)
>
> to the function will override default values in cri_type dict.

> **axh**
>
> > None, optional
> >
> > If None: create new figure with single axes, else plot on specified axes.

> **font_size**
>
> > _TM30_FONT_SIZE, optional
> >
> > Font size of text, axis labels and axis values.

> **kwargs**
>
> > Additional optional keyword arguments,
> >
> > the same as in cri.spd_to_cri()

> **Returns:**
>
> > **axh**
> >
> > > handle to figure axes.
> >
> > **data**
> >
> > > dictionary with required parameters for plotting functions.

luxpy.color.cri.**plot_tm30_Rxhj**(*spd*, *cri_type='ies-tm30'*, *axh=None*, *figsize=(6, 15)*, *font_size=11, **kwargs*)

> Plot Local Chroma Shifts (Rcshj), Local Hue Shifts (Rhshj) and Local Color Fidelity values (Rfhj) (one for each hue-bin).

> **Args:**
>
> > **spd**
> >
> > > ndarray or dict
> > >
> > > If ndarray: single spectral power distribution.
> > >
> > > If dict: dictionary with pre-computed parameters (using _tm30_process_spd()).
> > >
> > > > required keys:
> > > >
> > > > > dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',
> > > > >
> > > > > > 'xyzti', 'xyztw', 'xyzri', 'xyzrw',
> > > > > >
> > > > > > 'DEi', 'DEa', 'Rf', 'Rg',
> > > > > >
> > > > > > 'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])
> > >
> > > see cri.spd_to_cri() for more info on parameters.

> > **cri_type**
> >
> > > _CRI_TYPE_DEFAULT or str or dict, optional
> > >
> > > > -'str: specifies dict with default cri model parameters
> > > >
> > > > > (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
> > > >
> > > > - dict: user defined model parameters
> > > >
> > > > > (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
> > > > >
> > > > > for required structure)
> > >
> > > Note that any non-None input arguments (in kwargs)
> > >
> > > to the function will override default values in cri_type dict.

> > **axh**
> >
> > > None, optional
> > >
> > > If None: create new figure with single axes, else plot on specified axes.

> > **figsize**

(6,15), optional

Figure size of pyplot figure.

font_size

_TM30_FONT_SIZE, optional

Font size of text, axis labels and axis values.

kwargs

Additional optional keyword arguments,

the same as in cri.spd_to_cri()

Returns:

axh

handle to figure axes.

data

dictionary with required parameters for plotting functions.

luxpy.color.cri.**plot_tm30_Rcshj**(*spd, cri_type='ies-tm30', axh=None, xlabel=True, y_offset=0, font_size=11, **kwargs*)

Plot Local Chroma Shift values (Rcshj) (one for each hue-bin).

Args:

spd

ndarray or dict

If ndarray: single spectral power distribution.

If dict: dictionary with pre-computed parameters (using _tm30_process_spd()).

required keys:

dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',

'xyzti', 'xyztw', 'xyzri', 'xyzrw',

'DEi', 'DEa', 'Rf', 'Rg',

'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])

see cri.spd_to_cri() for more info on parameters.

cri_type

_CRI_TYPE_DEFAULT or str or dict, optional

-'str: specifies dict with default cri model parameters

(for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])

- dict: user defined model parameters

(see e.g. luxpy.cri._CRI_DEFAULTS['cierf']

for required structure)

Note that any non-None input arguments (in kwargs)

to the function will override default values in cri_type dict.

axh

None, optional

If None: create new figure with single axes, else plot on specified axes.

xlabel

True, optional

If False: don't add label and numbers to x-axis

(useful when plotting plotting all 'Local Rfhi, Rcshi, Rshhi'

values in 3x1 subplots with 'shared x-axis': saves vertical space)

y_offset

0, optional

text-offset from top of bars in barplot.

font_size

_TM30_FONT_SIZE, optional

Font size of text, axis labels and axis values.

kwargs

Additional optional keyword arguments,

the same as in cri.spd_to_cri()

Returns:

axh

handle to figure axes.

data

dictionary with required parameters for plotting functions.

luxpy.color.cri.**plot_tm30_Rhshj**(*spd*, *cri_type='ies-tm30'*, *axh=None*, *xlabel=True*, *y_offset=0*, *font_size=11*, ***kwargs*)

Plot Local Hue Shift values (Rhshj) (one for each hue-bin).

Args:

spd

ndarray or dict

If ndarray: single spectral power distribution.

If dict: dictionary with pre-computed parameters (using _tm30_process_spd()).

required keys:

dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',

'xyzti', 'xyztw', 'xyzri', 'xyzrw',

'DEi', 'DEa', 'Rf', 'Rg',

'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])

see cri.spd_to_cri() for more info on parameters.

cri_type

_CRI_TYPE_DEFAULT or str or dict, optional

-'str: specifies dict with default cri model parameters

(for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])

- dict: user defined model parameters

(see e.g. luxpy.cri._CRI_DEFAULTS['cierf']

for required structure)

Note that any non-None input arguments (in kwargs)

to the function will override default values in cri_type dict.

axh

None, optional

If None: create new figure with single axes, else plot on specified axes.

xlabel

True, optional

If False: don't add label and numbers to x-axis

(useful when plotting plotting all 'Local Rfhi, Rcshi, Rshhi'

values in 3x1 subplots with 'shared x-axis': saves vertical space)

y_offset

0, optional

text-offset from top of bars in barplot.

> > > **font_size**
> > >
> > > > _TM30_FONT_SIZE, optional
> > > > Font size of text, axis labels and axis values.
> > >
> > > **kwargs**
> > >
> > > > Additional optional keyword arguments,
> > > > the same as in cri.spd_to_cri()
> >
> > **Returns:**
> >
> > > **axh**
> > >
> > > > handle to figure axes.
> > >
> > > **data**
> > >
> > > > dictionary with required parameters for plotting functions.

luxpy.color.cri.**plot_tm30_Rfhj**(*spd*, *cri_type='ies-tm30'*, *axh=None*, *xlabel=True*, *y_offset=0*, *font_size=11*, ***kwargs*)

> Plot Local Color Fidelity values (Rfhj) (one for each hue-bin).
>
> **Args:**
>
> > **spd**
> >
> > > ndarray or dict
> > > If ndarray: single spectral power distribution.
> > > If dict: dictionary with pre-computed parameters (using _tm30_process_spd()).
> > >
> > > > required keys:
> > > >
> > > > > dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',
> > > > > 'xyzti', 'xyztw', 'xyzri', 'xyzrw',
> > > > > 'DEi', 'DEa', 'Rf', 'Rg',
> > > > > 'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])
> > >
> > > see cri.spd_to_cri() for more info on parameters.
> >
> > **cri_type**
> >
> > > _CRI_TYPE_DEFAULT or str or dict, optional
> > >
> > > > -'str: specifies dict with default cri model parameters
> > > >
> > > > > (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
> > > >
> > > > - dict: user defined model parameters
> > > >
> > > > > (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
> > > > > for required structure)
> > >
> > > Note that any non-None input arguments (in kwargs)
> > > to the function will override default values in cri_type dict.
> >
> > **axh**
> >
> > > None, optional
> > > If None: create new figure with single axes, else plot on specified axes.
> >
> > **xlabel**
> >
> > > True, optional
> > > If False: don't add label and numbers to x-axis
> > > (useful when plotting plotting all 'Local Rfhi, Rcshi, Rshhi'
> > >
> > > > values in 3x1 subplots with 'shared x-axis': saves vertical space)
> >
> > **y_offset**
> >
> > > 0, optional
> > > text-offset from top of bars in barplot.
> >
> > **font_size**

> _TM30_FONT_SIZE, optional
>
> Font size of text, axis labels and axis values.
>
> **kwargs**
>
> > Additional optional keyword arguments,
> >
> > the same as in cri.spd_to_cri()

Returns:

> **axh**
>
> > handle to figure axes.
>
> **data**
>
> > dictionary with required parameters for plotting functions.

luxpy.color.cri.**plot_tm30_spd**(*spd*, *cri_type='ies-tm30'*, *axh=None*, *font_size=11*, ***kwargs*)

> Plot test SPD and reference illuminant, both normalized to the same luminous power.
>
> **Args:**
>
> > **spd**
> >
> > > ndarray or dict
> > >
> > > If ndarray: single spectral power distribution.
> > >
> > > If dict: dictionary with pre-computed parameters (using _tm30_process_spd()).
> > >
> > > > required keys:
> > > >
> > > > > dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',
> > > > >
> > > > > > 'xyzti', 'xyztw', 'xyzri', 'xyzrw',
> > > > > >
> > > > > > 'DEi', 'DEa', 'Rf', 'Rg',
> > > > > >
> > > > > > 'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])
> > >
> > > see cri.spd_to_cri() for more info on parameters.
> >
> > **cri_type**
> >
> > > _CRI_TYPE_DEFAULT or str or dict, optional
> > >
> > > > -'str: specifies dict with default cri model parameters
> > > >
> > > > > (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
> > > >
> > > > - dict: user defined model parameters
> > > >
> > > > > (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
> > > > >
> > > > > for required structure)
> > >
> > > Note that any non-None input arguments (in kwargs)
> > >
> > > to the function will override default values in cri_type dict.
> >
> > **axh**
> >
> > > None, optional
> > >
> > > If None: create new figure with single axes, else plot on specified axes.
> >
> > **font_size**
> >
> > > _TM30_FONT_SIZE, optional
> > >
> > > Font size of text, axis labels and axis values.
> >
> > **kwargs**
> >
> > > Additional optional keyword arguments,
> > >
> > > the same as in cri.spd_to_cri()
>
> **Returns:**
>
> > **axh**
> >
> > > handle to figure axes.
> >
> > **data**

> dictionary with required parameters for plotting functions.

`luxpy.color.cri.`**`plot_tm30_report`**(*spd*, *cri_type='ies-tm30'*, *report_type='full'*, *source=''*, *manufacturer=''*, *date=''*, *model=''*, *notes=''*, *max_len_notes_line=40*, *figsize=None*, *save_fig_name=None*, *dpi=300*, *plot_report_top=True*, *plot_report_bottom=True*, *show_annexE_priority=True*, *show_Rcsh1_Rfh1=True*, *suptitle='ANSI/IES TM-30-18 Color Rendition Report'*, *font_size=None*, ***kwargs*)

> Create TM30 Color Rendition Report.
>
> **Args:**
>
> > **spd**
> >
> > > ndarray or dict
> > >
> > > If ndarray: single spectral power distribution.
> > >
> > > If dict: dictionary with pre-computed parameters (using _tm30_process_spd()).
> > >
> > > > required keys:
> > > >
> > > > > dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',
> > > > >
> > > > > > 'xyzti', 'xyztw', 'xyzri', 'xyzrw',
> > > > > >
> > > > > > 'DEi', 'DEa', 'Rf', 'Rg',
> > > > > >
> > > > > > 'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])
> > >
> > > see cri.spd_to_cri() for more info on parameters.
> >
> > **cri_type**
> >
> > > _CRI_TYPE_DEFAULT or str or dict, optional
> > >
> > > > -'str: specifies dict with default cri model parameters
> > > >
> > > > > (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
> > > >
> > > > - dict: user defined model parameters
> > > >
> > > > > (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
> > > > >
> > > > > for required structure)
> > >
> > > Note that any non-None input arguments (in kwargs)
> > >
> > > to the function will override default values in cri_type dict.
> >
> > **report_type**
> >
> > > 'full', optional
> > >
> > > Generate a full report as in ANSI/IES-TM30-2020
> > >
> > > Options :
> > >
> > > > - 'full': full report with spectrum plot, color vector graphic, local indices,
> > > > sample indices'simple', . . .
> > > >
> > > > - 'intermediate': color vector graphic + local chroma and hue shifts
> > > >
> > > > - 'simple': color vector graphic only
> > > >
> > > > - 'spd_cvg': spectrum plot + color vector graphic
> >
> > **source**
> >
> > > string with source name.
> >
> > **manufacturer**
> >
> > > string with source manufacturer.
> >
> > **model**
> >
> > > string with source model.
> >
> > **date**
> >
> > > string with source measurement date.
> >
> > **notes**

string to be split

max_len_notes_line

40, optional

Maximum length of a single line when splitting the string.

figsize

None, optional

Figure size of pyplot figure.

If None a default depending on the report_type is used:

- 'full': (7,12)
- 'intermediate' : (14,6)
- 'simple' : (6,6)
- 'spd_cvg': (14,6)

save_fig_name

None, optional

Filename (+path) to which the report will be saved as an image (png).

If None: don't save, just display.

dpi

300, optional

Dots-Per-Inch of image file (PNG).

plot_report_top

execute _plot_tm30_report_top()

plot_report_bottom

execute _plot_tm30_report_bottom()

show_annexE_priority

True, optional

Add Annex E priority levels for source.

show_Rcsh1_Rfh1

True, optional

Add the local chroma shift (%) and the local color fidelity index

for hue bin 1 at the bottom of the graph.

suptitle

'ANSI/IES TM-30-18 Color Rendition Report' or str, optional

report title (input for plt.suptitle).

font_size

None, optional

Font size of text, axis labels and axis values (adjust when changing figsizes).

Defaults : ('full': _TM30_FONT_SIZE_FULLREPORT, other options:
_TM30_FONT_SIZE)

kwargs

Additional optional keyword arguments,

the same as in cri.spd_to_cri()

Returns:

axs

dictionary with handles to each axes.

data

dictionary with required parameters for plotting functions.

luxpy.color.cri.**spd_to_tm30_report**(*spd,* *cri_type='ies-tm30',* *report_type='full',* *source='',* *manufacturer='',* *date='',* *model='',* *notes='',* *max_len_notes_line=40,* *figsize=None,* *save_fig_name=None,* *dpi=300,* *plot_report_top=True,* *plot_report_bottom=True,* *show_annexE_priority=True,* *show_Rcsh1_Rfh1=True,* *suptitle='ANSI/IES TM-30-18 Color Rendition Report',* *font_size=None,* **kwargs*)

> Create TM30 Color Rendition Report.
>
> **Args:**
>> **spd**
>>
>>> ndarray or dict
>>>
>>> If ndarray: single spectral power distribution.
>>>
>>> If dict: dictionary with pre-computed parameters (using _tm30_process_spd()).
>>>> required keys:
>>>>> dict_keys(['St', 'Sr', 'xyztw_cct', 'cct', 'duv',
>>>>>> 'xyzti', 'xyztw', 'xyzri', 'xyzrw',
>>>>>> 'DEi', 'DEa', 'Rf', 'Rg',
>>>>>> 'Rcshj', 'Rhshj', 'Rfhj', 'hue_bin_data'])
>>> see cri.spd_to_cri() for more info on parameters.
>>
>> **cri_type**
>>
>>> _CRI_TYPE_DEFAULT or str or dict, optional
>>>> -'str: specifies dict with default cri model parameters
>>>>> (for supported types, see luxpy.cri._CRI_DEFAULTS['cri_types'])
>>>> - dict: user defined model parameters
>>>>> (see e.g. luxpy.cri._CRI_DEFAULTS['cierf']
>>>>> for required structure)
>>> Note that any non-None input arguments (in kwargs)
>>>
>>> to the function will override default values in cri_type dict.
>>
>> **report_type**
>>
>>> 'full', optional
>>>
>>> Generate a full report as in ANSI/IES-TM30-2020
>>>
>>> Options :
>>>> - 'full': full report with spectrum plot, color vector graphic, local indices,
>>>> sample indices'simple', . . .
>>>> - 'intermediate': color vector graphic + local chroma and hue shifts
>>>> - 'simple': color vector graphic only
>>>> - 'spd_cvg': spectrum plot + color vector graphic
>>
>> **source**
>>
>>> string with source name.
>>
>> **manufacturer**
>>
>>> string with source manufacturer.
>>
>> **model**
>>
>>> string with source model.
>>
>> **date**
>>
>>> string with source measurement date.
>>
>> **notes**

string to be split

max_len_notes_line

40, optional

Maximum length of a single line when splitting the string.

figsize

None, optional

Figure size of pyplot figure.

If None a default depending on the report_type is used:

- 'full': (7,12)
- 'intermediate' : (14,6)
- 'simple' : (6,6)
-'spd_cvg': (14,6)

save_fig_name

None, optional

Filename (+path) to which the report will be saved as an image (png).

If None: don't save, just display.

dpi

300, optional

Dots-Per-Inch of image file (PNG).

plot_report_top

execute _plot_tm30_report_top()

plot_report_bottom

execute _plot_tm30_report_bottom()

show_annexE_priority

True, optional

Add Annex E priority levels for source.

show_Rcsh1_Rfh1

True, optional

Add the local chroma shift (%) and the local color fidelity index

for hue bin 1 at the bottom of the graph.

suptitle

'ANSI/IES TM-30-18 Color Rendition Report' or str, optional

report title (input for plt.suptitle).

font_size

None, optional

Font size of text, axis labels and axis values (adjust when changing figsizes).

Defaults : ('full': _TM30_FONT_SIZE_FULLREPORT, other options:
_TM30_FONT_SIZE)

kwargs

Additional optional keyword arguments,

the same as in cri.spd_to_cri()

Returns:

axs

dictionary with handles to each axes.

data

dictionary with required parameters for plotting functions.

luxpy.color.cri.**plot_cri_graphics**(*data, cri_type=None, hbins=16, start_hue=0.0,
scalef=100, plot_axis_labels=False, bin_labels=None,
plot_edge_lines=True, plot_center_lines=False,
plot_bin_colors=True, axtype='polar', ax=None,
force_CVG_layout=True, vf_model_type='M6',
vf_pcolorshift={'Cref': 40, 'href': array([3.7835e+00,
3.3161e+00, 2.8272e+00, 1.9093e+00, 5.2787e+00,
4.3081e+00, 3.7762e-01, 6.2055e+00, 1.4564e+00,
8.8927e-01]), 'labels': array(['5B', '5BG', '5G', '5GY',
'5P', '5PB', '5R', '5RP', '5Y', '5YR'], dtype=object),
'sig': 0.3}, vf_color='k', vf_bin_labels=array(['5B',
'5BG', '5G', '5GY', '5P', '5PB', '5R', '5RP', '5Y',
'5YR'], dtype=object), vf_plot_bin_colors=True,
scale_vf_chroma_to_sample_chroma=False,
plot_VF=True, plot_CF=False, plot_SF=False,
plot_test_sample_coord=False*)

Plot graphical information on color rendition properties (custom design).

Args:

> **data**
>
> > ndarray with spectral data or dict with pre-computed metrics.
>
> **cri_type**
>
> > None, optional
> >
> > If None: defaults to cri_type = 'iesrf'.
> >
> > :hbins:, :start_hue: and :scalef: are ignored if cri_type not None
> >
> > and values are replaced by those in cri_type['rg_pars']
>
> **hbins**
>
> > 16 or ndarray with sorted hue bin centers (°), optional
>
> **start_hue**
>
> > 0.0, optional
>
> **scalef**
>
> > 100, optional
> >
> > Scale factor for graphic.
>
> **plot_axis_labels**
>
> > False, optional
> >
> > Turns axis ticks on/off (True/False).
>
> **bin_labels**
>
> > None or list[str] or '#', optional
> >
> > Plots labels at the bin center hues.
> >
> > > - None: don't plot.
> > > - list[str]: list with str for each bin.
> > >   (len(:bin_labels:) = :nhbins:)
> > > - '#': plots number.
>
> **plot_edge_lines**
>
> > True or False, optional
> >
> > Plot grey bin edge lines with '–'.
>
> **plot_center_lines**

False or True, optional

Plot colored lines at 'center' of hue bin.

plot_bin_colors

True, optional

Colorize hue bins.

axtype

'polar' or 'cart', optional

Make polar or Cartesian plot.

ax

None or 'new' or 'same', optional

- None or 'new' creates new plot

- 'same': continue plot on same axes.

- axes handle: plot on specified axes.

force_CVG_layout

True, optional

True: Force plot of basis of CVG.

vf_model_type

_VF_MODEL_TYPE or 'M6' or 'M5', optional

Type of polynomial vector field model to use for the calculation of base color shift and metameric uncertainty.

vf_pcolorshift

_VF_PCOLORSHIFT or user defined dict, optional

The polynomial models of degree 5 and 6 can be fully specified or summarized by the model parameters themselved OR by calculating the dCoverC and dH at resp. 5 and 6 hues. :VF_pcolorshift: specifies these hues and chroma level.

vf_color

'k', optional

For plotting the vector fields.

vf_plot_bin_colors

True, optional

Colorize hue bins of VF graph.

scale_vf_chroma_to_sample_chroma

False, optional

Scale chroma of reference and test vf fields such that average of binned reference chroma equals that of the binned sample chroma before calculating hue bin metrics.

vf_bin_labels

see :bin_labels:

Set VF model hue-bin labels.

plot_CF

False, optional

Plot circle fields.

plot_VF

True, optional

>> Plot vector fields.

> **plot_SF**

>> True, optional
>> Plot sample shifts.

> **plot_test_sample_coord**

>> Plot the coordinates of the samples under the test illuminant
>> relative to the mean chromaticity under the reference illuminant (in the CVG plot).

Returns:

> **returns**

>> (data,
>> [plt.gcf(),ax_spd, ax_CVG, ax_locC, ax_locH, ax_VF],
>> cmap)

>> :data: is a dictionary with color rendering data
>> with keys:
>> - 'St, Sr' : ndarray of test SPDs and corresponding ref. illuminants.
>> - 'xyz_cct': xyz of white point calculate with cieobs defined for cct calculations in cri_type['cieobs']
>> - 'cct, duv': CCT and Duv obtained with cieobs in cri_type['cieobs']['cct']
>> - 'xyzti, xyzri': ndarray tristimulus values of test and ref. samples (obtained with with cieobs in cri_type['cieobs']['xyz'])
>> - 'xyztw, xyzrw': ndarray tristimulus values of test and ref. white points (obtained with with cieobs in cri_type['cieobs']['xyz'])
>> - 'DEi, DEa': ndarray with individual sample color differences DEi and average DEa between test and ref.
>> - 'Rf' : ndarray with general color fidelity index values
>> - 'Rg' : ndarray with color gamut area index values
>> - 'Rfi' : ndarray with specific (sample) color fidelity indices
>> - 'Rfhj' : ndarray with local (hue binned) fidelity indices
>> - 'DEhj' : ndarray with local (hue binned) color differences
>> - 'Rcshj': ndarray with local chroma shifts indices
>> - 'Rhshj': ndarray with local hue shifts indices
>> - 'hue_bin_data': dict with output from _get_hue_bin_data() [see its help for more info]
>> - 'cri_type': same as input (for reference purposes)
>> - 'vf' : dictionary with vector field measures and data.

>>> Keys:

>>>> - 'Rt' : ndarray with general metameric uncertainty index Rt
>>>> - 'Rti' : ndarray with specific metameric uncertainty indices Rti
>>>> - 'Rfhj' : ndarray with local (hue binned) fidelity indices
>>>>> obtained from VF model predictions at color space
>>>>> pixel coordinates
>>>> - 'DEhj' : ndarray with local (hue binned) color differences
>>>>> (same as above)
>>>> - 'Rcshj': ndarray with local chroma shifts indices for vectorfield coordinates
>>>>> (same as above)

> > - 'Rhshj': ndarray with local hue shifts indicesfor vectorfield coordinates
> > (same as above)
> > - 'Rfi': ndarray with sample fidelity indices for vectorfield coordinates
> > (same as above)
> > - 'DEi': ndarray with sample color differences for vectorfield coordinates
> > (same as above)
> > - 'hue_bin_data': dict with output from _get_hue_bin_data() for
> > vectorfield coordinates
> > - 'dataVF': dictionary with output of cri.VFPX.VF_colorshift_model()

> :[. . .]: list with handles to current figure and 5 axes.

> :cmap: list with rgb colors for hue bins (for use in other plotting fcns)

luxpy.color.cri.**spd_to_tm30_fast**(*St*)
> Calculate tm30 measures from spd.

luxpy.color.cri.**cri_ref_fast**(*ccts, wl3=array([360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830]), ref_type='iestm30', mix_range=[4000, 5000], cieobs='1931_2', force_daylight_below4000K=False, n=None, daylight_locus=None, wl=[360, 830, 1]*)

Calculates multiple reference illuminant spectra based on ccts for color rendering index calculations.

luxpy.color.cri.**xyz_to_jab_cam02ucs_fast**(*xyz, xyzw, ucs=True, conditions=None*)

Calculate CAM02-UCS J'a'b' coordinates from xyz tristimulus values of sample and white point.

Args:

> **xyz**
>
>> ndarray with sample tristimulus values
>
> **xyzw**
>
>> ndarray with white point tristimulus values
>
> **conditions**
>
>> None, optional
>> Dictionary with viewing conditions.
>> None results in:

{'La':100, 'Yb':20, 'D':1, 'surround':'avg'}

For more info see luxpy.cam.ciecam02()?

Returns:

> **jab**
>
> > ndarray with J'a'b' coordinates.

4.4.10 cri/VFPX/

> **py**
>
> > - __init__.py
> > - VF_PX_models.py
> > - vectorshiftmodel.py
> > - pixelshiftmodel.py
>
> **namespace** luxpy.cri.VFPX

luxpy.color.cri.VFPX.**get_poly_model**(*jabt*, *jabr*, *modeltype='M6'*)

> Setup base color shift model (delta_a, delta_b), determine model parameters and accuracy.

Calculates a base color shift (delta) from the ref. chromaticity ar, br.

> **Args:**
>
> > **jabt**
> >
> > > ndarray with jab color coordinates under the test SPD.
> >
> > **jabr**
> >
> > > ndarray with jab color coordinates under the reference SPD.
> >
> > **modeltype**
> >
> > > _VF_MODEL_TYPE or 'M6' or 'M5', optional
> > > Specifies degree 5 or degree 6 polynomial model in ab-coordinates.
> > > (see notes below)
>
> **Returns:**
>
> > **returns**
> >
> > > (poly_model,
> > >
> > > > pmodel,
> > > > dab_model,
> > > >
> > > > > dab_res,
> > > > > dCHoverC_res,
> > > > > dab_std,
> > > > > dCHoverC_std)
> > >
> > > :poly_model: function handle to model
> > > :pmodel: ndarray with model parameters
> > > :dab_model: ndarray with ab model predictions from ar, br.
> > > :dab_res: ndarray with residuals between 'da,db' of samples and
> > > > 'da,db' predicted by the model.
> > > :dCHoverC_res: ndarray with residuals between 'dCoverC,dH'

of samples and 'dCoverC,dH' predicted by the model.

Note: dCoverC = (Ct - Cr)/Cr and dH = ht - hr

(predicted from model, see notes below)

:dab_std: ndarray with std of :dab_res:

:dCHoverC_std: ndarray with std of :dCHoverC_res:

Notes:

1. **Model types:**

poly5_model = lambda a,b,p: p[0]*a + p[1]*b + p[2]*(a**2) + p[3]*a*b + p[4]*(b**2)

poly6_model = lambda a,b,p: p[0] + p[1]*a + p[2]*b + p[3]*(a**2) + p[4]*a*b + p[5]*(b**2)

2. **Calculation of dCoverC and dH:**

dCoverC = (np.cos(hr)*da + np.sin(hr)*db)/Cr

dHoverC = (np.cos(hr)*db - np.sin(hr)*da)/Cr

luxpy.color.cri.VFPX.**apply_poly_model_at_x**(*poly_model*, *pmodel*, *axr*, *bxr*)

Applies base color shift model at cartesian coordinates axr, bxr.

Args:

poly_model

function handle to model

pmodel

ndarray with model parameters.

axr

ndarray with a-coordinates under the reference conditions

bxr

ndarray with b-coordinates under the reference conditions

Returns:

returns

(axt,bxt,Cxt,hxt,

axr,bxr,Cxr,hxr)

ndarrays with ab-coordinates, chroma and hue

predicted by the model (xt), under the reference (xr).

luxpy.color.cri.VFPX.**generate_vector_field**(*poly_model*, *pmodel*, *axr=array([-40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40])*, *bxr=array([-40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40])*, *make_grid=True*, *limit_grid_radius=0*, *color='k'*)

Generates a field of vectors using the base color shift model.

Has the option to plot vector field.

Args:

poly_model

function handle to model

pmodel

ndarray with model parameters.

axr

> np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR), optional
>
> Ndarray specifying the a-coordinates at which to apply the model.

bxr

> np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR), optional
>
> Ndarray specifying the b-coordinates at which to apply the model.

make_grid

> True, optional
>
> True: generate a 2d-grid from :axr:, :bxr:.

limit_grid_radius

> 0, optional
>
> > A value of zeros keeps grid as specified by axr,bxr.
> >
> > A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

color

> 'k', optional
>
> For plotting the vector field.
>
> If :color: == 0, no plot will be generated.

Returns:

returns

> If :color: == 0: ndarray of axt,bxt,axr,bxr
>
> Else: handle to axes used for plotting.

luxpy.color.cri.VFPX.**VF_colorshift_model**(*S*, *cri_type='iesrf'*, *model_type='M6'*, *cspace={'Yw': None, 'conditions': {'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, 'mcat': 'cat02', 'type': 'jab_cam02ucs', 'xyzw': None, 'yellow-bluepurplecorrect': None}, sampleset=None, pool=False, pcolorshift={'Cref': 40, 'href': array([3.1416e-01, 9.4248e-01, 1.5708e+00, 2.1991e+00, 2.8274e+00, 3.4558e+00, 4.0841e+00, 4.7124e+00, 5.3407e+00, 5.9690e+00]), 'sig': 0.3}, vfcolor='k', verbosity=0*)

Applies full vector field model calculations to spectral data.

Args:

S

> nump.ndarray with spectral data.

cri_type

> _VF_CRI_DEFAULT or str or dict, optional
>
> Specifies type of color fidelity model to use.
>
> Controls choice of ref. ill., sample set, averaging, scaling, etc.
>
> See luxpy.cri.spd_to_cri for more info.

modeltype

> _VF_MODEL_TYPE or 'M6' or 'M5', optional
>
> Specifies degree 5 or degree 6 polynomial model in ab-coordinates.

cspace

> _VF_CSPACE or dict, optional

Specifies color space. See _VF_CSPACE_EXAMPLE for example structure.

sampleset

None or str or ndarray, optional

Sampleset to be used when calculating vector field model.

pool

False, optional

If :S: contains multiple spectra, True pools all jab data before
modeling the vector field, while False models a different field
for each spectrum.

pcolorshift

default dict (see below) or user defined dict, optional

Dict containing the specification input
for apply_poly_model_at_hue_x().

Default dict = {'href': np.arange(np.pi/10,2*np.pi,2*np.pi/10),

'Cref' : _VF_MAXR,

'sig' : _VF_SIG,

'labels' : '#'}

The polynomial models of degree 5 and 6 can be fully specified or
summarized by the model parameters themselved OR by calculating the
dCoverC and dH at resp. 5 and 6 hues.

vfcolor

'k', optional

For plotting the vector fields.

verbosity

0, optional

Report warnings or not.

Returns:

returns

list[dict] (each list element refers to a different test SPD)
with the following keys:

- 'Source': dict with ndarrays of the S, cct and duv of source spd.
- 'metrics': dict with ndarrays for:
    * Rf (color fidelity: base + metameric shift)
    * Rt (metameric uncertainty index)
    * Rfi (specific color fidelity indices)
    * Rti (specific metameric uncertainty indices)
    * cri_type (str with cri_type)
- 'Jab': dict with with ndarrays for Jabt, Jabr, DEi
- 'dC/C_dH_x_sig' :
    np.vstack((dCoverC_x,dCoverC_x_sig,dH_x,dH_x_sig)).T
    See get_poly_model() for more info.
- 'fielddata': dict with dicts containing data on the calculated
    vector-field and circle-fields:
        * 'vectorfield' : {'axt': vfaxt, 'bxt' : vfbxt,
            'axr' : vfaxr, 'bxr' : vfbxr},
        * 'circlefield' : {'axt': cfaxt, 'bxt' : cfbxt,

'axr' : cfaxr, 'bxr' : cfbxr}},

- 'modeldata' : dict with model info:

{'pmodel': pmodel,

'pcolorshift' : pcolorshift,

'dab_model' : dab_model,

'dab_res' : dab_res,

'dab_std' : dab_std,

'modeltype' : modeltype,

'fmodel' : poly_model,

'Jabtm' : Jabtm,

'Jabrm' : Jabrm,

'DEim' : DEim},

- 'vshifts' :dict with various vector shifts:

* 'Jabshiftvector_r_to_t' : ndarray with difference vectors

between jabt and jabr.

* 'vshift_ab_s' : vshift_ab_s: ab-shift vectors of samples

* 'vshift_ab_s_vf' : vshift_ab_s_vf: ab-shift vectors of

VF model predictions of samples.

* 'vshift_ab_vf' : vshift_ab_vf: ab-shift vectors of VF

model predictions of vector field grid.

luxpy.color.cri.VFPX.**initialize_VF_hue_angles**(*hx=None*, *Cxr=40*, *cri_type='iesrf'*, *modeltype='M6'*, *determine_hue_angles=True*)

Initialize the hue angles that will be used to 'summarize' the VF model fitting parameters.

Args:

> **hx**
>
>> None or ndarray, optional
>>
>> None defaults to Munsell H5 hues.
>
> **Cxr**
>
>> _VF_MAXR, optional
>
> **cri_type**
>
>> _VF_CRI_DEFAULT or str or dict, optional,
>>
>> Cri_type parameters for cri and VF model.
>
> **modeltype**
>
>> _VF_MODEL_TYPE or 'M5' or 'M6', optional
>>
>> Determines the type of polynomial model.
>
> **determine_hue_angles**
>
>> _DETERMINE_HUE_ANGLES or True or False, optional
>>
>> True: determines the 10 primary / secondary Munsell hues ('5..').
>>
>> Note that for 'M6', an additional

Returns:

> **pcolorshift**
>
>> {'href': href,
>>
>> 'Cref' : _VF_MAXR,
>>
>> 'sig' : _VF_SIG,
>>
>> 'labels' : list[str]}

`luxpy.color.cri.VFPX.`**`generate_grid`**(*jab_ranges=None*, *out='grid'*, *ax=array([-40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40])*, *bx=array([-40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40])*, *jx=None*, *limit_grid_radius=0*)

>   Generate a grid of color coordinates.

>   **Args:**

>>   **out**

>>>   'grid' or 'vectors', optional
>>>>   - 'grid': outputs a single 2d numpy.nd-vector with the grid coordinates
>>>>   - 'vector': outputs each dimension seperately.

>>   **jab_ranges**

>>>   None or ndarray, optional
>>>   Specifies the pixelization of color space.
>>>   (ndarray.shape = (3,3), with first axis: J,a,b, and second
>>>   axis: min, max, delta)

>>   **ax**

>>>   default ndarray or user defined ndarray, optional
>>>   default = np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR)

>>   **bx**

>>>   default ndarray or user defined ndarray, optional
>>>   default = np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR)

>>   **jx**

>>>   None, optional
>>>   Note that not-None :jab_ranges: override :ax:, :bx: and :jx input.

>>   **limit_grid_radius**

>>>   0, optional
>>>   A value of zeros keeps grid as specified by axr,bxr.
>>>   A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

>   **Returns:**

>>   **returns**

>>>   single ndarray with ax,bx [,jx]
>>>>   or
>>>   seperate ndarrays for each dimension specified.

`luxpy.color.cri.VFPX.`**`calculate_shiftvectors`**(*jabt*, *jabr*, *average=True*, *vtype='ab'*)

>   Calculate color shift vectors.

>   **Args:**

>>   **jabt**

>>>   ndarray with jab coordinates under the test SPD

>>   **jabr**

>>>   ndarray with jab coordinates under the reference SPD

>>   **average**

>>>   True, optional
>>>   If True, take mean of difference vectors along axis = 0.

>>   **vtype**

>>>   'ab' or 'jab', optional
>>>   Reduce output ndarray to only a,b coordinates of shift vector(s).

Returns:

        returns

        ndarray of (mean) shift vector(s).

`luxpy.color.cri.VFPX.`**`plot_shift_data`**(*data*, *fieldtype='vectorfield'*, *scalef=40*, *color='k'*, *axtype='polar'*, *ax=None*, *hbins=10*, *start_hue=0.0*, *bin_labels='#'*, *plot_center_lines=True*, *plot_axis_labels=False*, *plot_edge_lines=False*, *plot_bin_colors=True*, *force_CVG_layout=True*)

Plots vector or circle fields generated by VFcolorshiftmodel() or PXcolorshiftmodel().

Args:

        data

            dict generated by VFcolorshiftmodel() or PXcolorshiftmodel()

            Must contain 'fielddata'- key, which is a dict with possible keys:

                - key: 'vectorfield': ndarray with vector field data

                - key: 'circlefield': ndarray with circle field data

        color

            'k', optional

            Color for plotting the vector-fields.

        axtype

            'polar' or 'cart', optional

            Make polar or Cartesian plot.

        ax

            None or 'new' or 'same', optional

                - None or 'new' creates new plot

                - 'same': continue plot on same axes.

                - axes handle: plot on specified axes.

        hbins

            16 or ndarray with sorted hue bin centers (°), optional

        start_hue

            _VF_MAXR, optional

            Scale factor for graphic.

        plot_axis_labels

            False, optional

            Turns axis ticks on/off (True/False).

        bin_labels

            None or list[str] or '#', optional

            Plots labels at the bin center hues.

                 - None: don't plot.

                - list[str]: list with str for each bin.

                    (len(:bin_labels:) = :nhbins:)

                - '#': plots number.

        plot_edge_lines

            True or False, optional

            Plot grey bin edge lines with '–'.

        plot_center_lines

            False or True, optional

Plot colored lines at 'center' of hue bin.

plot_bin_colors

True, optional

Colorize hue-bins.

force_CVG_layout

False or True, optional

True: Force plot of basis of CVG.

Returns:

returns

figCVG, hax, cmap

:figCVG: handle to CVG figure

:hax: handle to CVG axes

:cmap: list with rgb colors for hue bins

(for use in other plotting fcns)

luxpy.color.cri.VFPX.**plotcircle**(*radii=array([0, 10, 20, 30, 40, 50]), angles=array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340]), color='k', linestyle='–', out=None*)

Plot one or more concentric circles around (0,0).

Args:

radii

np.arange(0,60,10) or ndarray with radii of circle(s), optional

angles

np.arange(0,350,10) or ndarray with angles (°), optional

color

'k', optional

Color for plotting.

linestyle

'–', optional

Linestyle of circles.

out

None, optional

If None: plot circles, return (x,y) otherwise.

Returns:

x,y

ndarrays with circle coordinates (only returned if out is 'x,y')

luxpy.color.cri.VFPX.**get_pixel_coordinates**(*jab, jab_ranges=None, jab_deltas=None, limit_grid_radius=0*)

Get pixel coordinates corresponding to array of jab color coordinates.

Args:

jab

ndarray of color coordinates

jab_ranges

None or ndarray, optional

Specifies the pixelization of color space.

> (ndarray.shape = (3,3), with first axis: J,a,b, and second axis: min, max, delta)

jab_deltas

float or ndarray, optional

Specifies the sampling range.

A float uses jab_deltas as the maximum Euclidean distance to select

samples around each pixel center. A ndarray of 3 deltas, uses

a city block sampling around each pixel center.

limit_grid_radius

0, optional

A value of zeros keeps grid as specified by axr,bxr.

A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

Returns:

returns

gridp, idxp, jabp, samplenrs, samplesIDs

> - :gridp: ndarray with coordinates of all pixel centers.
> - :idxp: list[int] with pixel index for each non-empty pixel
> - :jabp: ndarray with center color coordinates of non-empty pixels
> - :samplenrs: list[list[int]] with sample numbers belong to each
>   non-empty pixel
> - :sampleIDs: summarizing list,
>   with column order: 'idxp, jabp, samplenrs'

luxpy.color.cri.VFPX.**PX_colorshift_model**(*Jabt*, *Jabr*, *jab_ranges=None*, *jab_deltas=None*,
                                              limit_grid_radius=0)

Pixelates the color space and calculates the color shifts in each pixel.

Args:

Jabt

ndarray with color coordinates under the (single) test SPD.

Jabr

ndarray with color coordinates under the (single) reference SPD.

jab_ranges

None or ndarray, optional

Specifies the pixelization of color space.

(ndarray.shape = (3,3), with first axis: J,a,b, and second

axis: min, max, delta)

jab_deltas

float or ndarray, optional

Specifies the sampling range.

A float uses jab_deltas as the maximum Euclidean distance to select

samples around each pixel center. A ndarray of 3 deltas, uses

a city block sampling around each pixel center.

limit_grid_radius

0, optional

A value of zeros keeps grid as specified by axr,bxr.

A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

Returns:

> **returns**
>
> > dict with the following keys:
> >
> > > - 'Jab': dict with with ndarrays for:
> > >
> > > > Jabt, Jabr, DEi, DEi_ab (only ab-coordinates), DEa (mean)
> > > >
> > > > > and DEa_ab
> > >
> > > - 'vshifts': dict with:
> > >
> > > > * 'vectorshift': ndarray with vector shifts between average
> > > >
> > > > > Jabt and Jabr for each pixel
> > > >
> > > > * 'vectorshift_ab': ndarray with vector shifts averaged
> > > >
> > > > > over J for each pixel
> > > >
> > > > * 'vectorshift_ab_J0': ndarray with vector shifts averaged
> > > >
> > > > > over J for each pixel of J=0 plane.
> > > >
> > > > * 'vectorshift_len': length of 'vectorshift'
> > > > * 'vectorshift_ab_len': length of 'vectorshift_ab'
> > > > * 'vectorshift_ab_J0_len': length of 'vectorshift_ab_J0'
> > > > * 'vectorshift_len_DEnormed': length of 'vectorshift'
> > > >
> > > > > normalized to 'DEa'
> > > >
> > > > * 'vectorshift_ab_len_DEnormed': length of 'vectorshift_ab'
> > > >
> > > > > normalized to 'DEa_ab'
> > > >
> > > > * 'vectorshift_ab_J0_len_DEnormed': length of
> > > > 'vectorshift_ab_J0'
> > > >
> > > > > normalized to 'DEa_ab'
> > >
> > > - 'pixeldata': dict with pixel info:
> > >
> > > > * 'grid' ndarray with coordinates of all pixel centers.
> > > > * 'idx': list[int] with pixel index for each non-empty pixel
> > > > * 'Jab': ndarray with center coordinates of non-empty pixels
> > > > * 'samplenrs': list[list[int]] with sample numbers belong to
> > > >
> > > > > each non-empty pixel
> > > >
> > > > * 'IDs: summarizing list,
> > > >
> > > > > with column order: 'idxp, jabp, samplenrs'
> > >
> > > - 'fielddata' : dict with dicts containing data on the calculated
> > >
> > > > vector-field and circle-fields
> > >
> > > * 'vectorfield': dict with ndarrays for the ab-coordinates
> > >
> > > > under the ref. (axr, bxr) and test (axt, bxt) illuminants,
> > > >
> > > > centered at the pixel centers corresponding to the ab-coordinates of
> > > >
> > > > the reference illuminant.

luxpy.color.cri.VFPX.**calculate_VF_PX_models**(*S,   cri_type='iesrf',   sampleset=None,
pool=False, pcolorshift={'Cref': 40, 'href':
array([3.1416e-01, 9.4248e-01, 1.5708e+00,
2.1991e+00,   2.8274e+00,   3.4558e+00,
4.0841e+00,   4.7124e+00,   5.3407e+00,
5.9690e+00]), 'labels':   '#',   'sig':   0.3},
vfcolor='k', verbosity=0*)

Calculate Vector Field and Pixel color shift models.

> **Args:**
>
> > **cri_type**
> >
> > > _VF_CRI_DEFAULT or str or dict, optional
> > >
> > > Specifies type of color fidelity model to use.

Controls choice of ref. ill., sample set, averaging, scaling, etc.

See luxpy.cri.spd_to_cri for more info.

sampleset

None or str or ndarray, optional

Sampleset to be used when calculating vector field model.

pool

False, optional

If :S: contains multiple spectra, True pools all jab data before

modeling the vector field, while False models a different field

for each spectrum.

pcolorshift

default dict (see below) or user defined dict, optional

Dict containing the specification input

for apply_poly_model_at_hue_x().

Default dict = {'href': np.arange(np.pi/10,2*np.pi,2*np.pi/10),

'Cref' : _VF_MAXR,

'sig' : _VF_SIG,

'labels' : '#'}

The polynomial models of degree 5 and 6 can be fully specified or

summarized by the model parameters themselved OR by calculating the

dCoverC and dH at resp. 5 and 6 hues.

vfcolor

'k', optional

For plotting the vector fields.

verbosity

0, optional

Report warnings or not.

Returns:

returns

:dataVF:, :dataPX:

Dicts, for more info, see output description of resp.:

luxpy.cri.VF_colorshift_model() and luxpy.cri.PX_colorshift_model()

`luxpy.color.cri.VFPX.`**`subsample_RFL_set`**(*rfl,   rflpath='',   samplefcn='rand',*
S=array([[3.6000e+02, 3.6100e+02, 3.6200e+02,
3.6300e+02,   3.6400e+02,   3.6500e+02,
3.6600e+02,   3.6700e+02,   3.6800e+02,
3.6900e+02,   3.7000e+02,   3.7100e+02,
3.7200e+02,   3.7300e+02,   3.7400e+02,
3.7500e+02,   3.7600e+02,   3.7700e+02,
3.7800e+02,   3.7900e+02,   3.8000e+02,
3.8100e+02,   3.8200e+02,   3.8300e+02,
3.8400e+02,   3.8500e+02,   3.8600e+02,
3.8700e+02,   3.8800e+02,   3.8900e+02,
3.9000e+02,   3.9100e+02,   3.9200e+02,
3.9300e+02,   3.9400e+02,   3.9500e+02,
3.9600e+02,   3.9700e+02,   3.9800e+02,
3.9900e+02,   4.0000e+02,   4.0100e+02,
4.0200e+02,   4.0300e+02,   4.0400e+02,
4.0500e+02,   4.0600e+02,   4.0700e+02,
4.0800e+02,   4.0900e+02,   4.1000e+02,
4.1100e+02,   4.1200e+02,   4.1300e+02,
4.1400e+02,   4.1500e+02,   4.1600e+02,
4.1700e+02,   4.1800e+02,   4.1900e+02,
4.2000e+02,   4.2100e+02,   4.2200e+02,
4.2300e+02,   4.2400e+02,   4.2500e+02,
4.2600e+02,   4.2700e+02,   4.2800e+02,
4.2900e+02,   4.3000e+02,   4.3100e+02,
4.3200e+02,   4.3300e+02,   4.3400e+02,
4.3500e+02,   4.3600e+02,   4.3700e+02,
4.3800e+02,   4.3900e+02,   4.4000e+02,
4.4100e+02,   4.4200e+02,   4.4300e+02,
4.4400e+02,   4.4500e+02,   4.4600e+02,
4.4700e+02,   4.4800e+02,   4.4900e+02,
4.5000e+02,   4.5100e+02,   4.5200e+02,
4.5300e+02,   4.5400e+02,   4.5500e+02,
4.5600e+02,   4.5700e+02,   4.5800e+02,
4.5900e+02,   4.6000e+02,   4.6100e+02,
4.6200e+02,   4.6300e+02,   4.6400e+02,
4.6500e+02,   4.6600e+02,   4.6700e+02,
4.6800e+02,   4.6900e+02,   4.7000e+02,
4.7100e+02,   4.7200e+02,   4.7300e+02,
4.7400e+02,   4.7500e+02,   4.7600e+02,
4.7700e+02,   4.7800e+02,   4.7900e+02,
4.8000e+02,   4.8100e+02,   4.8200e+02,
4.8300e+02,   4.8400e+02,   4.8500e+02,
4.8600e+02,   4.8700e+02,   4.8800e+02,
4.8900e+02,   4.9000e+02,   4.9100e+02,
4.9200e+02,   4.9300e+02,   4.9400e+02,
4.9500e+02,   4.9600e+02,   4.9700e+02,
4.9800e+02,   4.9900e+02,   5.0000e+02,
5.0100e+02,   5.0200e+02,   5.0300e+02,
5.0400e+02,   5.0500e+02,   5.0600e+02,
5.0700e+02,   5.0800e+02,   5.0900e+02,
5.1000e+02,   5.1100e+02,   5.1200e+02,
5.1300e+02,   5.1400e+02,   5.1500e+02,
5.1600e+02,   5.1700e+02,   5.1800e+02,
5.1900e+02,   5.2000e+02,   5.2100e+02,
5.2200e+02,   5.2300e+02,   5.2400e+02,
5.2500e+02,   5.2600e+02,   5.2700e+02,
5.2800e+02,   5.2900e+02,   5.3000e+02,
5.3100e+02,   5.3200e+02,   5.3300e+02,
5.3400e+02,   5.3500e+02,   5.3600e+02,

Sub-samples a spectral reflectance set by pixelization of color space.

Args:

> **rfl**
>
>> ndarray or str
>>
>> Array with of str referring to a set of spectral reflectance
>>> functions to be subsampled.
>>
>> If str to file: file must contain data as columns, with first
>>> column the wavelengths.
>
> **rflpath**
>
>> '' or str, optional
>>
>> Path to folder with rfl-set specified in a str :rfl: filename.
>
> **samplefcn**
>
>> 'rand' or 'mean', optional
>>> -'rand': selects a random sample from the samples within each pixel
>>> -'mean': returns the mean spectral reflectance in each pixel.
>
> **S**
>
>> _CIE_ILLUMINANTS['E'], optional
>>
>> Illuminant used to calculate the color coordinates of the spectral
>>> reflectance samples.
>
> **jab_ranges**
>
>> None or ndarray, optional
>>
>> Specifies the pixelization of color space.
>>> (ndarray.shape = (3,3), with first axis: J,a,b, and second
>>>> axis: min, max, delta)
>
> **jab_deltas**
>
>> float or ndarray, optional
>>
>> Specifies the sampling range.
>>
>> A float uses jab_deltas as the maximum Euclidean distance to select
>>
>> samples around each pixel center. A ndarray of 3 deltas, uses
>>
>> a city block sampling around each pixel center.
>
> **cspace**
>
>> _VF_CSPACE or dict, optional
>>
>> Specifies color space. See _VF_CSPACE_EXAMPLE for example structure.
>
> **cieobs**
>
>> _VF_CIEOBS or str, optional
>>
>> Specifies CMF set used to calculate color coordinates.
>
> **ax**
>
>> default ndarray or user defined ndarray, optional
>>
>> default = np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR)
>
> **bx**
>
>> default ndarray or user defined ndarray, optional
>>
>> default = np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR)
>
> **jx**
>
>> None, optional
>>
>> Note that not-None :jab_ranges: override :ax:, :bx: and :jx input.

> **limit_grid_radius**
>
>> 0, optional
>>
>> A value of zeros keeps grid as specified by axr,bxr.
>>
>> A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

> Returns:
>
>> **returns**
>>
>>> rflsampled, jabp
>>>
>>> ndarrays with resp. the subsampled set of spectral reflectance
>>> functions and the pixel coordinate centers.

luxpy.color.cri.VFPX.**plot_VF_PX_models**(*dataVF=None, dataPX=None, plot_VF=True, plot_PX=True, axtype='polar', ax='new', plot_circle_field=True, plot_sample_shifts=False, plot_samples_shifts_at_pixel_center=False, jabp_sampled=None, plot_VF_colors=['g'], plot_PX_colors=['r'], hbin_cmap=None, bin_labels=None, plot_bin_colors=True, force_CVG_layout=False*)

Plot the VF and PX model color shift vectors.

> Args:
>
>> **dataVF**
>>
>>> None or list[dict] with VF_colorshift_model() output, optional
>>>
>>> None plots nothing related to VF model.
>>>
>>> Each list element refers to a different test SPD.
>>
>> **dataPX**
>>
>>> None or list[dict] with PX_colorshift_model() output, optional
>>>
>>> None plots nothing related to PX model.
>>>
>>> Each list element refers to a different test SPD.
>>
>> **plot_VF**
>>
>>> True, optional
>>>
>>> Plot VF model (if :dataVF: is not None).
>>
>> **plot_PX**
>>
>>> True, optional
>>>
>>> Plot PX model (if :dataPX: is not None).
>>
>> **axtype**
>>
>>> 'polar' or 'cart', optional
>>>
>>> Make polar or Cartesian plot.
>>
>> **ax**
>>
>>> None or 'new' or 'same', optional
>>>
>>>> - None or 'new' creates new plot
>>>> - 'same': continue plot on same axes.
>>>> - axes handle: plot on specified axes.
>>
>> **plot_circle_field**
>>
>>> True or False, optional
>>>
>>> Plot lines showing how a series of circles of color coordinates is
>>> distorted by the test SPD.
>>>
>>> The width (wider means more) and color (red means more) of the

lines specify the intensity of the hue part of the color shift.

plot_sample_shifts

> False or True, optional
> Plots the shifts of the individual samples of the rfl-set used to
> calculated the VF model.

plot_samples_shifts_at_pixel_center

> False, optional
> Offers the possibility of shifting the vector shifts of subsampled
> sets from the reference illuminant positions to the pixel centers.
> Note that the pixel centers must be supplied in :jabp_sampled:.

jabp_sampled

> None, ndarray, optional
> Corresponding pixel center for each sample in a subsampled set.

plot_VF_colors

> ['g'] or list[str], optional
> Specifies the plot color the color shift vectors of the VF model.
> If len(:plot_VF_colors:) == 1: same color for each list element
> of :dataVF:.

plot_VF_colors

> ['g'] or list[str], optional
> Specifies the plot color the color shift vectors of the VF model.
> If len(:plot_VF_colors:) == 1: same color for each list element
> of :dataVF:.

hbin_cmap

> None or colormap, optional
> Color map with RGB entries for each of the hue bins specified by
> the hues in _VF_PCOLORSHIFT.
> If None: cmap will be obtained on first run by
> luxpy.cri.plot_shift_data() and returned for use in other functions

plot_bin_colors

> True, optional
> Colorize hue-bins.

bin_labels

> None or list[str] or '#', optional
> Plots labels at the bin center hues.
>
>> - None: don't plot.
>> - list[str]: list with str for each bin.
>>> (len(:bin_labels:) = :nhbins:)
>> - '#': plots number.
>> - '_VF_PCOLORSHIFT': uses the labels in _VF_PCOLORSHIFT['labels']
>> - 'pcolorshift': uses the labels in dataVF['modeldata']['pcolorshift']['labels']

force_CVG_layout

> False or True, optional
> True: Force plot of basis of CVG.

Returns:

> **returns**

---

ax (handle to current axes), cmap (hbin_cmap)

4.4.11 XYZ,LAB classes

py

- CDATA.py

namespace luxpy

class luxpy.color.CDATA.**XYZ**(*value=None*, *relative=True*, *cieobs='1931_2'*, *dtype='xyz'*)

ctf(*dtype='Yuv'*, ***kwargs*)
> Convert XYZ tristimulus values to color space coordinates.
> **Args:**
> > **dtype**
> >
> > > _CSPACE or str, optional
> > > Convert to this color space.
> >
> > **kwargs**
> >
> > > additional input arguments required for
> > > color space transformation.
> > > See specific luxpy function for more info
> > > > (e.g. ?luxpy.xyz_to_lab)
> **Returns:**
> > **returns**
> >
> > > luxpy.LAB with .value field that is a ndarray
> > > > with color space coordinates

plot(*ax=None*, *title=None*, ***kwargs*)
> Plot tristimulus or cone fundamental values.
> **Args:**
> > **ax**
> >
> > > None or axes handles, optional
> > > None: create new figure axes, else use :ax: for plotting.
> >
> > **title**
> >
> > > None or str, optional
> > > Give plot a title.
> >
> > **kwargs**
> >
> > > additional arguments for use with
> > > matplotlib.pyplot.scatter
> **Returns:**
> > **gca**
> >
> > > handle to current axes.

to_Yxy()
> Convert XYZ tristimulus values CIE Yxy chromaticity values.
> **Returns:**
> > **Yxy**
> >
> > > luxpy.LAB with .value field that is a ndarray
> > > with Yxy chromaticity values.

(Y value refers to luminance or luminance factor)

to_Yuv(***kwargs*)

Convert XYZ tristimulus values CIE 1976 Yu'v' chromaticity values.

Returns:

> **Yuv**
>
> > luxpy.LAB with .value field that is a ndarray
> > with CIE 1976 Yu'v' chromaticity values.
> > (Y value refers to luminance or luminance factor)

to_Yuv76(***kwargs*)

Convert XYZ tristimulus values CIE 1976 Yu'v' chromaticity values.

Returns:

> **Yuv**
>
> > luxpy.LAB with .value field that is a ndarray
> > with CIE 1976 Yu'v' chromaticity values.
> > (Y value refers to luminance or luminance factor)

to_Yuv60(***kwargs*)

Convert XYZ tristimulus values CIE 1960 Yuv chromaticity values.

Returns:

> **Yuv**
>
> > luxpy.LAB with .value field that is a ndarray
> > with CIE 1960 Yuv chromaticity values.
> > (Y value refers to luminance or luminance factor)

to_wuv(*xyzw=array([1.0000e+02, 1.0000e+02, 1.0000e+02])*)

Convert XYZ tristimulus values CIE 1964 U*V*W* color space.

Args:

> **xyzw**
>
> > ndarray with tristimulus values of white point, optional
> > Defaults to luxpy._COLORTF_DEFAULT_WHITE_POINT

Returns:

> **wuv**
>
> > luxpy.LAB with .value field that is a ndarray
> > with W*U*V* values.

to_lms()

Convert XYZ tristimulus values or LMS cone fundamental responses to LMS cone fundamental responses.

Returns:

> **lms**
>
> > luxpy.XYZ with .value field that is a ndarray
> > with LMS cone fundamental responses.

to_xyz()

Convert XYZ tristimulus values or LMS cone fundamental responses to XYZ tristimulus values.

Returns:

> **xyz**
>
> > luxpy.XYZ with .value field that is a ndarray
> > with XYZ tristimulus values.

`to_lab`(*xyzw=None*, *cieobs='1931_2'*)

> Convert XYZ tristimulus values to CIE 1976 L*a*b* (CIELAB) coordinates.
>
> **Args:**
>> **xyzw**
>>
>>> None or ndarray with xyz values of white point, optional
>>> None defaults to xyz of CIE D65 using the :cieobs: observer.
>>
>> **cieobs**
>>
>>> luxpy._CIEOBS, optional
>>> CMF set to use when calculating xyzw.
>
> **Returns:**
>> **lab**
>>
>>> luxpy.LAB with .value field that is a ndarray
>>> with CIE 1976 L*a*b* (CIELAB) color coordinates

`to_luv`(*xyzw=None*, *cieobs='1931_2'*)

> Convert XYZ tristimulus values to CIE 1976 L*u*v* (CIELUV) coordinates.
>
> **Args:**
>> **xyzw**
>>
>>> None or ndarray with xyz values of white point, optional
>>> None defaults to xyz of CIE D65 using the :cieobs: observer.
>>
>> **cieobs**
>>
>>> luxpy._CIEOBS, optional
>>> CMF set to use when calculating xyzw.
>
> **Returns:**
>> **luv**
>>
>>> luxpy.LAB with .value field that is a ndarray
>>> with CIE 1976 L*u*v* (CIELUV) color coordinates

`to_Vrb_mb`(*cieobs='1931_2'*, *scaling=[1, 1]*, *M=None*)

> Convert XYZ tristimulus values to V,r,b (Macleod-Boynton) coordinates.
>
> Macleod Boynton: V = R+G, r = R/V, b = B/V
> Note that R,G,B ~ L,M,S
>
> **Args:**
>> **cieobs**
>>
>>> luxpy._CIEOBS, optional
>>> CMF set to use when calculating xyzw.
>>
>> **scaling**
>>
>>> list of scaling factors for r and b dimensions.
>>
>> **M**
>>
>>> None, optional
>>> Conversion matrix for going from XYZ to RGB (LMS)
>>> If None, :cieobs: determines the M (function does inversion)
>
> **Returns:**
>> **Vrb**
>>
>>> luxpy.LAB with .value field that is a ndarray

ndarray with V,r,b (Macleod-Boynton) color coordinates

to_ipt (*cieobs='1931_2'*, *xyzw=None*, *M=None*)
   Convert XYZ tristimulus values to IPT color coordinates.

   I: Lightness axis, P, red-green axis, T: yellow-blue axis.

   Args:
      xyzw

         None or ndarray with xyz values of white point, optional
         None defaults to xyz of CIE D65 using the :cieobs: observer.
      cieobs

         luxpy._CIEOBS, optional
         CMF set to use when calculating xyzw for rescaling Mxyz2lms
         (only when not None).
      M

         None, optional
         None defaults to conversion matrix determined by :cieobs:
   Returns:
      ipt

         luxpy.LAB with *.value* field that is a ndarray
         with IPT color coordinates
   Note:
      xyz is assumed to be under D65 viewing conditions!! | If necessary perform chromatic
         adaptation !!

to_Ydlep (*cieobs='1931_2'*, *xyzw=array([1.0000e+02, 1.0000e+02, 1.0000e+02])*)
   Convert XYZ values to Y, dominant (complementary) wavelength and excitation purity.
   Args:
      xyzw

         None or ndarray with xyz values of white point, optional
         None defaults to xyz of CIE D65 using the :cieobs: observer.
      cieobs

         luxpy._CIEOBS, optional
         CMF set to use when calculating spectrum locus coordinates.
   Returns:
      Ydlep

         ndarray with Y, dominant (complementary) wavelength
         and excitation purity

to_srgb (*gamma=2.4*)
   Calculates IEC:61966 sRGB values from xyz.
   Args:
      xyz

         ndarray with relative tristimulus values.
      gamma

         2.4, optional
         compression in sRGB

Returns:

rgb

ndarray with R,G,B values (uint8).

to_jabz (*ztype='jabz'*)

Convert XYZ tristimulus values to Jz,az,bz color coordinates.

Args:

xyz

ndarray with absolute tristimulus values (Y in cd/m^2!)

ztype

'jabz', optional

String with requested return:

Options: 'jabz', 'iabz'

Returns:

jabz

ndarray with Jz,az,bz color coordinates

Notes:

1. :xyz: is assumed to be under D65 viewing conditions! If necessary perform chromatic adaptation!

2a. Jz represents the 'lightness' relative to a D65 white with luminance = 10000 cd/m^2

(note that Jz that not exactly equal 1 for this high value, but rather for 102900 cd/m2)

2b. az, bz represent respectively a red-green and a yellow-blue opponent axis

(but note that a D65 shows a small offset from (0,0))

Reference: 1. Safdar, M., Cui, G., Kim,Y. J., and Luo,M. R. (2017). Perceptually uniform color space for image signals including high dynamic range and wide gamut. Opt. Express, vol. 25, no. 13, pp. 15131–15151, Jun. 2017.

to_jabM_ciecam02 (*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, yellowbluepurplecorrect=None, mcat='cat02'*)

See ?luxpy.xyz_to_jabM_ciecam02

to_jabC_ciecam02 (*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, yellowbluepurplecorrect=None, mcat='cat02'*)

See ?luxpy.xyz_to_jabC_ciecam02

to_jab_cam02ucs (*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, yellowbluepurplecorrect=None, mcat='cat02'*)

See ?luxpy.xyz_to_jab_cam02ucs

to_jab_cam02lcd (*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, yellowbluepurplecorrect=None, mcat='cat02'*)

See ?luxpy.xyz_to_jab_cam02lcd

to_jab_cam02scd(*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, yellowbluepurplecorrect=None, mcat='cat02'*)
 See ?luxpy.xyz_to_jab_cam02scd

to_jabM_ciecam16(*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, mcat='cat16'*)
 See ?luxpy.xyz_to_jabM_ciecam16

to_jabC_ciecam16(*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, mcat='cat16'*)
 See ?luxpy.xyz_to_jabC_ciecam16

to_jab_cam16ucs(*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, mcat='cat16'*)
 See ?luxpy.xyz_to_jab_cam02ucs

to_jab_cam16lcd(*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, mcat='cat16'*)
 See ?luxpy.xyz_to_jab_cam16lcd

to_jab_cam16scd(*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), Yw=100.0, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, naka_rushton_parameters=None, unique_hue_data=None, mcat='cat16'*)
 See ?luxpy.xyz_to_jab_cam16scd

to_jabM_zcam(*xyzw=None, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, mcat='cat16'*)
 See ?luxpy.xyz_to_jabM_zcam

to_jabC_zcam(*xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]]), conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, mcat='cat16'*)
 See ?luxpy.xyz_to_jabC_zcam

to_qabW_cam15u(*fov=10.0, parameters=None*)
 See ?luxpy.xyz_to_qabW_cam15u

to_lab_cam_sww_2016(*xyzw=None, Yb=20.0, Lw=400.0, relative=True, parameters=None, inputtype='xyz', cieobs='2006_10'*)
 See ?luxpy.xyz_to_lab_cam_sww_2016

to_qabS_cam18sl(*xyz, xyzb=None, Lb=[100], fov=10.0, parameters=None*)
 See ?luxpy.xyz_to_qabS_cam18sl

to_qabM_cam18sl(*xyz, xyzb=None, Lb=[100], fov=10.0, parameters=None*)
 See ?luxpy.xyz_to_qabM_cam18sl

class luxpy.color.CDATA.**LAB**(*value=None, relative=True, cieobs='1931_2', dtype='lab', xyzw=None, M=None, scaling=None, Lw=None, Yw=None, Yb=None, conditions=None, naka_rushton_parameters=None, unique_hue_data=None, yellowbluepurplecorrect=None, mcat=None, ucstype=None, fov=None, parameters=None*)

 ctf(***kwargs*)
  Convert color space coordinates to XYZ tristimulus values.
  Args:
    dtype

> 'xyz'
>> Convert to this color space.
>
> **kwargs**
>
>> additional input arguments required for
>> color space transformation.
>> See specific luxpy function for more info
>>> (e.g. ?luxpy.xyz_to_lab)
>
> **Returns:**
>> **returns**
>>
>>> luxpy.XYZ with .value field that is a ndarray
>>> with tristimulus values

plot (*plt_type='3d'*, *ax=None*, *title=None*, ***kwargs*)
>   Plot color coordinates.
>   **Args:**
>> **plt_type**
>>
>>> '3d' or 3 or '2d or 2, optional
>>>> -'3d' or 3: plot all 3 dimensions (lightness and chromaticity)
>>>> -'2d' or 2: plot only chromaticity dimensions.
>>
>> **ax**
>>
>>> None or axes handles, optional
>>> None: create new figure axes, else use :ax: for plotting.
>>
>> **title**
>>
>>> None or str, optional
>>> Give plot a title.
>>
>> **kwargs**
>>
>>> additional arguments for use with
>>> matplotlib.pyplot.scatter
>
>   **Returns:**
>> **gca**
>>
>>> handle to current axes.

to_xyz (***kwargs*)
>   Convert color space coordinates to XYZ tristimulus values.

4.5 Toolboxes

4.5.1 photbiochem/

> **py**
>
>> • __init__.py
>>
>> • cie_tn003_2015.py
>>
>> • ASNZS_1680_2_5_1997_COI.py
>>
>> • circadian_CS_CLa_lrc.py
>
> **namespace** luxpy.photbiochem

Module for calculating CIE (S026:2018 & TN003:2015) photobiological quantities

(Eelc, Eemc, Eesc, Eer, Eez, and Elc, Emc, Esc, Er, Ez)

Photore-ceptor	Photopigment (la-bel, α)	Spectral efficiency s$\alpha(\lambda)$	Quantity (α-opic irra-diance)	Q-symbol (Ee,α)	Unit symbol
l-cone	photopsin (lc)	erythrolabe	erythropic	Ee,lc	W.m2
m-cone	photopsin (mc)	chlorolabe	chloropic	Ee,mc	W.m2
s-cone	photopsin (sc)	cyanolabe	cyanopic	Ee,sc	W.m2
rod	rhodopsin (r)	rhodopic	rhodopic	Ee,r	W.m2
ipRGC	melanopsin (z)	melanopic	melanopic	Ee,z	W.m2

CIE recommends that the α-opic irradiance is determined by convolving the spectral irradiance, Ee,$\lambda(\lambda)$ (Wm2), for each wavelength, with the action spectrum, s$\alpha(\lambda)$, where s$\alpha(\lambda)$ is normalized to one at its peak:

> Ee,α = Ee,$\lambda(\lambda)$ s$\alpha(\lambda)$ dλ

where the corresponding units are Wm2 in each case.

The equivalent luminance is calculated as:

> E,α = Km Ee,$\lambda(\lambda)$ s$\alpha(\lambda)$ dλ V(λ) dλ / s$\alpha(\lambda)$ dλ

To avoid ambiguity, the weighting function used must be stated, so, for example, cyanopic refers to the cyanopic irradiance weighted using the s-cone or ssc(λ) spectral efficiency function.

> **_PHOTORECEPTORS** ['l-cone', 'm-cone','s-cone', 'rod', 'iprgc']
>
> **_Ee_SYMBOLS** ['Ee,lc','Ee,mc', 'Ee,sc','Ee,r', 'Ee,z']
>
> **_E_SYMBOLS** ['E,lc','E,mc', 'E,sc','E,r', 'E,z']
>
> **_Q_SYMBOLS** ['Q,lc','Q,mc', 'Q,sc','Q,r', 'Q,z']
>
> **_Ee_UNITS** ['Wm2'] * 5
>
> **_E_UNITS** ['lux'] * 5
>
> **_Q_UNITS** ['photons/m2/s'] * 5
>
> **_QUANTITIES**
>
>> list with actinic types of irradiance, illuminance
>> ['erythropic',
>>> 'chloropic',
>>> 'cyanopic',
>>> 'rhodopic',
>>> 'melanopic']
>
> **_ACTIONSPECTRA**
>
>> ndarray with default CIE-S026:2018 alpha-actinic action spectra. (stored in file:

'./data/cie_S026_2018_SI_action_spectra_CIEToolBox_v1.049.dat')

_ACTIONSPECTRA_CIES026

> ndarray with alpha-actinic action spectra. (stored in file:
> './data/cie_S026_2018_SI_action_spectra_CIEToolBox_v1.049.dat')

_ACTIONSPECTRA_CIETN003

> ndarray with CIE-TN003:2015 alpha-actinic action spectra. (stored in file:
> './data/cie_tn003_2015_SI_action_spectra.dat')

spd_to_aopicE()

> Calculate alpha-opic irradiance (Ee,α) and equivalent
> luminance (Eα) values for the l-cone, m-cone, s-cone,
> rod and iprgc (α) photoreceptor cells following
> CIE S026:2018 (= default actionspectra) or CIE TN003:2015.

spd_to_aopicEDI()

> Calculate alpha-opic equivalent daylight (D65) illuminance (lx)
> for the l-cone, m-cone, s-cone, rod and iprgc (α) photoreceptor cells.

spd_to_aopicDER()

> Calculate α-opic Daylight (D65) Efficacy Ratio
> for the l-cone, m-cone, s-cone, rod and iprgc (α) photoreceptor cells.

spd_to_aopicELR()

> Calculate α-opic Efficacy of Luminous Radiation
> for the l-cone, m-cone, s-cone, rod and iprgc (α) photoreceptor cells.

References: 1. CIE-S026:E2018 (2018). CIE System for Metrology of Optical Radiation for ipRGC-Influenced Responses to Light (Vienna, Austria). (https://files.cie.co.at/CIE%20S%20026%20alpha-opic%20Toolbox%20User%20Guide.pdf)

2. CIE-TN003:2015 (2015). Report on the first international workshop on circadian and neurophysiological photometry, 2013 (Vienna, Austria). (http://files.cie.co.at/785_CIE_TN_003-2015.pdf)

Module for calculation of cyanosis index (AS/NZS 1680.2.5:1997)

> **_COI_OBS** Default CMF set for calculations
>
> **_COI_CSPACE** Default color space (CIELAB)
>
> **_COI_RFL_BLOOD** ndarray with reflectance spectra of 100% and 50% oxygenated blood
>
> **spd_to_COI_ASNZS1680** Calculate the Cyanosis Observartion Index (COI) [ASNZS 1680.2.5-1995]

Reference: AS/NZS1680.2.5 (1997). INTERIOR LIGHTING PART 2.5: HOSPITAL AND MEDICAL TASKS.

Module for Blue light hazard calculations

> **_BLH** Blue Light Hazard function
>
> **spd_to_blh_eff()** Calculate Blue Light Hazard efficacy (K) or efficiency (eta) of radiation.

References:

1. IEC 62471:2006, 2006, Photobiological safety of lamps and lamp systems.

2. IEC TR 62778, 2014, Application of IEC 62471 for the assessment of blue light hazard to light sources and luminaires.

luxpy.toolboxes.photbiochem.**spd_to_aopicE**(*sid*, *Ee=None*, *E=None*, *Q=None*, *cieobs='1931_2'*, *sid_units='W/m2'*, *out='Eeas'*, *actionspectra='CIE-S026'*)

Calculate alpha-opic irradiance (Ee,α) values (W/m^2) for the l-cone, m-cone, s-cone, rod and iprgc (α) photoreceptor cells following CIE S026:2018.

Args:

> **sid**
>
> > numpy.ndarray with retinal spectral irradiance in :sid_units:
> >
> > (if 'uW/cm2', sid will be converted to SI units 'W/m2')
>
> **Ee**
>
> > None, optional
> >
> > If not None: normalize :sid: to an irradiance of :Ee:
>
> **E**
>
> > None, optional
> >
> > If not None: normalize :sid: to an illuminance of :E:
> >
> > Note that E is calculate using a Km factor corrected to standard air.
>
> **Q**
>
> > None, optional
> >
> > If not None: Normalize :sid: to a quantal energy of :Q:
>
> **cieobs**
>
> > _CIEOBS or str, optional
> >
> > Type of cmf set to use for photometric units.
>
> **sid_units**
>
> > 'W/m2', optional
> >
> > Other option 'uW/m2', input units of :sid:
>
> **out**
>
> > 'Eeas' or str, optional
> >
> > Determines values to return.
> >
> > (to get also get equivalent illuminance Eα set :out: to 'Eeas,Eas')
>
> **actionspectra**
>
> > 'CIES026', optional
> >
> > Actionspectra to use in calculation
> >
> > options:
> >
> > - 'CIE-S026': will use action spectra as defined in CIE S026
> >
> > - 'CIE-TN003': will use action spectra as defined in CIE TN003

Returns:

> **returns**
>
> > Eeas a numpy.ndarray with the α-opic irradiance
> >
> > of all spectra in :sid: in SI-units (W/m^2).
> >
> > (other choice can be set using :out:)

References: 1. CIE-S026:E2018 (2018). CIE System for Metrology of Optical Radiation for iPRGC-Influenced Responses to Light (Vienna, Austria). (https://files.cie.co.at/CIE%20S%20026%

20alpha-opic%20Toolbox%20User%20Guide.pdf)

2. CIE-TN003:2015 (2015). Report on the first international workshop on circadian and neurophysiological photometry, 2013 (Vienna, Austria). (http://files.cie.co.at/785_CIE_TN_003-2015.pdf)

luxpy.toolboxes.photbiochem.**spd_to_aopicEDI**(*sid*, *Ee=None*, *E=None*, *Q=None*, *cieobs='1931_2'*, *sid_units='W/m2'*, *actionspectra='CIE-S026'*, *ref='D65'*, *out='a_edi'*)

Calculate alpha-opic equivalent daylight (D65) illuminance (lux) for the l-cone, m-cone, s-cone, rod and iprgc (α) photoreceptor cells.

Args:

> **sid**
>
>> numpy.ndarray with retinal spectral irradiance in :sid_units:
>> (if 'uW/cm2', sid will be converted to SI units 'W/m2')
>
> **Ee**
>
>> None, optional
>> If not None: normalize :sid: to an irradiance of :Ee:
>
> **E**
>
>> None, optional
>> If not None: normalize :sid: to an illuminance of :E:
>> Note that E is calculate using a Km factor corrected to standard air.
>
> **Q**
>
>> None, optional
>> If not None: nNormalize :sid: to a quantal energy of :Q:
>
> **cieobs**
>
>> _CIEOBS or str, optional
>> Type of cmf set to use for photometric units.
>
> **sid_units**
>
>> 'W/m2', optional
>> Other option 'uW/m2', input units of :sid:
>
> **actionspectra**
>
>> 'CIES026', optional
>> Actionspectra to use in calculation
>> options:
>> - 'CIE-S026': will use action spectra as defined in CIE S026
>> - 'CIE-TN003': will use action spectra as defined in CIE TN003
>
> **ref**
>
>> 'D65', optional
>> Reference (daylight) spectrum to use. ('D65' or 'E' or ndarray)
>
> **out**
>
>> 'Eeas, Eas' or str, optional
>> Determines values to return.

Returns:

> **returns**
>
>> ndarray with the α-opic Equivalent Daylight Illuminance (lux) with the
>> for the l-cone, m-cone, s-cone, rod and iprgc photoreceptors
>> of all spectra in :sid: in SI-units.

`luxpy.toolboxes.photbiochem.`**`spd_to_aopicDER`**(*sid*, *cieobs='1931_2'*, *sid_units='W/m2'*, *actionspectra='CIE-S026'*, *ref='D65'*)

Calculate α-opic Daylight (D65) Efficacy Ratio (= α-opic Daylight (D65) Efficiency) for the l-cone, m-cone, s-cone, rod and iprgc (α) photoreceptor cells.

> **Args:**
>> **sid**
>>> numpy.ndarray with retinal spectral irradiance in :sid_units:
>>> (if 'uW/cm2', sid will be converted to SI units 'W/m2')
>>
>> **cieobs**
>>> _CIEOBS or str, optional
>>> Type of cmf set to use for photometric units.
>>
>> **sid_units**
>>> 'W/m2', optional
>>> Other option 'uW/m2', input units of :sid:
>>
>> **actionspectra**
>>> 'CIES026', optional
>>> Actionspectra to use in calculation
>>> options:
>>> - 'CIE-S026': will use action spectra as defined in CIE S026
>>> - 'CIE-TN003': will use action spectra as defined in CIE TN003
>>
>> **ref**
>>> 'D65', optional
>>> Reference (daylight) spectrum to use. ('D65' or 'E' or ndarray)
>
> **Returns:**
>> **returns**
>>> ndarray with the α-opic Daylight Efficacy Ratio with the
>>> for the l-cone, m-cone, s-cone, rod and iprgc photoreceptors
>>> of all spectra in :sid: in SI-units.

`luxpy.toolboxes.photbiochem.`**`spd_to_aopicELR`**(*sid*, *cieobs='1931_2'*, *sid_units='W/m2'*, *actionspectra='CIE-S026'*, *ref='D65'*)

Calculate α-opic Efficacy of Luminous Radiation (W/lm) for the l-cone, m-cone, s-cone, rod and iprgc (α) photoreceptor cells.

> **Args:**
>> **sid**
>>> numpy.ndarray with retinal spectral irradiance in :sid_units:
>>> (if 'uW/cm2', sid will be converted to SI units 'W/m2')
>>
>> **cieobs**
>>> _CIEOBS or str, optional
>>> Type of cmf set to use for photometric units.
>>
>> **sid_units**
>>> 'W/m2', optional
>>> Other option 'uW/m2', input units of :sid:
>>
>> **actionspectra**
>>> 'CIES026', optional
>>> Actionspectra to use in calculation
>>> options:

- 'CIE-S026': will use action spectra as defined in CIE S026
- 'CIE-TN003': will use action spectra as defined in CIE TN003

ref

'D65', optional

Reference (daylight) spectrum to use. ('D65' or 'E' or ndarray)

Returns:

returns

ndarray with the α-opic Efficacy of Luminous Radiation (W/lm) with the

for the l-cone, m-cone, s-cone, rod and iprgc photoreceptors

of all spectra in :sid: in SI-units.

luxpy.toolboxes.photbiochem.**spd_to_COI_ASNZS1680**(*S=None*, *tf='lab'*, *cieobs='1931_2'*, *out='COI*, *cct'*, *extrapolate_rfl=False*)

Calculate the Cyanosis Observation Index (COI) [ASNZS 1680.2.5-1995].

Args:

S

ndarray with light source spectrum (first column are wavelengths).

tf

_COI_CSPACE, optional

Color space in which to calculate the COI.

Default is CIELAB.

cieobs

_COI_CIEOBS, optional

CMF set to use.

Default is '1931_2'.

out

'COI,cct' or str, optional

Determines output.

extrapolate_rfl

False, optional

If False:

limit the wavelength range of the source to that of the standard

reflectance spectra for the 50% and 100% oxygenated blood.

Returns:

COI

ndarray with cyanosis indices for input sources.

cct

ndarray with correlated color temperatures.

Note: Clause 7.2 of the ASNZS 1680.2.5-1995. standard mentions the properties demanded of the light source used in region where visual conditions suitable to the detection of cyanosis should be provided:

1. The correlated color temperature (CCT) of the source should be from 3300 to 5300 K.
2. The cyanosis observation index should not exceed 3.3

luxpy.toolboxes.photbiochem.**spd_to_CS_CLa_lrc**(*El=None*, *version='CLa2.0'*, *E=None*, *sum_sources=False*, *interpolate_sources=True*, *t_CS=1.0*, *f_CS=1.0*)

Calculate Circadian Stimulus (CS) and Circadian Light (CLa, CLa2.0).

Args:

El

>   ndarray, optional

>   Defaults to D65

>   light source spectral irradiance distribution

version

>   'CLa2.0', optional

>   CLa version to calculate

>   Options:

>   - 'CLa1.0': Rea et al. 2012

>   - 'CLa2.0': Rea et al. 2021, 2022

E

>   None, float or ndarray, optional

>   Illuminance of light sources.

>   If None: El is used as is, otherwise El is renormalized to have

>   an illuminance equal to E.

sum_sources

>   False, optional

>>   - False: calculate CS (1.0,2.0) and CLa (1.0, 2.0) for all sources in El array.

>>   - True: sum sources in El to a single source and perform calc.

interpolate_sources

>   True, optional

>>   - True: El is interpolated to wavelength range of efficiency

>>>   functions (as in LRC calculator).

>>   - False: interpolate efficiency functions to source range.

>>>   Source interpolation is not recommended due to possible

>>>   errors for peaky spectra.

>>>   (see CIE15-2018, "Colorimetry").

t_CS

>   1.0, optional

>   The duration factor (in hours): a continuous value from 0.5 to 3.0

f_CS

>   1.0, optional

>   The spatial distribution factor: a discrete value (2, 1, or 0.5)

>   depending upon the spatial distribution of the light source.

>   Default = 1 (for t = 1 h, CS is equal to the 2012 version).

>   Options:

>   - 2.0: full visual field, as with a Ganzfeld.

>   - 1.0: central visual field, as with a discrete light box on a desk.

>   - 0.5: superior visual field, as from ceiling mounted down-light fixtures.

Returns:

>   **CS**

>>   ndarray with Circadian stimulus values

>   **CLa**

>>   ndarray with Circadian Light values

Notes on CLa1.0 (2012 version): 1. The original 2012 (E.q. 1) had set the peak wavelength of the melanopsin at 480 nm. Rea et al. later published a corrigendum with updated model parameters for k, a_{b-y} and

a_rod. The comparison table between showing values calculated for a number of sources with the old and updated parameters were very close (~1 unit voor CLa).

2. In that corrrection paper they did not mention a change in the factor (1622) that multiplies the (sum of) the integral(s) in Eq. 1. HOWEVER, the excel calculator released in 2017 and the online calculator show that factor to have a value of 1547.9. The change in values due to the new factor is much larger than their the updated mentioned in note 1!

3. For reasons of consistency the calculator uses the latest model parameters, as could be read from the excel calculator. They values adopted are: multiplier 1547.9, k = 0.2616, a_{b-y} = 0.7 and a_rod = 3.3.

4. The parameter values to convert CLa to CS were also taken from the 2017 excel calculator.

Notes on CLa2.0 (2021 version): 1. In the original model, 1000 lux of CIE Illuminant A resulted in a CLA = 1000. In the revised model, a photopic illuminance of 1000 lux from CIE Illuminant A (approximately that of an incandescent lamp operated at 2856 K) results in a CLA 2.0 = 813. The value of 813 CLA 2.0 should be used by those wishing to calibrate instrumentation designed to report CLA 2.0 and CS. CLA 2.0 values can still be used to approximate the photopic illuminance, in lux, from a nonspecific "white" light source. For comparison, CLA 2.0 values should be multiplied by 1.23 to estimate the equivalent photopic illuminance from CIE Illuminant A, or by 0.66 to estimate the equivalent photopic illuminance from CIE Illuminant D65 (an approximation of daylight with a CCT of 6500 K).

2. Nov. 6, 2021: To get a value of CLa2.0 = 813, Eq. 3 from the paper must be adjusted to also divide by the transmision of the macula ('mp' in paper) the S-cone and Vlambda functions prior to calculating the integrals in the denominators of the first factor after the a_rod_1 and a_rod_2 scalars! Failure to do so results in a CLa2.0 of 800, instead of the reported 813 by the online calculator. Verification of the code on github shows indeed that these denominators are calculated by using the macular transmission divided S-cone and Vlambda functions. Is this an error in the code or in the paper?

3. Feb. 22, 2022: A corrigendum has been released for Eq. 3 in the original paper, where the normalization is indeed done.

4. Feb. 22, 2022: While the rodsat value in the corrigendum is defined as 6.50 W/m^2, this calculator uses the value as used in the online calculator: 6.5215 W/m^2. (see code base on github:)

References:

1. LRC Online Circadian stimulus calculator

2. LRC Excel based Circadian stimulus calculator.

3. Rea MS, Figueiro MG, Bierman A, and Hamner R (2012). Modelling the spectral sensitivity of the human circadian system. Light. Res. Technol. 44, 386–396.

4. Rea MS, Figueiro MG, Bierman A, and Hamner R (2012). Erratum: Modeling the spectral sensitivity of the human circadian system (Lighting Research and Technology (2012) 44:4 (386-396)). Light. Res. Technol. 44, 516.

5. Rea, M. S., Nagare, R., & Figueiro, M. G. (2021). Modeling Circadian Phototransduction: Quantitative Predictions of Psychophysical Data. Frontiers in Neuroscience, 15, 44.

6. Rea, M. S., Nagare, R., & Figueiro, M. G. (2022). Corrigendum: Modeling Circadian Photo-transduction: Quantitative Predictions of Psychophysical Data. Frontiers in Neuroscience, 16.

7. LRC Online Circadian stimulus calculator (CLa2.0, 2021)

8. Github code: LRC Online Circadian stimulus calculator (CLa2.0, accessed Nov. 5, 2021)

`luxpy.toolboxes.photbiochem.`**`CLa_to_CS`**(*CLa*, *t=1*, *f=1*, *forward=True*)

Convert Circadian Light to Circadian Stimulus (and back).

Args:

> **CLa**
>
> > ndarray with Circadian Light values

or Circadian Stimulus values (if forward == False)

t

1.0, optional

The duration factor (in hours): a continuous value from 0.5 to 3.0

f

1.0, optional

The spatial distribution factor: a discrete value (2, 1, or 0.5)

depending upon the spatial distribution of the light source.

Default = 1 (for t = 1 h, CS is equal to the 2012 version).

Options:

- 2.0: full visual field, as with a Ganzfeld.

- 1.0: central visual field, as with a discrete light box on a desk.

- 0.5: superior visual field, as from ceiling mounted down-light fixtures.

forward

True, optional

If True: convert CLa to CS values.

If False: convert CS values to CLa values.

Returns:

CS

ndarray with CS values or with CLa values (if forward == False)

References: 1. Rea MS, Figueiro MG, Bierman A, and Hamner R (2012). Modelling the spectral sensitivity of the human circadian system. Light. Res. Technol. 44, 386–396.

2. Rea MS, Figueiro MG, Bierman A, and Hamner R (2012). Erratum: Modeling the spectral sensitivity of the human circadian system (Lighting Research and Technology (2012) 44:4 (386-396)). Light. Res. Technol. 44, 516.

3. Rea, M. S., Nagare, R., & Figueiro, M.G. (2021). Modeling Circadian Phototransduction: Quantitative Predictions of Psychophysical Data. Frontiers in Neuroscience, 15, 44.

4. LRC Online Circadian Stimulus calculator (CLa2.0, 2021)

luxpy.toolboxes.photbiochem.**spd_to_blh_eff**(*spd*, *efficacy=True*, *cieobs='1931_2'*, *scr='dict'*, *K=None*)

Calculate Blue Light Hazard efficacy (K) or efficiency (eta) of radiation.

Args:

S

ndarray with spectral data

cieobs

str, optional

Sets the type of Vlambda function to obtain.

scr

'dict' or array, optional

- 'dict': get from ybar from _CMF

- 'array': ndarray in :cieobs:

Determines whether to load cmfs from file (./data/cmfs/)

or from dict defined in .cmf.py

Vlambda is obtained by collecting Ybar.

K

None, optional

e.g. K = 683 lm/W for '1931_2' (relative == False)

or K = 100/sum(spd*dl) (relative == True)

Returns:

> **eff**
>
> > ndarray with blue light hazard efficacy or efficiency of radiation values.

References:
1. IEC 62471:2006, 2006, Photobiological safety of lamps and lamp systems.
2. IEC TR 62778, 2014, Application of IEC 62471 for the assessment of blue light hazard to light sources and luminaires.

4.5.2 indvcmf/

py

- __init__.py
- individual_observer_cmf_model.py

namespace luxpy.indvcmf

Module for Individual Observer lms-CMFs (Asano, 2016 and CIE TC1-97)

_DATA_PATH path to data files

_DATA Dict with required data

_DSRC_STD_DEF default data source for stdev of physiological data ('matlab', 'germany')

_DSRC_LMS_ODENS_DEF default data source for lms absorbances and optical densities ('asano', 'cietc197')

_LMS_TO_XYZ_METHOD default method to calculate lms to xyz conversion matrix ('asano', 'cietc197')

_WL_CRIT critical wavelength above which interpolation of S-cone data fails.

_WL default wavelengths of spectral data in INDVCMF_DATA.

load_database() Load a database with parameters and data required by the Asano model.

init() Initialize: load database required for Asano Individual Observer Model into the default _DATA dict and set some options for rounding, sign. figs and chopping small value to zero; for source data to use for spectral data for LMS absorp. and optical densities, . . .

query_state() print current settings for global variables.

compute_cmfs() Generate Individual Observer CMFs (cone fundamentals) based on CIE2006 cone fundamentals and published literature on observer variability in color matching and in physiological parameters (Use of Asano optical data and model; or of CIE TC1-91 data and 'variability'-extended model possible).

cie2006cmfsEx() Generate Individual Observer CMFs (cone fundamentals) based on CIE2006 cone fundamentals and published literature on observer variability in color matching and in physiological parameters. (Use of Asano optical data and model; or of CIE TC1-91 data and 'variability'-extended model possible)

getMonteCarloParam() Get dict with normally-distributed physiological factors for a population of observers.

getUSCensusAgeDist() Get US Census Age Distribution

genMonteCarloObs() Monte-Carlo generation of individual observer color matching functions (cone fundamentals) for a certain age and field size.

getCatObs() Generate cone fundamentals for categorical observers.

get_lms_to_xyz_matrix() Calculate lms to xyz conversion matrix for a specific field size determined as a weighted combination of the 2° and 10° matrices.

lmsb_to_xyzb() Convert from LMS cone fundamentals to XYZ CMFs using conversion matrix determined as a weighted combination of the 2° and 10° matrices.

add_to_cmf_dict() Add set of cmfs to _CMF dict.

plot_cmfs() Plot cmf set.

References

1. Asano Y, Fairchild MD, and Blondé L (2016). Individual Colorimetric Observer Model. PLoS One 11, 1–19.

2. Asano Y, Fairchild MD, Blondé L, and Morvan P (2016). Color matching experiment for highlighting interobserver variability. Color Res. Appl. 41, 530–539.

3. CIE TC1-36 (2006). Fundamental Chromaticity Diagram with Physiological Axes - Part I (Vienna: CIE).

4. Asano's Individual Colorimetric Observer Model

5. CIE TC1-97 cmf functions python code developed by Ivar Farup and Jan Hendrik Wold.

Notes

1. Port of Matlab code from: https://www.rit.edu/cos/colorscience/re_AsanoObserverFunctions.php (Accessed April 20, 2018) 2. Adjusted/extended following CIE TC1-97 Python code (and data): github.com/ifarup/ciefunctions (Copyright (C) 2012-2017 Ivar Farup and Jan Henrik Wold) (Accessed Dec 18, 2019)

luxpy.toolboxes.indvcmf.**load_database**(*wl=None*, *dsrc_std=None*, *dsrc_lms_odens=None*, *path=None*)
Load database required for Asano Individual Observer Model.

Args:

> **wl**
>
>> None, optional
>>
>> Wavelength range to interpolate data to.
>>
>> None defaults to the wavelength range associated with data in :dsrc_lms_odens:
>
> **path**
>
>> None, optional
>>
>> Path where data files are stored (If None: look in ./data/ folder under toolbox path)
>
> **dsrc_std**
>
>> None, optional
>>
>> Data source ('matlab' code, or 'germany') for stdev data on physiological factors.
>>
>> None defaults to string in _DSRC_STD_DEF
>
> **dsrc_lms_odens**

None, optional

Data source ('asano', 'cietc197') for LMS absorbance and optical density data.

None defaults to string in _DSRC_LMS_ODENS_DEF

Returns:

data

dict with data for:

- 'LMSa': LMS absorbances

- 'rmd': relative macular pigment density

- 'docul': ocular media optical density

- 'USCensus2010population': data (age and numbers) on a 2010 US Census

- 'CatObsPfctr': dict with iteratively derived Categorical Observer physiological stdevs.

- 'M2d': Asano 2° lms to xyz conversion matrix

- 'M10d': Asano 10° lms to xyz conversion matrix

- standard deviations on physiological parameters: 'od_lens', 'od_macula', 'od_L', 'od_M', 'od_S', 'shft_L', 'shft_M', 'shft_S'

luxpy.toolboxes.indvcmf.**init**(*wl=None*, *dsrc_std=None*, *dsrc_lms_odens=None*, *lms_to_xyz_method=None*, *use_sign_figs=True*, *use_my_round=True*, *use_chop=True*, *path=None*, *out=None*, *verbosity=1*)

Initialize: load database required for Asano Individual Observer Model into the default _DATA dict and set some options for rounding, sign. figs and chopping small value to zero; for source data to use for spectral data for LMS absorp. and optical desnities, . . .

Args:

wl

None, optional

Wavelength range to interpolate data to.

None defaults to the wavelength range associated with data in :dsrc_lms_odens:

dsrc_std

None, optional

Data source ('matlab' code, or 'germany') for stdev data on physiological factors.

None defaults to string in _DSRC_STD_DEF

dsrc_lms_odens

None, optional

Data source ('asano', 'cietc197') for LMS absorbance and optical density data.

None defaults to string in _DSRC_LMS_ODENS_DEF

lms_to_xyz_method

None, optional

Method to use to determine lms-to-xyz conversion matrix (options: 'asano', 'cietc197')

use_my_round

True, optional

If True: use my_rounding() conform CIE TC1-91 Python code 'ciefunctions'. (slows down code)

by setting _USE_MY_ROUND.

use_sign_figs

True, optional

>>> If True: use sign_figs() conform CIE TC1-91 Python code 'ciefunctions'. (slows down code)

>>> by setting _USE_SIGN_FIGS.

use_chop

>>> True, optional

>>> If True: use chop() conform CIE TC1-91 Python code 'ciefunctions'. (slows down code)

>>> by setting _USE_CHOP.

path

>>> None, optional

>>> Path where data files are stored (If None: look in ./data/ folder under toolbox path)

out

>>> None, optional

>>> If None: only set global variables, do not output _DATA.copy()

verbosity

>>> 1, optional

>>> Print new state of global settings.

Returns:

data

>>> if out is not None: return a dict with dict with data for:

>>> - 'LMSa': LMS absorbances

>>> - 'rmd': relative macular pigment density

>>> - 'docul': ocular media optical density

>>> - 'USCensus2010population': data (age and numbers) on a 2010 US Census

>>> - 'CatObsPfctr': dict with iteratively derived Categorical Observer physiological stdevs.

>>> - 'M2d': Asano 2° lms to xyz conversion matrix

>>> - 'M10d': Asano 10° lms to xyz conversion matrix

>>> - standard deviations on physiological parameters: 'od_lens', 'od_macula', 'od_L', 'od_M', 'od_S', 'shft_L', 'shft_M', 'shft_S'

luxpy.toolboxes.indvcmf.**query_state**()

>   Print current settings for 'global variables'.

luxpy.toolboxes.indvcmf.**cie2006cmfsEx**(*age=32,   fieldsize=10,   wl=None, var_od_lens=0,   var_od_macula=0,   var_od_L=0, var_od_M=0,   var_od_S=0,   var_shft_L=0, var_shft_M=0,   var_shft_S=0,   norm_type=None, out='lms',   base=False,   strategy_2=True, odata0=None,   lms_to_xyz_method=None, allow_negative_values=False,   normalize_lms_to_xyz_matrix=False*)

>   Generate Individual Observer CMFs (cone fundamentals) based on CIE2006 cone fundamentals and published literature on observer variability in color matching and in physiological parameters.

>   **Args:**

>> **age**

>>> 32 or float or int, optional

>>> Observer age

>> **fieldsize**

> 10, optional
>
> Field size of stimulus in degrees (between 2° and 10°).

wl

> None, optional
>
> Interpolation/extraplation of :LMS: output to specified wavelengths.
>
> None: output original _WL

var_od_lens

> 0, optional
>
> Std Dev. in peak optical density [%] of lens.

var_od_macula

> 0, optional
>
> Std Dev. in peak optical density [%] of macula.

var_od_L

> 0, optional
>
> Std Dev. in peak optical density [%] of L-cone.

var_od_M

> 0, optional
>
> Std Dev. in peak optical density [%] of M-cone.

var_od_S

> 0, optional
>
> Std Dev. in peak optical density [%] of S-cone.

var_shft_L

> 0, optional
>
> Std Dev. in peak wavelength shift [nm] of L-cone.

var_shft_L

> 0, optional
>
> Std Dev. in peak wavelength shift [nm] of M-cone.

var_shft_S

> 0, optional
>
> Std Dev. in peak wavelength shift [nm] of S-cone.

norm_type

> None, optional
>
> - 'max': normalize LMSq functions to max = 1
> - 'area': normalize to area
> - 'power': normalize to power

out

> 'lms' or 'xyz', optional
>
> Determines output.

base

> False, boolean, optional
>
> The returned energy-based LMS cone fundamentals given to the
> precision of 9 sign. figs. if 'True', and to the precision of
> 6 sign. figs. if 'False'.

strategy_2

> True, bool, optional

Use strategy 2 in github.com/ifarup/ciefunctions issue #121 for
computing the weighting factor. If false, strategy 3 is applied.

odata0

None, optional

Dict with uncorrected ocular media and macula density functions and LMS
absorptance functions

None defaults to the ones stored in _DATA

lms_to_xyz_method

None, optional

Method to use to determine lms-to-xyz conversion matrix (options: 'asano',
'cietc197')

allow_negative_values

False, optional

Cone fundamentals or color matching functions should not have negative values.
If False: X[X<0] = 0.

normalize_lms_to_xyz_matrix

False, optional

Normalize that EEW is always at [100,100,100] in XYZ and LMS system.

Returns:

returns

- 'LMS' [or 'XYZ']: ndarray with individual observer equal area-normalized
cone fundamentals. Wavelength have been added.

[- 'M': lms to xyz conversion matrix
- 'trans_lens': ndarray with lens transmission
(no interpolation)
- 'trans_macula': ndarray with macula transmission
(no interpolation)
- 'sens_photopig' : ndarray with photopigment sens.
(no interpolation)]

References: 1. Asano Y, Fairchild MD, and Blondé L, (2016), Individual Colorimetric Observer Model. PLoS
One 11, 1–19.

2. Asano Y, Fairchild MD, Blondé L, and Morvan P (2016). Color matching experiment for highlighting
interobserver variability. Color Res. Appl. 41, 530–539.

3. CIE TC1-36, (2006), Fundamental Chromaticity Diagram with Physiological Axes - Part I (Vienna:
CIE).

4. Asano's Individual Colorimetric Observer Model

5. CIE TC1-97 Python code for cone fundamentals and XYZ cmf calculations (by Ivar Farup and Jan
Henrik Wold, (c) 2012-2017)

luxpy.toolboxes.indvcmf.**getMonteCarloParam**(*n_obs=1, stdDevAllParam={'dsrc': 'matlab',*
'od_L': 17.9, 'od_M': 17.9, 'od_S': 14.7,
'od_lens': 19.1, 'od_macula': 37.2, 'shft_L':
4.0, 'shft_M': 3.0, 'shft_S': 2.5})

Get dict with normally-distributed physiological factors for a population of observers.

Args:

n_obs

> 1, optional
> Number of individual observers in population.

stdDevAllParam

> _DATA['stdev'], optional
> Dict with parameters for:
>> ['od_lens', 'od_macula',
>>> 'od_L', 'od_M', 'od_S',
>>> 'shft_L', 'shft_M', 'shft_S']

Returns:

returns

> dict with n_obs randomly drawn parameters.

luxpy.toolboxes.indvcmf.**genMonteCarloObs**(*n_obs=1,   fieldsize=10,   list_Age=[32],
wl=None,   norm_type=None,   out='lms',
base=False,   strategy_2=True,   odata0=None,
lms_to_xyz_method=None,   al-
low_negative_values=False*)

Monte-Carlo generation of individual observer cone fundamentals.

Args:

n_obs

> 1, optional
> Number of observer CMFs to generate.

list_Age

> list of observer ages or str, optional
> Defaults to 32 (cfr. CIE2006 CMFs)
> If 'us_census': use US population census of 2010
> to generate list_Age.

fieldsize

> fieldsize in degrees (between 2° and 10°), optional
> Defaults to 10°.

wl

> None, optional
> Interpolation/extraplation of :LMS: output to specified wavelengths.
> None: output original _WL

norm_type

> None, optional
> - 'max': normalize LMSq functions to max = 1
> - 'area': normalize to area
> - 'power': normalize to power

out

> 'lms' or 'xyz', optional
> Determines output.

base

> False, boolean, optional
> The returned energy-based LMS cone fundamentals given to the
> precision of 9 sign. figs. if 'True', and to the precision of
> 6 sign. figs. if 'False'.

> **strategy_2**
>
>> True, bool, optional
>>
>> Use strategy 2 in github.com/ifarup/ciefunctions issue #121 for
>>
>> computing the weighting factor. If false, strategy 3 is applied.
>
> **odata0**
>
>> None, optional
>>
>> Dict with uncorrected ocular media and macula density functions and LMS
>> absorptance functions
>>
>> None defaults to the ones stored in _DATA
>
> **lms_to_xyz_method**
>
>> None, optional
>>
>> Method to use to determine lms-to-xyz conversion matrix (options: 'asano',
>> 'cietc197')
>
> **allow_negative_values**
>
>> False, optional
>>
>> Cone fundamentals or color matching functions should not have negative values.
>>> If False: X[X<0] = 0.

Returns:

> **returns**
>
>> LMS [,var_age, vAll]
>>> - LMS: ndarray with population LMS functions.
>>> - var_age: ndarray with population observer ages.
>>> - vAll: dict with population physiological factors (see .keys())

References: 1. Asano Y., Fairchild M.D., and Blondé L., (2016), Individual Colorimetric Observer Model. PLoS One 11, 1–19.

> 2. Asano Y, Fairchild MD, Blondé L, and Morvan P (2016). Color matching experiment for highlighting interobserver variability. Color Res. Appl. 41, 530–539.
>
> 3. CIE TC1-36, (2006), Fundamental Chromaticity Diagram with Physiological Axes - Part I. (Vienna: CIE).
>
> 4. Asano's Individual Colorimetric Observer Model

luxpy.toolboxes.indvcmf.**getCatObs**(*n_cat=10*, *fieldsize=2*, *wl=None*, *norm_type=None*, *out='lms'*, *base=False*, *strategy_2=True*, *odata0=None*, *lms_to_xyz_method=None*, *allow_negative_values=False*)

Generate cone fundamentals for categorical observers.

Args:

> **n_cat**
>
>> 10, optional
>>
>> Number of observer CMFs to generate.
>
> **fieldsize**
>
>> fieldsize in degrees (between 2° and 10°), optional
>>
>> Defaults to 10°.
>
> **out**
>
>> 'LMS' or str, optional
>>
>> Determines output.
>
> **wl**
>
>> None, optional

Interpolation/extraplation of :LMS: output to specified wavelengths.

None: output original _WL

norm_type

None, optional

- 'max': normalize LMSq functions to max = 1

- 'area': normalize to area

- 'power': normalize to power

out

'lms' or 'xyz', optional

Determines output.

base

False, boolean, optional

The returned energy-based LMS cone fundamentals given to the

precision of 9 sign. figs. if 'True', and to the precision of

6 sign. figs. if 'False'.

strategy_2

True, bool, optional

Use strategy 2 in github.com/ifarup/ciefunctions issue #121 for

computing the weighting factor. If false, strategy 3 is applied.

odata0

None, optional

Dict with uncorrected ocular media and macula density functions and LMS
absorptance functions

None defaults to the ones stored in _DATA

lms_to_xyz_method

None, optional

Method to use to determine lms-to-xyz conversion matrix (options: 'asano',
'cietc197')

allow_negative_values

False, optional

Cone fundamentals or color matching functions should not have negative values.

If False: X[X<0] = 0.

Returns:

returns

LMS [,var_age, vAll]

- LMS: ndarray with population LMS functions.

- var_age: ndarray with population observer ages.

- vAll: dict with population physiological factors (see .keys())

Notes: 1. Categorical observers are observer functions that would represent color-normal populations. They are
finite and discrete as opposed to observer functions generated from the individual colorimetric observer
model. Thus, they would offer more convenient and practical approaches for the personalized color
imaging workflow and color matching analyses. Categorical observers were derived in two steps. At
the first step, 10000 observer functions were generated from the individual colorimetric observer model
using Monte Carlo simulation. At the second step, the cluster analysis, a modified k-medoids algorithm,
was applied to the 10000 observers minimizing the squared Euclidean distance in cone fundamentals
space, and categorical observers were derived iteratively. Since the proposed categorical observers are
defined by their physiological parameters and ages, their CMFs can be derived for any target field size.

2. Categorical observers were ordered by the importance; the first categorical observer vas the average observer equivalent to CIEPO06 with 38 year-old for a given field size, followed by the second most important categorical observer, the third, and so on.

3. see: https://www.rit.edu/cos/colorscience/re_AsanoObserverFunctions.php

luxpy.toolboxes.indvcmf.**compute_cmfs**(*fieldsize=10, age=32, wl=None, var_od_lens=0, var_od_macula=0, var_shft_LMS=[0, 0, 0], var_od_LMS=[0, 0, 0], norm_type=None, out='lms', base=False, strategy_2=True, odata0=None, lms_to_xyz_method=None, allow_negative_values=False, normalize_lms_to_xyz_matrix=False*)

Generate Individual Observer CMFs (cone fundamentals) based on CIE2006 cone fundamentals and published literature on observer variability in color matching and in physiological parameters.

Args:

> **age**
>
> > 32 or float or int, optional
> >
> > Observer age
>
> **fieldsize**
>
> > 10, optional
> >
> > Field size of stimulus in degrees (between 2° and 10°).
>
> **wl**
>
> > None, optional
> >
> > Interpolation/extraplation of :LMS: output to specified wavelengths.
> >
> > None: output original _WL
>
> **var_od_lens**
>
> > 0, optional
> >
> > Variation of optical density of lens.
>
> **var_od_macula**
>
> > 0, optional
> >
> > Variation of optical density of macula.
>
> **var_shft_LMS**
>
> > [0, 0, 0] optional
> >
> > Variation (shift) of LMS peak absorptance.
>
> **var_od_LMS**
>
> > [0, 0, 0] optional
> >
> > Variation of LMS optical densities.
>
> **norm_type**
>
> > None, optional
> >
> > - 'max': normalize LMSq functions to max = 1
> >
> > - 'area': normalize to area
> >
> > - 'power': normalize to power
>
> **out**
>
> > 'lms' or 'xyz', optional
> >
> > Determines output.
>
> **base**
>
> > False, boolean, optional
> >
> > The returned energy-based LMS cone fundamentals given to the

precision of 9 sign. figs. if 'True', and to the precision of
6 sign. figs. if 'False'.

strategy_2

True, bool, optional

Use strategy 2 in github.com/ifarup/ciefunctions issue #121 for
computing the weighting factor. If false, strategy 3 is applied.

odata0

None, optional

Dict with uncorrected ocular media and macula density functions and LMS
absorptance functions

None defaults to the ones stored in _DATA

lms_to_xyz_method

None, optional

Method to use to determine lms-to-xyz conversion matrix (options: 'asano',
'cietc197')

allow_negative_values

False, optional

Cone fundamentals or color matching functions should not have negative values.
    If False: X[X<0] = 0.

normalize_lms_to_xyz_matrix

False, optional

Normalize that EEW is always at [100,100,100] in XYZ and LMS system.

Returns:

returns

- 'LMS' [or 'XYZ']: ndarray with individual observer equal area-normalized
    cone fundamentals. Wavelength have been added.

[- 'M': lms to xyz conversion matrix
    - 'trans_lens': ndarray with lens transmission
        (no interpolation)
    - 'trans_macula': ndarray with macula transmission
        (no interpolation)
    - 'sens_photopig' : ndarray with photopigment sens.
        (no interpolation)]

References: 1. Asano Y, Fairchild MD, and Blondé L, (2016), Individual Colorimetric Observer Model. PLoS
One 11, 1–19.

2. Asano Y, Fairchild MD, Blondé L, and Morvan P (2016). Color matching experiment for highlighting
interobserver variability. Color Res. Appl. 41, 530–539.

3. CIE, TC1-36, (2006). Fundamental Chromaticity Diagram with Physiological Axes - Part I (Vienna:
CIE).

4. Asano's Individual Colorimetric Observer Model

5. CIE TC1-97 Python code for cone fundamentals and XYZ cmf calculations (by Ivar Farup and Jan
Henrik Wold, (c) 2012-2017)

```
luxpy.toolboxes.indvcmf.add_to_cmf_dict(bar=None,          cieobs='indv',          K=683,
                                        M=array([[1.0000e+00,          0.0000e+00,
                                        0.0000e+00],     [0.0000e+00,     1.0000e+00,
                                        0.0000e+00],     [0.0000e+00,     0.0000e+00,
                                        1.0000e+00]]))
```
> Add set of cmfs to _CMF dict.
> **Args:**
>> **bar**
>>
>>> None, optional
>>> Set of CMFs. None: initializes to empty ndarray.
>>
>> **cieobs**
>>
>>> 'indv' or str, optional
>>> Name of CMF set.
>>
>> **K**
>>
>>> 683 (lm/W), optional
>>> Conversion factor from radiometric to photometric quantity.
>>
>> **M**
>>
>>> np.eye, optional
>>> Matrix for lms to xyz conversion.

```
luxpy.toolboxes.indvcmf.plot_cmfs(cmf, axh=None, **kwargs)
```
> Plot cmf set.

4.5.3 spdbuild/

> **py**
>
>>> • __init__.py
>>>
>>> • spdbuilder.py
>>>
>>> • spdbuilder2020.py
>>>
>>> • spdoptimzer2020.py
>
> **namespace** luxpy.spdbuild/

Module for building and optimizing SPDs

> spdbuilder.py

Functions

> **gaussian_spd()** Generate Gaussian spectrum.
>
> **butterworth_spd()** Generate Butterworth based spectrum.
>
> **lorentzian2_spd()** Generate 2nd order Lorentzian based spectrum.
>
> **roundedtriangle_spd()** Generate a rounded triangle based spectrum.
>
> **mono_led_spd()** Generate monochromatic LED spectrum based on a Gaussian or butter-
>> worth profile or according to Ohno (Opt. Eng. 2005).

> **spd_builder()** Build spectrum based on Gaussians, monochromatic and/or phophor LED spectra.

> **color3mixer()** Calculate fluxes required to obtain a target chromaticity when (additively) mixing 3 light sources.

> **colormixer()** Calculate fluxes required to obtain a target chromaticity when (additively) mixing N light sources.

> **colormixer_pinv()** Additive color mixer of N primaries using using Moore-Penrose pseudo-inverse matrix.

> **spd_builder()** Build spectrum based on Gaussians, monochromatic and/or phophor LED-type spectra.

> **get_w_summed_spd()** Calculate weighted sum of spds.

> **fitnessfcn()** Fitness function that calculates closeness of solution x to target values for specified objective functions.

> **spd_constructor_2()** Construct spd from spectral model parameters using pairs of intermediate sources.

> **spd_constructor_3()** Construct spd from spectral model parameters using trio's of intermediate sources.

> **spd_optimizer_2_3()** Optimizes the weights (fluxes) of a set of component spectra by combining pairs (2) or trio's (3) of components to intermediate sources until only 3 remain. Color3mixer can then be called to calculate required fluxes to obtain target chromaticity and fluxes are then back-calculated.

> **get_optim_pars_dict()** Setup dict with optimization parameters.

> **initialize_spd_model_pars()** Initialize spd_model_pars (for spd_constructor) based on type of component_data.

> **initialize_spd_optim_pars()** Initialize spd_optim_pars (x0, lb, ub for use with math.minimizebnd) based on type of component_data.

> **spd_optimizer()** Generate a spectrum with specified white point and optimized for certain objective functions from a set of component spectra or component spectrum model parameters.

Module for building and optimizing SPDs (2)

This module implements a class based spectral optimizer. It differs from the spdoptimizer function in spdbuild.py, in that it can use several different minimization algorithms, as well as a user defined method. It is also written such that the user can easily write his own primary constructor function. It supports the '3mixer' algorithm (but no '2mixer') and a 'no-mixer' algorithm (chromaticity as part of the list of objectives) for calculating the mixing contributions of the primaries.

Functions

> **gaussian_prim_constructor()** constructs a gaussian based primary set.

> **_setup_wlr()** Initialize the wavelength range for use with PrimConstructor.

> **_extract_prim_optimization_parameters()** Extract the primary parameters from the optimization vector x and the pdefs dict for use with PrimConstructor.

_stack_wlr_spd() Stack the wavelength range 'on top' of the spd values for use with Prim-
Constructor.

PrimConstructor class for primary (spectral) construction

Minimizer class for minimization of fitness of each of the objective functions

ObjFcns class to specify one or more objective functions for minimization

SpectralOptimizer class for spectral optimization (initialization and run)

spd_optimizer2() Generate a spectrum with specified white point and optimized for certain
objective functions from a set of component spectra or component spectrum model
parameters (functional wrapper around SpectralOptimizer class).

Notes

1. See examples below (in spdoptimizer2020.'__main__') for use.

`luxpy.toolboxes.spdbuild.`**`gaussian_spd`**(*peakwl=530, fwhm=20, wl=[360.0, 830.0, 1.0],
with_wl=True*)

Generate Gaussian spectrum.

Args:

> **peakw**
>
>> int or float or list or ndarray, optional
>> Peak wavelength
>
> **fwhm**
>
>> int or float or list or ndarray, optional
>> Full-Width-Half-Maximum of gaussian.
>
> **wl**
>
>> _WL3, optional
>> Wavelength range.
>
> **with_wl**
>
>> True, optional
>> True outputs a ndarray with first row wavelengths.

Returns:

> **returns**
>
>> ndarray with spectra.

Note:

> Gaussian:
>> g = exp(-0.5*((wl - peakwl)/sig)**2)
> with sig = fwhm/(2*(2*np.log(2))**0.5)

`luxpy.toolboxes.spdbuild.`**`butterworth_spd`**(*peakwl=530, fwhm=20, bw_order=1, wl=[360.0,
830.0, 1.0], with_wl=True*)

Generate Butterworth based spectrum.

Args:

> **peakw**
>
>> int or float or list or ndarray, optional
>> Peak wavelength
>
> **fwhm**
>
>> int or float or list or ndarray, optional

Full-Width-Half-Maximum of butterworth.

bw_order

1, optional
Order of the butterworth function.

wl

_WL3, optional
Wavelength range.

with_wl

True, optional
True outputs a ndarray with first row wavelengths.

Returns:

returns

ndarray with spectra.

Note:

Butterworth :
bw = 1 / (1 + ((2*(wl - peakwl)/fwhm)**2))

luxpy.toolboxes.spdbuild.**lorentzian2_spd**(*peakwl=530, fwhm=20, wl=[360.0, 830.0, 1.0],*
with_wl=True)
Generate 2nd order Lorentzian spectrum.

Args:

peakw

int or float or list or ndarray, optional
Peak wavelength

fwhm

int or float or list or ndarray, optional
Full-Width-Half-Maximum of lorentzian.

wl

_WL3, optional
Wavelength range.

with_wl

True, optional
True outputs a ndarray with first row wavelengths.

Returns:

returns

ndarray with spectra.

Note:

Lorentzian (2nd order):
lz = (1 + ((n*(wl - peakwl)/fwhm)**2))**(-2)
with n = 2*(2**0.5-1)**0.5

luxpy.toolboxes.spdbuild.**roundedtriangle_spd**(*peakwl=530, fwhm=100, rounding=0.5,*
wl=[360.0, 830.0, 1.0], with_wl=True,
min_v=0.0, max_v=1.0, fw=100, rw=100)
Generate rounded triangle spectrum.

Args:

peakw

int or float or list or ndarray, optional

Peak wavelength

fwhm

int or float or list or ndarray, optional

Full-Width-Half-Maximum of rounded triangle.

rounding

int or float or list or ndarray, optional

Amount of rounding of triangle corners (top, bottom-left, bottom-right)

wl

_WL3, optional

Wavelength range.

with_wl

True, optional

True outputs a ndarray with first row wavelengths.

min_v, max_v

0.0, 1.0, optional

Minimum and maximum of spd.

fw

100, optional

front width of triangle.

Only used when fwhm is set to None.

rw

100, optional

rear width of triangle.

Only used when fwhm is set to None.

Returns:

returns

ndarray with spectra.

luxpy.toolboxes.spdbuild.**mono_led_spd**(*peakwl=530, fwhm=20, wl=[360.0, 830.0, 1.0], with_wl=True, strength_shoulder=2, bw_order=-1*)

Generate monochromatic LED spectrum based on a Gaussian or or Lorentzian or butterworth profile or according to Ohno (Opt. Eng. 2005).

Args:

peakw

int or float or list or ndarray, optional

Peak wavelength

fwhm

int or float or list or ndarray, optional

Full-Width-Half-Maximum of gaussian used to simulate led.

wl

_WL3, optional

Wavelength range.

with_wl

True, optional

True outputs a ndarray with first row wavelengths.

strength_shoulder

2, optional

Determines the strength of the spectrum shoulders of the mono led.

A value of 0 reduces to a pure Gaussian model (if bw_order >= -1).

bw_order

-1, optional

Order of Butterworth function.

If -1 or 0: spd profile is Ohno's gaussian based

(to obtain pure Gaussian: set strength_shoulder = 0).

If -2: spd profile is Lorentzian,

else (>0): Butterworth.

Returns:

returns

ndarray with spectra.

Note:

Gaussian:

g = exp(-0.5*((wl - peakwl)/sig)**2)

with sig = fwhm/(2*(2*np.log(2))**0.5)

Lorentzian (2nd order):

lz = (1 + ((n*(wl - peakwl)/fwhm)**2))**(-2)

with n = 2*(2**0.5-1)**0.5

Butterworth :

bw = 1 / (1 + ((2*(wl - peakwl)/fwhm)**2))

Ohno's model:

ohno = (g + strength_shoulder*g**5)/(1+strength_shoulder)

mono_led_spd = ohno*((bw_order >= -1) & (bw_order <= 0)).T + bw*(bw_order > 0).T + lz*((bw_order >=-2) & (bw_order < -1)).T

Reference: 1. Ohno Y (2005). Spectral design considerations for white LED color rendering. Opt. Eng. 44, 111302.

luxpy.toolboxes.spdbuild.**phosphor_led_spd**(*peakwl=450, fwhm=20, wl=[360.0, 830.0, 1.0], bw_order=-1, with_wl=True, strength_shoulder=2, strength_ph=0, peakwl_ph1=530, fwhm_ph1=80, strength_ph1=1, peakwl_ph2=560, fwhm_ph2=80, strength_ph2=None, use_piecewise_fcn=False, verbosity=0, out='spd'*)

Generate phosphor LED spectrum with up to 2 phosphors based on Smet (Opt. Expr. 2011).

Model:

1) If strength_ph2 is not None:

phosphor_spd = (strength_ph1*mono_led_spd(peakwl_ph1, ..., strength_shoulder = 1)

+ strength_ph2)*mono_led_spd(peakwl_ph2, ..., strength_shoulder = 1))

/ (strength_ph1 + strength_ph2)

else:

phosphor_spd = (strength_ph1*mono_led_spd(peakwl_ph1, ..., strength_shoulder = 1)

+ (1-strength_ph1)*mono_led_spd(peakwl_ph2, ..., strength_shoulder = 1))

2) S = (mono_led_spd() + strength_ph*(phosphor_spd/phosphor_spd.max()))/(1 + strength_ph)

3) piecewise_fcn = S for wl < peakwl and 1 for wl >= peakwl

4) phosphor_led_spd = S*piecewise_fcn

Args:

peakw

int or float or list or ndarray, optional
Peak wavelengths of the monochromatic led.

fwhm

int or float or list or ndarray, optional
Full-Width-Half-Maximum of mono_led spectrum.

wl

_WL3, optional
Wavelength range.

bw_order

-1, optional
Order of Butterworth function.
If -1 or 0: spd profile is Ohno's gaussian based
    (to obtain pure Gaussian: set strength_shoulder = 0).
If -2: spd profile is Lorentzian,
else (>0): Butterworth.
Note that this only applies to the monochromatic led spds and not
the phosphors spds (these are always gaussian based).

with_wl

True, optional
True outputs a ndarray with first row wavelengths.

strength_shoulder

2, optiona l
Determines the strength of the spectrum shoulders of the mono led.

strength_ph

0, optional
Total contribution of phosphors in mixture.

peakwl_ph1

int or float or list or ndarray, optional
Peak wavelength of the first phosphor.

fwhm_ph1

int or float or list or ndarray, optional
Full-Width-Half-Maximum of gaussian used to simulate first phosphor.

strength_ph1

1, optional

Strength of first phosphor in phosphor mixture.

If :strength_ph2: is None: value should be in the [0,1] range.

peakwl_ph2

int or float or list or ndarray, optional

Peak wavelength of the second phosphor.

fwhm_ph2

int or float or list or ndarray, optional

Full-Width-Half-Maximum of gaussian used to simulate second phosphor.

strength_ph2

None, optional

Strength of second phosphor in phosphor mixture.

If None: strength is calculated as (1-:strength_ph1:)

:target: np2d([100,1/3,1/3]), optional

ndarray with Yxy chromaticity of target.

verbosity

0, optional

If > 0: plots spectrum components (mono_led, ph1, ph2, . . .)

out

'spd', optional

Specifies output.

use_piecewise_fcn

False, optional

True: uses piece-wise function as in Smet et al. 2011. Can give

non_smooth spectra optimized from components to which it is applied.

Returns:

returns

spd, component_spds

ndarrays with spectra (and component spds used to build the

final spectra)

References: 1. Ohno Y (2005). Spectral design considerations for white LED color rendering. Opt. Eng. 44, 111302.

2. Smet K, Ryckaert WR, Pointer MR, Deconinck G, and Hanselaer P (2011). Optimal colour quality of LED clusters based on memory colours. Opt. Express 19, 6903–6912.

luxpy.toolboxes.spdbuild.**spd_builder**(*flux=None, component_spds=None, peakwl=450, fwhm=20, bw_order=-1, pair_strengths=None, wl=[360.0, 830.0, 1.0], with_wl=True, strength_shoulder=2, strength_ph=0, peakwl_ph1=530, fwhm_ph1=80, strength_ph1=1, peakwl_ph2=560, fwhm_ph2=80, strength_ph2=None, target=None, tar_type='Yuv', cspace_bwtf={}, cieobs='1931_2', use_piecewise_fcn=False, verbosity=0, out='spd', **kwargs*)*

Build spectrum based on Gaussian, monochromatic and/or phophor type spectra.

Args:

flux

None, optional

Fluxes of each of the component spectra.

None outputs the individual component spectra.

component_spds

None or ndarray, optional

If None: calculate component spds from input args.

peakw

int or float or list or ndarray, optional

Peak wavelengths of the monochromatic led.

fwhm

int or float or list or ndarray, optional (but must be same shape as peakw!)

Full-Width-Half-Maximum of gaussian.

wl

_WL3, optional

Wavelength range.

bw_order

-1, optional

Order of Butterworth function.

If -1 or 0: spd profile is Ohno's gaussian based

(to obtain pure Gaussian: set strength_shoulder = 0).

If -2: spd profile is Lorentzian,

else (>0): Butterworth.

Note that this only applies to the monochromatic led spds and not

the phosphors spds (these are always gaussian based).

pair_strengths

ndarray with pair_strengths of mono_led spds, optional

If None: will be randomly selected, possibly resulting in

unphysical (out-of-gamut) solution.

with_wl

True, optional

True outputs a ndarray with first row wavelengths.

strength_shoulder

2, optiona l

Determines the strength of the spectrum shoulders of the mono led.

strength_ph

0, optional

Total contribution of phosphors in mixture.

peakwl_ph1

int or float or list or ndarray, optional

Peak wavelength of the first phosphor.

fwhm_ph1

int or float or list or ndarray, optional

Full-Width-Half-Maximum of gaussian used to simulate first phosphor.

strength_ph1

1, optional

Strength of first phosphor in phosphor mixture.

If :strength_ph2: is None: value should be in the [0,1] range.

peakwl_ph2

int or float or list or ndarray, optional

Peak wavelength of the second phosphor.

fwhm_ph2

int or float or list or ndarray, optional

Full-Width-Half-Maximum of gaussian used to simulate second phosphor.

strength_ph2

None, optional

Strength of second phosphor in phosphor mixture.

If None: strength is calculated as (1-:strength_ph1:)

:target: np2d([100,1/3,1/3]), optional

ndarray with Yxy chromaticity of target.

verbosity

0, optional

If > 0: plots spectrum components (mono_led, ph1, ph2, ...)

out

'spd', optional

Specifies output.

use_piecewise_fcn

False, optional

True: uses piece-wise function as in Smet et al. 2011. Can give

non_smooth spectra optimized from components to which it is

applied.

target

None, optional

ndarray with Yxy chromaticity of target.

If None: don't override phosphor strengths, else calculate strength

to obtain :target: using color3mixer().

If not None AND strength_ph is None or 0: components are

monochromatic and colormixer is used to optimize fluxes to

obtain target chromaticity (N can be > 3 components)

tar_type

'Yxy' or str, optional

Specifies the input type in :target: (e.g. 'Yxy' or 'cct')

cieobs

_CIEOBS, optional

CIE CMF set used to calculate chromaticity values.

cspace_bwtf

{}, optional

Backward (..._to_xyz) transform parameters

(see colortf()) to go from :tar_type: to 'Yxy')

Returns:

returns

ndarray with spectra.

Note: 1. Target-optimization is only for phophor_leds with three components (blue pump, ph1 and ph2) spanning a sufficiently large gamut.

References: 1. Ohno Y (2005). Spectral design considerations for white LED color rendering. Opt. Eng. 44, 111302.

2. Smet K, Ryckaert WR, Pointer MR, Deconinck G, and Hanselaer P (2011). Optimal colour quality of LED clusters based on memory colours. Opt. Express 19, 6903–6912.

luxpy.toolboxes.spdbuild.**get_w_summed_spd**(*w*, *spds*)

> Calculate weighted sum of spds.
>
> **Args:**
>> **w**
>>
>>> ndarray with weigths (e.g. fluxes)
>>
>> **spds**
>>
>>> ndarray with component spds.
>
> **Returns:**
>> **returns**
>>
>>> ndarray with weighted sum.

luxpy.toolboxes.spdbuild.**fitnessfcn**(*x,   spd_constructor,   spd_constructor_pars=None,*
F_rss=True,   decimals=[3],   obj_fcn=[None],
obj_fcn_pars=[{}],   obj_fcn_weights=[1],
obj_tar_vals=[0], verbosity=0, out='F')

> Fitness function that calculates closeness of solution x to target values for specified objective functions.
>
> **Args:**
>> **x**
>>
>>> ndarray with parameter values
>>
>> **spd_constructor**
>>
>>> function handle to a function that constructs the spd from parameter values in :x:.
>>
>> **spd_constructor_pars**
>>
>>> None, optional,
>>>
>>> Parameters required by :spd_constructor:
>>
>> **F_rss**
>>
>>> True, optional
>>>
>>> Take Root-Sum-of-Squares of 'closeness' values between target and
>>>
>>> objective function values.
>>
>> **decimals**
>>
>>> 3, optional
>>>
>>> List of rounding decimals of objective function values.
>>
>> **obj_fcn**
>>
>>> [None] or list, optional
>>>
>>> List of function handles to objective function.
>>
>> **obj_fcn_weights**
>>
>>> [1] or list, optional.
>>>
>>> List of weigths for each obj. fcn
>>
>> **obj_fcn_pars**
>>
>>> [None] or list, optional
>>>
>>> List of parameter dicts for each obj. fcn.
>>
>> **obj_tar_vals**

> > [0] or list, optional
> >
> > List of target values for each objective function.
>
> **verbosity**
>
> > 0, optional
> >
> > If > 0: print intermediate results.
>
> **out**
>
> > 'F', optional
> >
> > Determines output.

> **Returns:**
>
> > **F**
> >
> > float or ndarray with fitness value for current solution :x:.

luxpy.toolboxes.spdbuild.**spd_constructor_2**(*x*, *constructor_pars={}*, ***kwargs*)

> Construct spd from model parameters using pairs of intermediate sources.

> Pairs (odd,even) of components are selected and combined using
> > 'pair_strength'. This process is continued until only 3 intermediate
> > (combined) sources remain. Color3mixer is then used to calculate the
> > fluxes for the remaining 3 sources, after which the fluxes of all
> > components are back-calculated.

> **Args:**
>
> > **x**
> >
> > > vector of optimization parameters.
> >
> > **constructor_pars**
> >
> > > dict with model parameters.
> > > Key 'list' determines which parameters are in :x: and key 'len'
> > > (Specifies the number of variables representing each parameter).

> **Returns:**
>
> > **returns**
> >
> > > spd, M, spds
> > > ndarrays with spectrum corresponding to x, M the fluxes of
> > > the spectral components of spd and spds the spectral components
> > > themselves.

luxpy.toolboxes.spdbuild.**color3mixer**(*Yxyt*, *Yxy1*, *Yxy2*, *Yxy3*)

> Calculate fluxes required to obtain a target chromaticity when (additively) mixing 3 light sources.

> **Args:**
>
> > **Yxyt**
> >
> > > ndarray with target Yxy chromaticities.
> >
> > **Yxy1**
> >
> > > ndarray with Yxy chromaticities of light sources 1.
> >
> > **Yxy2**
> >
> > > ndarray with Yxy chromaticities of light sources 2.
> >
> > **Yxy3**
> >
> > > ndarray with Yxy chromaticities of light sources 3.

> **Returns:**

> **M**
>
>> ndarray with fluxes.
>
> Note: Yxyt, Yxy1, … can contain multiple rows, referring to single mixture.

luxpy.toolboxes.spdbuild.**colormixer**(*Yxyt=None*, *Yxyi=None*, *n=4*, *pair_strengths=None*, *source_order=None*)

> Calculate fluxes required to obtain a target chromaticity when (additively) mixing N light sources.
>
> **Args:**
>
>> **Yxyt**
>>
>>> ndarray with target Yxy chromaticities.
>>> Defaults to equi-energy white.
>>
>> **Yxyi**
>>
>>> ndarray with Yxy chromaticities of light sources i = 1 to n.
>>
>> **n**
>>
>>> 4 or int, optional
>>> Number of source components to randomly generate when Yxyi is None.
>>
>> **pair_strengths**
>>
>>> ndarray with light source pair strengths.
>>
>> **source_order**
>>
>>> ndarray with order of source components.
>>> If None: use np.arange(n)
>
> **Returns:**
>
>> **M**
>>
>>> ndarray with fluxes.
>
> **Note:**
>
>> **Algorithm**
>>
>>> 1. Loop over all source components and create intermediate sources
>>>   from all (even,odd)-pairs using the relative strengths
>>>   of the pair (specified in pair_strengths).
>>> 2. Collect any remaining sources.
>>> 3. Combine with new intermediate source components
>>> 4. Repeat 1-3 until there are only 3 source components left.
>>> 5. Use color3mixer to calculate the required fluxes of the 3 final
>>>   intermediate components to obtain the target chromaticity.
>>> 6. Backward calculate the fluxes of all original source components
>>>   from the 3 final intermediate fluxes.

luxpy.toolboxes.spdbuild.**colormixer_pinv**(*xyzt*, *xyzi*, *input_fmt='xyz'*)

> Additive color mixer of N primaries using using Moore-Penrose pseudo-inverse matrix.
>
> **Args:**
>
>> **xyzt**
>>
>>> ndarray with target XYZ tristimulus values or Yxy chromaticity coordinates.
>>
>> **xyzi**
>>
>>> ndarray with XYZ tristimulus values or Yxy chromaticity coordinates of light sources
>>> i = 1 to n.
>>
>> **input_fmt**
>>
>>> 'xyz', optional
>>> Format specifier of :xyzt: and :xyzi: input arguments.

- options: 'xyz', 'Yxy'

Returns:

w

ndarray with fluxes (weights) of each of the primaries in the mixture.

luxpy.toolboxes.spdbuild.**spd_constructor_3**(*x*, *constructor_pars={}, **kwargs*)
Construct spd from model parameters using trio's of intermediate sources.

The triangle/trio method creates for all possible combinations of 3 primary
component spectra a spectrum that results in the target chromaticity
using color3mixer() and then optimizes the weights of each of the latter
spectra such that adding them (additive mixing) results in obj_vals as
close as possible to the target values.

Args:

x

vector of optimization parameters.

constructor_pars

dict with model parameters.
Key 'list' determines which parameters are in :x: and key 'len'
(specifies the number of variables representing each parameter).

Returns:

returns

spd, M, spds
ndarrays with spectrum corresponding to x, M the fluxes of
the spectral components of spd and spds the spectral components
themselves.

luxpy.toolboxes.spdbuild.**spd_optimizer_2_3**(*optimizer_type='2mixer',
spd_constructor=None,
spd_model_pars=None, component_data=4,
N_components=None, wl=[360.0, 830.0, 1.0],
allow_nongaussianbased_mono_spds=False,
Yxy_target=array([[1.0000e+02, 3.3333e-
01, 3.3333e-01]]), cieobs='1931_2',
obj_fcn=[None], obj_fcn_pars=[{}],
obj_fcn_weights=[1], obj_tar_vals=[0],
decimals=[5], minimize_method='Nelder-
Mead', minimize_opts=None, F_rss=True,
verbosity=0, **kwargs*)

Optimizes the weights (fluxes) of a set of component spectra by combining
pairs (2) or trio's (3) of components to intermediate sources until only 3
remain. Color3mixer can then be called to calculate required fluxes to
obtain target chromaticity and fluxes are then back-calculated.

Args:

optimizer_type

'2mixer' or '3mixer' or 'user', optional
Specifies whether to optimize spectral model parameters by

combining pairs or trio's of comonponents.

spd_constructor

None, optional

Function handle to user defined spd_constructor function.

Input: fcn(x, constructor_pars = {}, kwargs)

Output: spd,M,spds

nd array with:

- spd: spectrum resulting from x

- M: fluxes of all component spds

- spds: component spds (in [N+1,wl] format)

(See e.g. spd_constructor_2 or spd_constructor_3)

spd_model_pars

dict with model parameters required by spd_constructor

and with optimization parameters required by minimize (x0, lb, ub). .

Only used when :optimizer_type: == 'user'.

component_data

4, optional

Component spectra data:

If int: specifies number of components used in optimization

(peakwl, fwhm and pair_strengths will be optimized).

If dict: generate components based on parameters (peakwl, fwhm,

pair_strengths, etc.) in dict.

(keys with None values will be optimized)

If ndarray: optimize pair_strengths of component spectra.

N_components

None, optional

Specifies number of components used in optimization. (only used

when :component_data: is dict and user wants to override dict.

Note that shape of parameters arrays must match N_components).

allow_nongaussianbased_mono_spds

False, optional

False: use pure Gaussian based monochrom. spds.

wl

_WL3, optional

Wavelengths used in optimization when :component_data: is not

ndarray with spectral data.

Yxy_target

np2d([100,1/3,1/3]), optional

ndarray with Yxy chromaticity of target.

cieobs

_CIEOBS, optional

CIE CMF set used to calculate chromaticity values if not provided

in :Yxyi:.

F_rss

True, optional

> > Take Root-Sum-of-Squares of 'closeness' values between target and
> > objective function values.

decimals

> > 5, optional
> > Rounding decimals of objective function values.

obj_fcn

> > [None] or list, optional
> > Function handles to objective function.

obj_fcn_weights

> > [1] or list, optional.
> > Weigths for each obj. fcn

obj_fcn_pars

> > [None] or list, optional
> > Parameter dicts for each obj. fcn.

obj_tar_vals

> > [0] or list, optional
> > Target values for each objective function.

minimize_method

> > 'Nelder-Mead', optional
> > Optimization method used by minimize function.

minimize_opts

> > None, optional
> > Dict with minimization options.
> > None defaults to: {'xtol': 1e-5, 'disp': True, 'maxiter': 1000*Nc,
> > > 'maxfev' : 1000*Nc,'fatol': 0.01}

verbosity

> > 0, optional
> > If > 0: print intermediate results.

> **Returns:**

> > **returns**

> > > M, spd_opt, obj_vals
> > > > - 'M': ndarray with fluxes for each component spectrum.
> > > > - 'spd_opt': optimized spectrum.
> > > > - 'obj_vals': values of the obj. fcns for the optimized spectrum.

luxpy.toolboxes.spdbuild.**get_optim_pars_dict**(*target=array([[1.0000e+02, 3.3333e-01, 3.3333e-01]]), tar_type='Yxy', cieobs='1931_2', optimizer_type='2mixer', spd_constructor=None, spd_model_pars=None, cspace='Yuv', cspace_bwtf={}, cspace_fwtf={}, component_spds=None, N_components=None, obj_fcn=[None], obj_fcn_pars=[{}], obj_fcn_weights=[1], obj_tar_vals=[0], decimals=[5], minimize_method='Nelder-Mead', minimize_opts=None, F_rss=True, peakwl=[450, 530, 610], fwhm=[20, 20, 20], allow_nongaussianbased_mono_spds=False, bw_order=[-1], wl=[360.0, 830.0, 1.0], with_wl=True, strength_shoulder=2, strength_ph=[0], use_piecewise_fcn=False, peakwl_ph1=[530], fwhm_ph1=[80], strength_ph1=[1], peakwl_ph2=[560], fwhm_ph2=[80], strength_ph2=None, verbosity=0, pair_strengths=None, triangle_strengths=None, peakwl_min=[400], peakwl_max=[700], fwhm_min=[5], fwhm_max=[600], bw_order_min=[-2], bw_order_max=[100]*)

> Setup dict with optimization parameters.
>
> **Args:** See ?spd_optimizer for more info.
>
> **Returns:**
>
> > > **opts**
> > >
> > > > dict with keys and values of the function's keywords and values.

luxpy.toolboxes.spdbuild.**initialize_spd_model_pars**(*component_data, N_components=None, allow_nongaussianbased_mono_spds=False, optimizer_type='2mixer', wl=[360.0, 830.0, 1.0]*)

> Initialize spd_model_pars dict (for spd_constructor) based on type of component_data.
>
> **Args:**
>
> > **component_data**
> >
> > > None, optional
> > >
> > > Component spectra data:
> > >
> > > If int: specifies number of components used in optimization
> > >
> > > > (peakwl, fwhm and pair_strengths will be optimized).
> > >
> > > If dict: generate components based on parameters (peakwl, fwhm,
> > >
> > > > pair_strengths, etc.) in dict.
> > >
> > > > (keys with None values will be optimized)
> > >
> > > If ndarray: optimize pair_strengths of component spectra.
> >
> > **N_components**
> >
> > > None, optional
> > >
> > > Specifies number of components used in optimization. (only used

---

when :component_data: is dict and user wants to override dict.

Note that shape of parameters arrays must match N_components).

allow_nongaussianbased_mono_spds

> False, optional
>
> > - False: use Gaussian based monochrom. spds.
> >
> > - True: also allow butterworth and lorentzian type monochrom. spds while optimizing.

optimizer_type

> '2mixer', optional
>
> Type of spectral optimization routine.
>
> (other options: '3mixer', 'search')

wl

> _WL3, optional
>
> Wavelengths used in optimization when :component_data: is not an
>
> ndarray with spectral data.

Returns:

> **spd_model_pars**
>
> > dict with spectrum-model parameters

luxpy.toolboxes.spdbuild.**initialize_spd_optim_pars**(*component_data, N_components=None, allow_nongaussianbased_mono_spds=False, optimizer_type='2mixer', wl=[360.0, 830.0, 1.0], spd_model_pars=None*)

Initialize spd_optim_pars dict based on type of component_data.

Args:

> **component_data**
>
> > None, optional
> >
> > Component spectra data:
> >
> > If int: specifies number of components used in optimization
> >
> > > (peakwl, fwhm and pair_strengths will be optimized).
> >
> > If dict: generate components based on parameters (peakwl, fwhm,
> >
> > > pair_strengths, etc.) in dict.
> > >
> > > (keys with None values will be optimized)
> >
> > If ndarray: optimize pair_strengths of component spectra.
>
> **N_components**
>
> > None, optional
> >
> > Specifies number of components used in optimization. (only used
> >
> > when :component_data: is dict and user wants to override dict.
> >
> > Note that shape of parameters arrays must match N_components).
>
> **allow_nongaussianbased_mono_spds**
>
> > False, optional
> >
> > False: use Gaussian based monochrom. spds.
>
> **optimizer_type**
>
> > '2mixer', optional
> >
> > Type of spectral optimization routine.
> >
> > (other options: '3mixer', 'search')

> **wl**
>> _WL3, optional
>> Wavelengths used in optimization when :component_data: is not an
>> ndarray with spectral data.
>
> **spd_model_pars**
>> None, optional
>> If None, initialize based on type of component_data.
>> else: initialize on pre-defined spd_model_pars dict.

Returns:
> **spd_optim_pars**
>> dict with optimization parameters (x0, ub, lb)

luxpy.toolboxes.spdbuild.**get_primary_fluxratios**(*res*, *primaries*, *Ytarget=1*, *ptype='pu'*,
                                                    cieobs='1931_2', *out='M, Sopt'*)

> Get flux ratios of primaries.

Args:
> **res**
>> dict or ndarray with optimized fluxes for component spds normalized to max = 1.
>> (output of spd_optimizer)
>
> **primaries**
>> ndarray with primary spectra.
>
> **Ytarget**
>> 1, optional
>> M will be scaled to result in a photo-/radio-metric power of Ytarget
>
> **ptype**
>> 'pu' or 'ru', optional
>> Type of power:
>> -'pu': photometric units
>> -'ru': radiometric units
>
> **cieobs**
>> _CIEOBS, optional
>> CMF set/Vlambda to use in calculation of power.

Returns:
> **M**
>> ndarray with flux ratios.
>
> **Sopt**
>> ndarray with optimized scaled spectrum.

```
luxpy.toolboxes.spdbuild.spd_optimizer(target=array([[1.0000e+02, 3.3333e-01, 3.3333e-
                                        01]]),    tar_type='Yxy',    cieobs='1931_2',    opti-
                                        mizer_type='2mixer',         spd_constructor=None,
                                        spd_model_pars=None,             cspace='Yuv',
                                        cspace_bwtf={},      cspace_fwtf={},      compo-
                                        nent_spds=None,           N_components=None,
                                        obj_fcn=[None],              obj_fcn_pars=[{}],
                                        obj_fcn_weights=[1],          obj_tar_vals=[0],
                                        decimals=[5],           minimize_method='Nelder-
                                        Mead',     minimize_opts=None,     F_rss=True,
                                        peakwl=[450, 530, 610], fwhm=[20, 20, 20],
                                        allow_nongaussianbased_mono_spds=False,
                                        bw_order=[-1],      wl=[360.0,     830.0,    1.0],
                                        with_wl=True,              strength_shoulder=2,
                                        strength_ph=[0],       use_piecewise_fcn=False,
                                        peakwl_ph1=[530],             fwhm_ph1=[80],
                                        strength_ph1=[1],            peakwl_ph2=[560],
                                        fwhm_ph2=[80], strength_ph2=None, verbosity=0,
                                        pair_strengths=None,         peakwl_min=[400],
                                        peakwl_max=[700],            fwhm_min=[5],
                                        fwhm_max=[600],              bw_order_min=-2,
                                        bw_order_max=100, out='spds,M')
```

Generate a spectrum with specified white point and optimized for certain objective functions from a set of component spectra or component spectrum model parameters.

Args:

> **target**
>
>> np2d([100,1/3,1/3]), optional
>> ndarray with Yxy chromaticity of target.
>
> **tar_type**
>
>> 'Yxy' or str, optional
>> Specifies the input type in :target: (e.g. 'Yxy' or 'cct')
>
> **cieobs**
>
>> _CIEOBS, optional
>> CIE CMF set used to calculate chromaticity values, if not provided
>> in :Yxyi:.
>
> **optimizer_type**
>
>> '2mixer', optional
>> Specifies type of chromaticity optimization
>> ('3mixer' or '2mixer' or 'search')
>> For help on '2mixer' and '3mixer' algorithms, see notes below.
>
> **spd_constructor**
>
>> None, optional
>> Function handle to user defined spd_constructor function.
>>> Input: fcn(x, constructor_pars = {}, kwargs)
>>> Output: spd,M,spds
>>>> nd array with:
>>>>> - spd: spectrum resulting from x

> > > - M: fluxes of all component spds
> > > - spds: component spds (in [N+1,wl] format)
> > (See e.g. spd_constructor_2 or spd_constructor_3)

spd_model_pars

> dict with model parameters required by spd_constructor
> and with optimization parameters required by minimize (x0, lb, ub). .
> Only used when :optimizer_type: == 'user'.

cspace

> 'Yuv', optional
> Color space for 'search'-type optimization.

cspace_bwtf

> {}, optional
> Backward (cspace_to_xyz) transform parameters
> (see colortf()) to go from :tar_type: to 'Yxy').

cspace_fwtf

> {}, optional
> Forward (xyz_to_cspace) transform parameters
> (see colortf()) to go from xyz to :cspace:).

component_spds

> ndarray of component spectra.
> If None: they are built from input args.

N_components

> None, optional
> Specifies number of components used in optimization. (only used
> when :component_data: is dict and user wants to override dict value
> Note that shape of parameters arrays must match N_components).

allow_nongaussianbased_mono_spds

> False, optional
> False: use Ohno monochromatic led spectra based on Gaussian spds.
> True: also use Butterworth and Lorentzian spds.

wl

> _WL3, optional
> Wavelengths used in optimization when :component_data: is not an
> ndarray with spectral data.

F_rss

> True, optional
> Take Root-Sum-of-Squares of 'closeness' values between target and
> objective function values.

decimals

> 5, optional
> Rounding decimals of objective function values.

obj_fcn

> [None] or list, optional
> Function handles to objective function.

obj_fcn_weights

[1] or list, optional.

Weigths for each obj. fcn

obj_fcn_pars

[None] or list, optional

Parameter dicts for each obj. fcn.

obj_tar_vals

[0] or list, optional

Target values for each objective function.

minimize_method

'Nelder-Mead', optional

Optimization method used by minimize function.

minimize_opts

None, optional

Dict with minimization options.

None defaults to: {'xtol': 1e-5, 'disp': True, 'maxiter': 1000*Nc,

'maxfev' : 1000*Nc,'fatol': 0.01}

verbosity

0, optional

If > 0: print intermediate results.

out

'spds,M', optional

Determines output of function.

Note: peakwl:, :fwhm:, . . . : see ?spd_builder for more info.

Returns:

returns

spds, M

- 'spds': optimized spectrum.

- 'M': ndarray with fluxes for each component spectrum.

Notes:

Optimization algorithms

1. '2mixer': Pairs (odd,even) of components are selected and combined using 'pair_strength'. This process is continued until only 3 (combined) intermediate sources remain. Color3mixer is then used to calculate the fluxes for the remaining 3 sources, after which the fluxes of all components are back-calculated.

2. '3mixer': The triangle/trio method creates for all possible combinations of 3 primary component spectra a spectrum that results in the target chromaticity using color3mixer() and then optimizes the weights of each of the latter spectra such that adding them (additive mixing) results in obj_vals as close as possible to the target values.

class luxpy.toolboxes.spdbuild.**PrimConstructor**(*f=<function*   *gaussian_prim_constructor>,*
       ptypes=['peakwl', 'fwhm'], pdefs={})

get_spd(*nprim=None, wlr=[360, 830, 1]*)

Get ndarray with spds for prims.

Args:

nprim

None, optional

> If not None: generate nprim random prims (based fixed pars and bounds in pdefs)
>
> else: values for all pars should be defined in pdefs!
>
> > (nprims is determined by number of elements in pdefs[ptypes[0]])

class luxpy.toolboxes.spdbuild.**Minimizer**(*method='Nelder-Mead'*, *opts={}*, *x0=None*, *pareto=False*, *display=True*)

> **_set_defopts_and_pareto**(*pareto=None*, *x0=None*, *display=None*)
> > Set default options if not provided, as well as pareto (False: output Root-Sum-Squares of Fi in _fitnessfcn).
>
> **apply**(*fitness_fcn*, *npars*, *fitness_args_dict*, *bounds*, *verbosity=1*)
> > Run minimizer on fitness function with specified fitness_args_dict input arguments and bounds.

class luxpy.toolboxes.spdbuild.**ObjFcns**(*f=None*, *fp=[{}]*, *fw=[1]*, *ft=[0]*, *ft_tol=[0]*, *f_requires_solution_info=[False]*, *decimals=[5]*)

> **_equalize_sizes**(*x*)
> > Equalize structure of x to that of self.f for ease of looping of the objective functions in the fitness function
>
> **_calculate_fj**(*spdi*, *j=0*, *solution_info={}*)
> > Calculate objective function j for input spd.
>
> **_get_normalization_factors**()
> > Set normalization factor for F-calculation
>
> **_get_fj_output_str**(*j*, *obj_vals_ij*, *F_ij=nan*, *verbosity=1*)
> > get output string for objective function fj

class luxpy.toolboxes.spdbuild.**SpectralOptimizer**(*target=array([[1.0000e+02, 3.3333e-01, 3.3333e-01]])*, *tar_type='Yxy'*, *cspace_bwtf={}*, *nprim=4*, *wlr=[360, 830, 1]*, *cieobs='1931_2'*, *out='spds,primss,Ms,results'*, *optimizer_type='3mixer'*, *triangle_strengths_bnds=None*, *prim_constructor=<luxpy.toolboxes.spdbuild.spdoptimizer202 object>*, *prims=None*, *obj_fcn=<luxpy.toolboxes.spdbuild.spdoptimizer2020.ObjFcn object>*, *minimizer=<luxpy.toolboxes.spdbuild.spdoptimizer2020.Minimize object>*, *verbosity=1*)

> **_update_nprim_prims**(*nprim=None*, *prims=None*)
> > Update prims (and nprim).
>
> **_update_target**(*target=None*, *tar_type=None*, *cspace_bwtf=None*)
> > Update target chromaticity.
>
> **_update_prim_pars_bnds**(*nprim=None*, ***kwargs*)
> > Get and set fixed and free parameters, as well as bnds on latter for an nprim primary mixture.
>
> **_update_triangle_strengths_bnds**(*nprim=None*, *triangle_strengths_bnds=None*)
> > Update bounds of triangle_strengths for for an nprim primary mixture.
>
> **_update_bnds**(*nprim=None*, *triangle_strengths_bnds=None*, ***prim_kwargs*)
> > Update all bounds (triangle_strengths and those of free parameters of primary constructor) for an nprim primary mixture..

update(*nprim=None*, *prims=None*, *cieobs=None*, *target=None*, *tar_type=None*, *cspace_bwtf=None*, *triangle_strengths_bnds=None*, ***prim_kwargs*)

Updates all that is needed when one of the input arguments is changed.

_spd_constructor_tri(*x*)

Construct a mixture spectrum composed of n primaries using the 3mixer algorithm.

Args:

> x

>> optimization parameters, first n!/(n-3)!*3! are the strengths of
>> the triangles in the '3mixer' algorithm.

Returns:

> **spd, prims, M**

>> - spd: spectrum resulting from x
>> - spds: primary spds
>> - M: fluxes of all primaries

Notes: 1. '3mixer' - optimization algorithm: The triangle/trio method creates for all possible combinations of 3 primary component spectra a spectrum that results in the target chromaticity using color3mixer() and then optimizes the weights of each of the latter spectra such that adding them (additive mixing) results in obj_vals as close as possible to the target values.

_spd_constructor_nomixer(*x*)

Construct a mixture spectrum composed of n primaries using no mixer algorithm (just simple weighted sum of primaries).

Args:

> x

>> optimization parameters, first n are the strengths of individual primaries.

Returns:

> **spd, prims, M**

>> - spd: spectrum resulting from x
>> - spds: primary spds
>> - M: fluxes of all primaries

Notes:

> 1. 'no-mixer' - simple weighted sum of primaries.

_fitness_fcn(*x*, *out='F'*)

Fitness function that calculates closeness of solution x to target values for specified objective functions.

start(*verbosity=None*, *out=None*)

Start optimization of _fitnessfcn for n primaries using the initialized minimizer and the selected optimizer_type.

Returns variables specified in :out:

luxpy.toolboxes.spdbuild.**_extract_prim_optimization_parameters**(*x*, *nprims*, *prim_constructor_parameter_types*, *prim_constructor_parameter_defs*)

Extract the primary parameters from the optimization vector x and the prim_constructor_parameter_defs dict, for use with PrimConstructor..

luxpy.toolboxes.spdbuild.**_stack_wlr_spd**(*wlr*, *spd*)

Stack the wavelength range on top of the spd values for use with PrimConstructor.

luxpy.toolboxes.spdbuild.**_setup_wlr**(*wlr*)

Setup the wavelength range for use with PrimConstructor.

luxpy.toolboxes.spdbuild.**spd_optimizer2**(*target=array([[1.0000e+02, 3.3333e-01, 3.3333e-01]]), tar_type='Yxy', cspace_bwtf={}, n=4, wlr=[360, 830, 1], prims=None, cieobs='1931_2', out='spds,primss,Ms,results', optimizer_type='3mixer', prim_constructor=<function gaussian_prim_constructor>, prim_constructor_parameter_types=['peakwl', 'fwhm'], prim_constructor_parameter_defs={}, obj_fcn=None, obj_fcn_pars=[{}], obj_fcn_weights=[1], obj_tar_vals=[0], obj_tar_tols=[0], decimals=[5], triangle_strengths_bnds=None, minimize_method='Nelder-Mead', minimize_opts={}, x0=None, pareto=False, display=False, verbosity=1*)*

Generate a spectrum with specified white point and optimized for certain objective functions from a set of primary spectra or primary spectrum model parameters.

Args:

> **target**
>
> > np2d([100,1/3,1/3]), optional
> > ndarray with Yxy chromaticity of target.
>
> **tar_type**
>
> > 'Yxy' or str, optional
> > Specifies the input type in :target: (e.g. 'Yxy' or 'cct')
>
> **cspace_bwtf**
>
> > {}, optional
> > Backward (cspace_to_xyz) transform parameters
> > (see colortf()) to go from :tar_type: to 'Yxy').
>
> **n**
>
> > 4, optional
> > Number of primaries in light mixture.
>
> **wl**
>
> > [360,830,1], optional
> > Wavelengths used in optimization when :prims: is not an ndarray with spectral data.
>
> **cieobs**
>
> > _CIEOBS, optional
> > CIE CMF set used to calculate chromaticity values, if not provided
> > in :Yxyi:.
>
> **optimizer_type**
>
> > '3mixer', optional
> > Specifies type of chromaticity optimization
> > For help on '3mixer' algorithm, see notes below.
>
> **prims**
>
> > ndarray of predefined primary spectra.
> > If None: they are built from optimization parameters using the
> > function in :prim_constructor:

prim_constructor

> function that constructs the primaries from the optimization parameters
> Should have the form:
>
> ```
> prim_constructor(x, n, wl,
> prim_constructor_parameter_types,
> **prim_constructor_parameter_defs)
> ```

prim_constructor_parameter_types

> gaussian_prim_parameter_types ['peakwl', 'fwhm'], optional
> List with strings of the parameters used by prim_constructor() to
> calculate the primary spd. All parameters listed and that do not
> have default values (one for each prim!!!) in prim_constructor_parameters_defs
> will be optimized.

prim_constructor_parameters_defs

> {}, optional
> Dict with constructor parameters required by prim_constructor and/or
> default values for parameters that are not being optimized.
> For example: {'fwhm': 30} will keep fwhm fixed and not optimize it.

obj_fcn

> [None] or list, optional
> Function handles to objective function.

obj_fcn_weights

> [1] or list, optional.
> Weigths for each obj. fcn

obj_fcn_pars

> [{}] or list, optional
> Parameter dicts for each obj. fcn.

obj_tar_vals

> [0] or list, optional
> Target values for each objective function.

obj_tar_tols

> [0] or list, optional
> Tolerance of objective function values with target values.

decimals

> [5], optional
> Rounding decimals of objective function values.

minimize_method

> 'Nelder-Mead', optional
> Optimization method used by minimize function.
> options:
>> - 'Nelder-Mead': Nelder-Mead simplex local optimization
>>> using the luxpy.math.minimizebnd wrapper
>>> with method set to 'Nelder-Mead'.
>> - 'demo' : Differential Evolutionary Multiobjective Optimizatizer
>>> (using math.DEMO.demo_opt)
>> - 'particleswarm': Pseudo-global optimizer using particle swarms

(from pyswarm wrapper module luxpy.math.pyswarms_particleswarm)

- 'nsga_ii': Pareto multiobjective optimizer using the NSGA-II genetic algorithm

(from pymoo wrapper module luxpy.math.pymoo_nsga_ii)

- A user-defined minimization function (see _start_optimization_tri? for

info on the requirements of this function)

minimize_opts

None, optional

Dict with minimization options.

None defaults to the options depending on choice of minimize_method

- 'Nelder-Mead' : {'xtol': 1e-5, 'disp': True, 'maxiter': 1000*Nc,

'maxfev' : 1000*Nc,'fatol': 0.01}

- 'demo' : {'F': 0.5, 'CR': 0.3, 'kmax': 300, 'mu': 100, 'display': True}

- 'particleswarm' : {'iters': 100, 'n_particles': 10, 'ftol': -np.inf,

'ps_opts' : {'c1': 0.5, 'c2': 0.3, 'w':0.9}}

- 'nsga_ii' : {'n_gen' : 40, 'n_pop' : 400, 'n_offsprings' : None,

'termination' : ('n_gen' , 40), 'seed' : 1,

'ga_opts' : {'sampling' : ("real_random",{}),

'crossover' : ("real_sbx", {'prob' : 0.9, 'eta' : 15}),

'mutation' : ("real_pm", {'eta' : 20})}}

- dict with options for user-defined minimization method.

triangle_strength_bnds

(None,None)

Specifies lower- and upper-bounds for the strengths of each of the primary

combinations that will be made during the optimization using '3mixer'.

x0

None, optional

If None: a random starting value will be generated for the Nelder-Mead

minimization algorithm, else the user defined starting value will be used.

Note that it should only contain a value for each peakwl and/or fwhm that

is set to be optimized. The triangle_strengths are added automatically.

pareto

False, optional

Specifies whether the output of the fitnessfcn should be the Root-Sum-of-Squares

of all weighted objective function values or not. Individual function values are

required by true multi-objective optimizers (i.e. pareto == True).

display

True, optional

Turn native display options of minimizers on (True) or off (False).

verbosity

0, optional

If > 0: print intermediate results.

out

'spds,primss,Ms,results', optional

Determines output of function (see :returns:).

Returns:

> **returns**
>
> > spds, primss,Ms,results
> >
> > - 'spds': optimized spectrum (or spectra: for demo, particleswarm and nsga_ii minimization methods)
> > - 'primss': primary spectra of each optimized spectrum
> > - 'Ms' : ndarrays with fluxes of each primary
> > - 'results': dict with optimization results

Notes on the optimization algorithms:

> 1. '3mixer': The triangle/trio method creates for all possible combinations of 3 primary component spectra a spectrum that results in the target chromaticity using color3mixer() and then optimizes the weights of each of the latter spectra such that adding them (additive mixing) results in obj_vals as close as possible to the target values.
>
> 2. '2mixer': APRIL 2020, NOT YET IMPLEMENTED!! Pairs (odd,even) of components are selected and combined using 'pair_strength'. This process is continued until only 3 (combined) intermediate sources remain. Color3mixer is then used to calculate the fluxes for the remaining 3 sources, after which the fluxes of all components are back-calculated.

luxpy.toolboxes.spdbuild.**gaussian_prim_constructor**(*x*, *nprims*, *wlr*, *ptypes*, ***pdefs*)

> Construct a set of nprim gaussian primaries with wavelengths wlr using the input in x and in kwargs.
>
> **Args:**
>
> > **x**
> >
> > > ndarray (M x nprim) with optimization parameters.
> >
> > **nprim**
> >
> > > number of primaries
> >
> > **wlr**
> >
> > > wavelength range for which to construct a spectrum
> >
> > **prim_constructor**
> >
> > > function that constructs the primaries from the optimization parameters
> > > Should have the form:
> > >
> > > > prim_constructor(x, n, wl, ptypes, pdefs)
> >
> > **ptypes**
> >
> > > gaussian_prim_parameter_types ['peakwl', 'fwhm'], optional
> > > List with strings of the parameters used by PrimConstructor()) to
> > > calculate the primary spd. All parameters listed and that do not
> > > have default values (one for each prim!!!) in pdefs will be optimized.
> >
> > **pdefs**
> >
> > > Dict with constructor parameters required by PrimConstructor and/or
> > > default values for parameters that are not being optimized.
> > > For example: {'fwhm': [30]} will keep fwhm fixed and not optimize it.
>
> **Returns:**
>
> > **spd**
> >
> > > ndarray with spectrum of nprim primaries (1st row = wavelengths)
>
> **Example on how to create constructor:**
>
> > `def gaussian_prim_constructor(x, nprims, wlr, ptypes, **pdefs):`
> >
> > `  `
> >
> > ` # Extract the primary parameters from x and pdefs:`

```
` pars = _extract_prim_optimization_parameters(x, nprims, ptypes,
pdefs)`
` `

` # Setup wavelengths:`
` wlr = _setup_wlr(wlr)`
` `

` # Conversion factor for FWHM to sigma of Gaussian:`
` fwhm_to_sig = 1/(2*(2*np.log(2))**0.5) `
` `

` # Create spectral profile function:  `
` spd =
np.exp(-0.5*((pars['peakwl']-wlr)/(pars['fwhm']*fwhm_to_sig))**2)`
` `

` # Stack wlr and spd together:  `
` return _stack_wlr_spd(wlr,spd)`
```

luxpy.toolboxes.spdbuild.**_triangle_mixer**(*Yxy_target*, *Yxyi*, *triangle_strengths*)

Calculates the fluxes of each of the primaries to realize the target chromaticity Yxy_target given the triangle_strengths.

luxpy.toolboxes.spdbuild.**_color3mixer**(*Yxyt*, *Yxy1*, *Yxy2*, *Yxy3*)

Calculate fluxes required to obtain a target chromaticity when (additively) mixing 3 light sources.

> **Args:**
>> **Yxyt**
>>
>>> ndarray with target Yxy chromaticities.
>>
>> **Yxy1**
>>
>>> ndarray with Yxy chromaticities of light sources 1.
>>
>> **Yxy2**
>>
>>> ndarray with Yxy chromaticities of light sources 2.
>>
>> **Yxy3**
>>
>>> ndarray with Yxy chromaticities of light sources 3.
>
> **Returns:**
>> **M**
>>
>>> ndarray with fluxes.
>
> **Note:** Yxyt, Yxy1, … can contain multiple rows, referring to single mixture.

4.5.4 hypspcim/

> **py**
>
>> • __init__.py
>>
>> • hyperspectral_img_simulator.py
>
> **namespace** luxpy.hypspcim

Module for hyper spectral image simulation

> **_HYPSPCIM_PATH** path to module
>
> **_HYPSPCIM_DEFAULT_IMAGE** path + filename to default image

xyz_to_rfl() approximate spectral reflectance of xyz based on k nearest neighbour interpolation of samples from a standard reflectance set.

render_image() Render image under specified light source spd.

luxpy.toolboxes.hypspcim.**render_image**(*img=None*, *spd=None*, *rfl=None*, *out='img_hyp'*, *refspd=None*, *D=None*, *cieobs='1931_2'*, *cspace='xyz'*, *cspace_tf={}*, *CSF=None*, *interp_type='nd'*, *k_neighbours=4*, *show=True*, *verbosity=0*, *show_ref_img=True*, *stack_test_ref=12*, *write_to_file=None*, *csf_based_rgb_rounding=6*)

Render image under specified light source spd.

Args:

> **img**
>
>> None or str or ndarray with float (max = 1) rgb image.
>> None load a default image.
>
> **spd**
>
>> ndarray, optional
>> Light source spectrum for rendering
>> If None: use CIE illuminant F4
>
> **rfl**
>
>> ndarray, optional
>> Reflectance set for color coordinate to rfl mapping.
>
> **out**
>
>> 'img_hyp' or str, optional
>>> (other option: 'img_ren': rendered image under :spd:)
>
> **refspd**
>
>> None, optional
>> Reference spectrum for color coordinate to rfl mapping.
>> None defaults to D65 (srgb has a D65 white point)
>
> **D**
>
>> None, optional
>> Degree of (von Kries) adaptation from spd to refspd.
>
> **cieobs**
>
>> _CIEOBS, optional
>> CMF set for calculation of xyz from spectral data.
>
> **cspace**
>
>> 'xyz', optional
>> Color space for color coordinate to rfl mapping.
>> Tip: Use linear space (e.g. 'xyz', 'Yuv',...) for (interp_type == 'nd'),
>>> and perceptually uniform space (e.g. 'ipt') for (interp_type == 'nearest')
>
> **cspace_tf**
>
>> {}, optional
>> Dict with parameters for xyz_to_cspace and cspace_to_xyz transform.
>
> **CSF**
>
>> None, optional
>> RGB camera response functions.
>> If None: input :xyz: contains raw rgb values. Override :cspace:

argument and perform estimation directly in raw rgb space!!!

interp_type

'nd', optional

Options:

- 'nd': perform n-dimensional linear interpolation using Delaunay triangulation.

- 'nearest': perform nearest neighbour interpolation.

k_neighbours

4 or int, optional

Number of nearest neighbours for reflectance spectrum interpolation.

Neighbours are found using scipy.spatial.cKDTree

show

True, optional

Show images.

verbosity

0, optional

If > 0: make a plot of the color coordinates of original and rendered image pixels.

show_ref_img

True, optional

True: shows rendered image under reference spd. False: shows

original image.

write_to_file

None, optional

None: do nothing, else: write to filename(+path) in :write_to_file:

stack_test_ref

12, optional

- 12: left (test), right (ref) format for show and imwrite

- 21: top (test), bottom (ref)

- 1: only show/write test

- 2: only show/write ref

- 0: show both, write test

csf_based_rgb_rounding

_ROUNDING, optional

Int representing the number of decimals to round the RGB values (obtained from not-None CSF input) to before applying the search algorithm.

Smaller values increase the search speed, but could cause fatal error that causes python kernel to die. If this happens increase the rounding int value.

Returns:

returns

img_hyp, img_ren,

ndarrays with float hyperspectral image and rendered images

luxpy.toolboxes.hypspcim.**xyz_to_rfl**(*xyz*, *CSF=None*, *rfl=None*, *out='rfl_est'*, *refspd=None*, *D=None*, *cieobs='1931_2'*, *cspace='xyz'*, *cspace_tf={}*, *interp_type='nd'*, *k_neighbours=4*, *verbosity=0*, *csf_based_rgb_rounding=6*)

Approximate spectral reflectance of xyz values based on nd-dimensional linear interpolation or k nearest neighbour interpolation of samples from a standard reflectance set.

Args:

> **xyz**
>
> > ndarray with xyz values of target points.
>
> **CSF**
>
> > None, optional
> >
> > RGB camera response functions.
> >
> > If None: input :xyz: contains raw rgb (float) values. Override :cspace:
> >
> > argument and perform estimation directly in raw rgb space!!!
>
> **rfl**
>
> > ndarray, optional
> >
> > Reflectance set for color coordinate to rfl mapping.
>
> **out**
>
> > 'rfl_est' or str, optional
>
> **refspd**
>
> > None, optional
> >
> > Refer ence spectrum for color coordinate to rfl mapping.
> >
> > None defaults to D65.
>
> **cieobs**
>
> > _CIEOBS, optional
> >
> > CMF set used for calculation of xyz from spectral data.
>
> **cspace**
>
> > 'xyz', optional
> >
> > Color space for color coordinate to rfl mapping.
> >
> > Tip: Use linear space (e.g. 'xyz', 'Yuv',. . .) for (interp_type == 'nd'),
> >
> > > and perceptually uniform space (e.g. 'ipt') for (interp_type == 'nearest')
>
> **cspace_tf**
>
> > {}, optional
> >
> > Dict with parameters for xyz_to_cspace and cspace_to_xyz transform.
>
> **interp_type**
>
> > 'nd', optional
> >
> > Options:
> >
> > - 'nd': perform n-dimensional linear interpolation using Delaunay triangulation.
> >
> > - 'nearest': perform nearest neighbour interpolation.
>
> **k_neighbours**
>
> > 4 or int, optional
> >
> > Number of nearest neighbours for reflectance spectrum interpolation.
> >
> > Neighbours are found using scipy.spatial.cKDTree
>
> **verbosity**
>
> > 0, optional
> >
> > If > 0: make a plot of the color coordinates of original and
> >
> > rendered image pixels.
>
> **csf_based_rgb_rounding**
>
> > _ROUNDING, optional
> >
> > Int representing the number of decimals to round the RGB values (obtained from
> >
> > not-None CSF input) to before applying the search algorithm.

Smaller values increase the search speed, but could cause fatal error that causes
python kernel to die. If this happens increase the rounding int value.

Returns:

returns

:rfl_est:

ndarrays with estimated reflectance spectra.

luxpy.toolboxes.hypspcim.**get_superresolution_hsi**(*lrhsi,   hrci,   CSF,   wl=[380,   780,
1],   csf_based_rgb_rounding=6,
interp_type='nd',   k_neighbours=4,
verbosity=0*)

Get a HighResolution HyperSpectral Image (super-resolution HSI) based on a LowResolution HSI and a High-
Resolution Color Image.

Args:

lrhsi

ndarray with float (max = 1) LowResolution HSI [m,m,L].

hrci

ndarray with float (max = 1) HighResolution HSI [M,N,3].

CSF

None, optional

ndarray with camera sensitivity functions

If None: use Nikon D700

wl

[380,780,1], optional

Wavelength range and spacing or ndarray with wavelengths of HSI image.

interp_type

'nd', optional

Options:

- 'nd': perform n-dimensional linear interpolation using Delaunay triangulation.

- 'nearest': perform nearest neighbour interpolation.

k_neighbours

4 or int, optional

Number of nearest neighbours for reflectance spectrum interpolation.

Neighbours are found using scipy.spatial.cKDTree

verbosity

0, optional

Verbosity level for sub-call to render_image().

If > 0: make a plot of the color coordinates of original and

rendered image pixels.

csf_based_rgb_rounding

_ROUNDING, optional

Int representing the number of decimals to round the RGB values (obtained from
not-None CSF input) to before applying the search algorithm.

Smaller values increase the search speed, but could cause fatal error that causes
python kernel to die. If this happens increase the rounding int value.

Returns:

hrhsi

ndarray with HighResolution HSI [M,N,L].

Procedure:

Call render_image(hrci, rfl = lrhsi_2, CSF = . . .) to estimate a hyperspectral image
from the high-resolution color image hrci with the reflectance spectra
in the low-resolution hyper-spectral image as database for the estimation.
Estimation is done in raw RGB space with the lrhsi converted using the
camera sensitivity functions in CSF.

luxpy.toolboxes.hypspcim.**hsi_to_rgb**(*hsi, spd=None, cieobs='1931_2', srgb=False, lin-ear_rgb=False, CSF=None, normalize_to_white=True, wl=[380, 780, 1]*)

Convert HyperSpectral Image to rgb.

Args:

> **hsi**
>
> > ndarray with hyperspectral image [M,N,L]
>
> **spd**
>
> > None, optional
> > ndarray with illumination spectrum
>
> **cieobs**
>
> > _CIEOBS, optional
> > CMF set to convert spectral data to xyz tristimulus values.
>
> **srgb**
>
> > False, optional
> > If False: Use xyz_to_srgb(spd_to_xyz(. . .)) to convert to srgb values
> > If True: use camera sensitivity functions.
>
> **linear_rgb**
>
> > False, optional
> > If False: use gamma = 2.4 in xyz_to_srgb, if False: use gamma = 1 and set
> > :use_linear_part: to False.
>
> **CSF**
>
> > None, optional
> > ndarray with camera sensitivity functions
> > If None: use Nikon D700
>
> **normalize_to_white**
>
> > True, optional
> > If True & CSF is not None: white-balance output rgb to a perfect white diffuser.
>
> **wl**
>
> > [380,780,1], optional
> > Wavelength range and spacing or ndarray with wavelengths of HSI image.

Returns:

> **rgb**
>
> > ndarray with rgb image [M,N,3]

luxpy.toolboxes.hypspcim.**rfl_to_rgb**(*rfl, spd=None, CSF=None, wl=None, normal-ize_to_white=True*)

Convert spectral reflectance functions (illuminated by spd) to Camera Sensitivity Functions.

Args:

> **rfl**

---

ndarray with spectral reflectance functions (1st row is wavelengths if wl is None).

spd

None, optional

ndarray with illumination spectrum

CSF

None, optional

ndarray with camera sensitivity functions

If None: use Nikon D700

normalize_to_white

True, optional

If True: white-balance output rgb to a perfect white diffuser.

Returns:

rgb

ndarray with rgb values for each spectral reflectance functions

4.5.5 dispcal/

py

- __init__.py

- displaycalibration.py

namespace luxpy.dispcal

Module for display characterization

_PATH_DATA path to package data folder

_RGB set of RGB values that work quite well for display characterization

_XYZ example set of measured XYZ values corresponding to the RGB values in _RGB

calibrate() Calculate TR parameters/lut and conversion matrices

calibration_performance() Check calibration performance (cfr. individual and average color differences for each stimulus).

rgb_to_xyz() Convert input rgb to xyz

xyz_to_rgb() Convert input xyz to rgb

DisplayCalibration() Calculate TR parameters/lut and conversion matrices and store in object.

luxpy.toolboxes.dispcal.**calibrate**(*rgbcal*, *xyzcal*, *L_type='lms'*, *tr_type='lut'*, *cieobs='1931_2'*, *nbit=8*, *cspace='lab'*, *avg=<function <lambda>>*, *ensure_increasing_lut_at_low_rgb=0.2*, *verbosity=1*, *sep=', '*, *header=None*)

Calculate TR parameters/lut and conversion matrices.

Args:

rgbcal

ndarray [Nx3] or string with filename of RGB values

rgcal must contain at least the following type of settings:

- pure R,G,B: e.g. for pure R: (R != 0) & (G==0) & (B == 0)

- white(s): R = G = B = 2**nbit-1

- gray(s): R = G = B

- black(s): R = G = B = 0

- binary colors: cyan (G = B, R = 0), yellow (G = R, B = 0), magenta (R = B, G = 0)

xyzcal

ndarray [Nx3] or string with filename of measured XYZ values for
the RGB settings in rgbcal.

L_type

'lms', optional
Type of response to use in the derivation of the Tone-Response curves.
options:

- 'lms': use cone fundamental responses: L vs R, M vs G and S vs B
    (reduces noise and generally leads to more accurate characterization)

- 'Y': use the luminance signal: Y vs R, Y vs G, Y vs B

tr_type

'lut', optional
options:

- 'lut': Derive/specify Tone-Response as a look-up-table

- 'gog': Derive/specify Tone-Response as a gain-offset-gamma function

cieobs

'1931_2', optional
CIE CMF set used to determine the XYZ tristimulus values
(needed when L_type == 'lms': determines the conversion matrix to
convert xyz to lms values)

nbit

8, optional
RGB values in nbit format (e.g. 8, 16, . . .)

cspace

color space or chromaticity diagram to calculate color differences in
when optimizing the xyz_to_rgb and rgb_to_xyz conversion matrices.

avg

lambda x: ((x**2).mean()**0.5), optional
Function used to average the color differences of the individual RGB settings
in the optimization of the xyz_to_rgb and rgb_to_xyz conversion matrices.

ensure_increasing_lut_at_low_rgb

0.2 or float (max = 1.0) or None, optional
Ensure an increasing lut by setting all values below the RGB with the maximum
zero-crossing of np.diff(lut) and RGB/RGB.max() values of
:ensure_increasing_lut_at_low_rgb:
(values of 0.2 are a good rule of thumb value)
Non-strictly increasing lut values can be caused at low RGB values due
to noise and low measurement signal.
If None: don't force lut, but keep as is.

verbosity

1, optional

> 0: print and plot optimization results

sep

' ', optional

separator in files with rgbcal and xyzcal data

header

None, optional

header specifier for files with rgbcal and xyzcal data

(see pandas.read_csv)

Returns:

M

linear rgb to xyz conversion matrix

N

xyz to linear rgb conversion matrix

tr

Tone Response function parameters or lut

xyz_black

ndarray with XYZ tristimulus values of black

xyz_white

ndarray with tristimlus values of white

luxpy.toolboxes.dispcal.**calibration_performance**(*rgb*, *xyztarget*, *M*, *N*, *tr*, *xyz_black*, *xyz_white*, *tr_type='lut'*, *cspace='lab'*, *avg=<function <lambda>>*, *rgb_is_xyz=False*, *is_verification_data=False*, *nbit=8*, *verbosity=1*, *sep=' '*, *header=None*)

Check calibration performance. Calculate DE for each stimulus.

Args:

rgb

ndarray [Nx3] or string with filename of RGB values

(or xyz values if argument rgb_to_xyz == True!)

xyztarget

ndarray [Nx3] or string with filename of target XYZ values corresponding

to the RGB settings (or the measured XYZ values, if argument rgb_to_xyz == True).

M

linear rgb to xyz conversion matrix

N

xyz to linear rgb conversion matrix

tr

Tone Response function parameters or lut

xyz_black

ndarray with XYZ tristimulus values of black

xyz_white

ndarray with tristimlus values of white

tr_type

'lut', optional

options:
- 'lut': Derive/specify Tone-Response as a look-up-table
- 'gog': Derive/specify Tone-Response as a gain-offset-gamma function

cspace

color space or chromaticity diagram to calculate color differences in.

avg

lambda x: ((x**2).mean()**0.5), optional

Function used to average the color differences of the individual RGB settings in the optimization of the xyz_to_rgb and rgb_to_xyz conversion matrices.

rgb_is_xyz

False, optional

If True: the data in argument rgb are actually measured XYZ tristimulus values and are directly compared to the target xyz.

is_verification_data

False, optional

If False: the data is assumed to be corresponding to RGB value settings used in the calibration (i.e. containing whites, blacks, grays, pure and binary mixtures)

If True: no assumptions on content of rgb, so use this settings when checking the performance for a set of measured and target xyz data different than the ones used in the actual calibration measurements.

nbit

8, optional

RGB values in nbit format (e.g. 8, 16, . . .)

verbosity

1, optional

> 0: print and plot optimization results

sep

',', optional

separator in files with rgbcal and xyzcal data

header

None, optional

header specifier for files with rgbcal and xyzcal data

(see pandas.read_csv)

   Returns:

M

linear rgb to xyz conversion matrix

N

xyz to linear rgb conversion matrix

tr

Tone Response function parameters or lut

xyz_black

ndarray with XYZ tristimulus values of black

xyz_white

ndarray with tristimlus values of white

`luxpy.toolboxes.dispcal.`**`rgb_to_xyz`**(*rgb*, *M*, *tr*, *xyz_black*, *tr_type='lut'*)

> Convert input rgb to xyz.
>
> **Args:**
>> **rgb**
>>
>>> ndarray [Nx3] with RGB values
>>
>> **M**
>>
>>> linear rgb to xyz conversion matrix
>>
>> **tr**
>>
>>> Tone Response function parameters or lut
>>
>> **xyz_black**
>>
>>> ndarray with XYZ tristimulus values of black
>>
>> **tr_type**
>>
>>> 'lut', optional
>>>
>>> Type of Tone Response in tr input argument
>>>
>>> options:
>>>> - 'lut': Tone-Response as a look-up-table
>>>> - 'gog': Tone-Response as a gain-offset-gamma function
>
> **Returns:**
>> **xyz**
>>
>>> ndarray [Nx3] of XYZ tristimulus values

`luxpy.toolboxes.dispcal.`**`xyz_to_rgb`**(*xyz*, *N*, *tr*, *xyz_black*, *tr_type='lut'*)

> Convert xyz to input rgb.
>
> **Args:**
>> **xyz**
>>
>>> ndarray [Nx3] with XYZ tristimulus values
>>
>> **N**
>>
>>> xyz to linear rgb conversion matrix
>>
>> **tr**
>>
>>> Tone Response function parameters or lut
>>
>> **xyz_black**
>>
>>> ndarray with XYZ tristimulus values of black
>>
>> **tr_type**
>>
>>> 'lut', optional
>>>
>>> Type of Tone Response in tr input argument
>>>
>>> options:
>>>> - 'lut': Tone-Response as a look-up-table
>>>> - 'gog': Tone-Response as a gain-offset-gamma function
>
> **Returns:**
>> **rgb**
>>
>>> ndarray [Nx3] of display RGB values

class `luxpy.toolboxes.dispcal.`**`DisplayCalibration`**(*rgbcal*, *xyzcal=None*, *L_type='lms'*, *cieobs='1931_2'*, *tr_type='lut'*, *nbit=8*, *cspace='lab'*, *avg=<function DisplayCalibration.<lambda>>*, *ensure_increasing_lut_at_low_rgb=0.2*, *verbosity=1*, *sep=', '*, *header=None*)

Class for display_calibration.

Args:

>   **rgbcal**
>
>>   ndarray [Nx3] or string with filename of RGB values
>>   rgcal must contain at least the following type of settings:
>>   - pure R,G,B: e.g. for pure R: (R != 0) & (G==0) & (B == 0)
>>   - white(s): R = G = B = 2**nbit-1
>>   - gray(s): R = G = B
>>   - black(s): R = G = B = 0
>>   - binary colors: cyan (G = B, R = 0), yellow (G = R, B = 0), magenta (R = B, G = 0)
>
>   **xyzcal**
>
>>   None, optional
>>   ndarray [Nx3] or string with filename of measured XYZ values for
>>   the RGB settings in rgbcal.
>>   if None: rgbcal is [Nx6] ndarray containing rgb (columns 0-2) and xyz data (columns
>>   3-5)
>
>   **L_type**
>
>>   'lms', optional
>>   Type of response to use in the derivation of the Tone-Response curves.
>>   options:
>>>   - 'lms': use cone fundamental responses: L vs R, M vs G and S vs B
>>>   (reduces noise and generally leads to more accurate characterization)
>>>   - 'Y': use the luminance signal: Y vs R, Y vs G, Y vs B
>
>   **tr_type**
>
>>   'lut', optional
>>   options:
>>>   - 'lut': Derive/specify Tone-Response as a look-up-table
>>>   - 'gog': Derive/specify Tone-Response as a gain-offset-gamma function
>
>   **cieobs**
>
>>   '1931_2', optional
>>   CIE CMF set used to determine the XYZ tristimulus values
>>   (needed when L_type == 'lms': determines the conversion matrix to
>>   convert xyz to lms values)
>
>   **nbit**
>
>>   8, optional
>>   RGB values in nbit format (e.g. 8, 16, . . .)
>
>   **cspace**
>
>>   color space or chromaticity diagram to calculate color differences in
>>   when optimizing the xyz_to_rgb and rgb_to_xyz conversion matrices.
>
>   **avg**
>
>>   lambda x: ((x**2).mean()**0.5), optional
>>   Function used to average the color differences of the individual RGB settings
>>   in the optimization of the xyz_to_rgb and rgb_to_xyz conversion matrices.
>
>   **verbosity**
>
>>   1, optional
>>   > 0: print and plot optimization results

> > **sep**
> >
> > > ',', optional
> > >
> > > separator in files with rgbcal and xyzcal data
> >
> > **header**
> >
> > > None, optional
> > >
> > > header specifier for files with rgbcal and xyzcal data
> > >
> > > (see pandas.read_csv)
>
> **Return:**
>
> > **calobject**
> >
> > > attributes are:
> > >
> > > > - M: linear rgb to xyz conversion matrix
> > > >
> > > > - N: xyz to linear rgb conversion matrix
> > > >
> > > > - TR: Tone Response function parameters or lut
> > > >
> > > > - xyz_black: ndarray with XYZ tristimulus values of black
> > > >
> > > > - xyz_white: ndarray with tristimlus values of white
> > >
> > > as well as:
> > >
> > > > - rgbcal, xyzcal, cieobs, avg, tr_type, nbit, cspace, verbosity
> > > >
> > > > - performance: dictionary with various color differences set to np.nan
> > > >
> > > > - (run calobject.performance() to fill it with actual values)

check_performance(*rgb=None*, *xyz=None*, *verbosity=None*, *sep=', '*, *header=None*, *rgb_is_xyz=False*, *is_verification_data=True*)

> Check calibration performance (if rgbcal is None: use calibration data).
>
> **Args:**
>
> > **rgb**
> >
> > > None, optional
> > >
> > > ndarray [Nx3] or string with filename of RGB values
> > >
> > > (or xyz values if argument rgb_to_xyz == True!)
> > >
> > > If None: use self.rgbcal
> >
> > **xyz**
> >
> > > None, optional
> > >
> > > ndarray [Nx3] or string with filename of target XYZ values corresponding
> > >
> > > to the RGB settings (or the measured XYZ values, if argument rgb_to_xyz ==
> > >
> > > True).
> > >
> > > If None: use self.xyzcal
> >
> > **verbosity**
> >
> > > None, optional
> > >
> > > if None: use self.verbosity
> > >
> > > if > 0: print and plot optimization results
> >
> > **sep**
> >
> > > ',', optional
> > >
> > > separator in files with rgb and xyz data
> >
> > **header**
> >
> > > None, optional
> > >
> > > header specifier for files with rgb and xyz data
> > >
> > > (see pandas.read_csv)
> >
> > **rgb_is_xyz**

> > > > False, optional
> > > >
> > > > If True: the data in argument rgb are actually measured XYZ tristimulus values
> > > > > and are directly compared to the target xyz.

> > > **is_verification_data**

> > > > False, optional
> > > >
> > > > If False: the data is assumed to be corresponding to RGB value settings used
> > > > > in the calibration (i.e. containing whites, blacks, grays, pure and binary
> > > > > mixtures)
> > > > > Performance results are stored in self.performance.
> > > >
> > > > If True: no assumptions on content of rgb, so use this settings when
> > > > > checking the performance for a set of measured and target xyz data
> > > > > different than the ones used in the actual calibration measurements.

> > **Return:**

> > > **performance**

> > > > dictionary with various color differences.

> **to_xyz**(*rgb*)
> > Convert display rgb to xyz.

> **to_rgb**(*xyz*)
> > Convert xyz to display rgb.

4.5.6 rgb2spec/

> **py**

> > > • __init__.py
> > >
> > > • smits_mitsuba.py

> > **namespace** luxpy.rgb2spec

Module for RGB to spectrum conversions

> > **_BASESPEC_SMITS** Default dict with base spectra for white, cyan, magenta, yellow, blue,
> > > green and red for each intent ('rfl' or 'spd')

> > **rgb_to_spec_smits()** Convert an array of (linearized) RGB values to a spectrum using a
> > > smits like conversion as implemented in mitsuba (July 10, 2019)

> > **convert()** Convert an array of (linearized) RGB values to a spectrum (wrapper around
> > > rgb_to_spec_smits(), future: implement other methods)

luxpy.toolboxes.rgb2spec.**convert**(*rgb, linearized_rgb=True, method='smits_mtsb', intent='rfl',
> > > > > bitdepth=8, wlr=[360.0, 830.0, 1.0], rgb2spec=None*)

> Convert an array of RGB values to a spectrum.
> **Args:**
> > **rgb**

> > > ndarray of list of rgb values
> > **linearized_rgb**

> > > True, optional
> > >
> > > If False: RGB values will be linearized using:
> > > > rgb_lin = xyz_to_srgb(srgb_to_xyz(rgb), gamma = 1, use_linear_part = False)
> > > If True: user has entered pre-linearized RGB values.
> > **method**

> 'smits_mtsb', optional
>
> Method to use for conversion:
>
> > - 'smits_mtsb': use a smits like conversion as implemented in mitsuba.

intent

> 'rfl' (or 'spd'), optional
>
> type of requested spectrum conversion .

bitdepth

> 8, optional
>
> bit depth of rgb values

wlr

> _WL3, optional
>
> desired wavelength (nm) range of spectrum.

rgb2spec

> None, optional
>
> Dict with base spectra for white, cyan, magenta, yellow, blue, green and red for each intent.
>
> If None: use _BASESPEC_SMITS.

> **Returns:**
>
> > **spec**
> >
> > > ndarray with spectrum or spectra (one for each rgb value, first row are the wavelengths)

luxpy.toolboxes.rgb2spec.**rgb_to_spec_smits**(*rgb,   intent='rfl',   linearized_rgb=True, bitdepth=8,   wlr=[360.0,   830.0,   1.0], rgb2spec=None*)

Convert an array of (linearized) RGB values to a spectrum using a Smits like conversion as implemented in Mitsuba.

Args:

> **rgb**
>
> > ndarray of list of (linearized) rgb values

linearized_rgb

> True, optional
>
> If False: RGB values will be linearized using:
>
> > rgb_lin = xyz_to_srgb(srgb_to_xyz(rgb), gamma = 1, use_linear_part = False)
>
> If True: user has entered pre-linearized RGB values.

intent

> 'rfl' (or 'spd'), optional
>
> type of requested spectrum conversion.

bitdepth

> 8, optional
>
> bit depth of rgb values

wlr

> _WL3, optional
>
> desired wavelength (nm) range of spectrum.

rgb2spec

> None, optional

Dict with base spectra for white, cyan, magenta, yellow, blue, green and red for each intent.

If None: use _BASESPEC_SMITS.

Returns:

    spec

        ndarray with spectrum or spectra (one for each rgb value, first row are the wavelengths)

4.5.7 iolidfiles/

    py

        • __init__.py

        • io_lid_files.py

    namespace luxpy.iolidfiles

Module for reading and writing IES and LDT files.

    read_lamp_data Read in light intensity distribution and other lamp data from LDT or IES files.

    Notes: 1.Only basic support. Writing is not yet implemented. 2.Reading IES files is based on Blender's ies2cycles.py 3.This was implemented to build some uv-texture maps for rendering and only tested for a few files. 4. Use at own risk. No warranties.

`luxpy.toolboxes.iolidfiles.`**`read_lamp_data`**(*datasource*, *multiplier=1.0*, *verbosity=0*, *normalize='I0'*, *only_common_keys=False*)

Read in light intensity distribution and other lamp data from LDT or IES files.

Args:

    datasource

        Filename of LID file or StringIO object or string with LID data.

    multiplier

        1.0, optional

        Scaler for candela values.

    verbosity

        0, optional

        Display messages while reading file.

    normalize

        'I0', optional

        If 'I0': normalize LID to intensity at (theta,phi) = (0,0)

        If 'max': normalize to max = 1.

    only_common_keys

        False, optional

        If True, output only common dict keys related to angles, values

        and such of LID.

        read_lid_lamp_data(?) for print of common keys and return

            empty dict with common keys.

Returns:

lid dict with IES or LDT file data. | | If LIDtype == 'ies': | dict_keys(| ['datasource', 'version', 'lamps_num', 'lumens_per_lamp', | 'candela_mult', 'v_angles_num', 'h_angles_num', 'photometric_type', | 'units_type', 'width', 'length', 'height', 'ballast_factor', | 'future_use', 'input_watts', 'v_angs', 'h_angs', 'lamp_cone_type', | 'lamp_h_type', 'candela_values', 'candela_2d', 'v_same', 'h_same', | 'intensity', 'theta', 'values', 'phi', 'map','Iv0'] |) | | If LIDtype == 'ldt': | dict_keys(| ['datasource', 'version', 'manufacturer', 'Ityp','Isym', | 'Mc', 'Dc', 'Ng', 'name', Dg', 'cct/cri', 'tflux', 'lumens_per_lamp', | 'candela_mult', 'tilt', lamps_num', | 'cangles', 'tangles','candela_values', 'candela_2d', | 'intensity', 'theta', 'values', 'phi', 'map', 'Iv0'] |)

Notes:

1. if only_common_keys: output is dictionary with keys: ['datasource', 'version', 'intensity', 'theta', 'phi', 'values', 'map', 'Iv0', 'candela_values', 'candela_2d']
2. 'theta','phi', 'values' (='candela_2d') contain the original theta angles, phi angles and normalized candelas as specified in file.
3. 'map' contains a dicionary with keys 'thetas', 'phis', 'values'. This data has been complete to full angle ranges thetas: [0,180]; phis: [0,360]
4. LDT map completion only supported for Isymm == 4 (since 31/10/2018), and Isymm == 1 (since, 02/10/2021), Map will be filled with original 'theta', 'phi' and normalized 'candela_2d' values !
5. LIDtype is checked by looking for the presence of 'TILT=' in datasource content (if True->'IES' else 'LDT')
6. IES files with TILT=INCLUDE or TILT=<filename> are not supported!

luxpy.toolboxes.iolidfiles.**get_uv_texture**(*theta*, *phi=None*, *values=None*, *input_types=('array'*, *'array')*, *method='linear'*, *theta_min=0*, *angle_res=1*, *close_phi=False*, *deg=True*, *r=1*, *show=True*, *out='values_map'*)

Create a uv-texture map. | with specified angular resolution (°) and with positive z-axis as normal. | u corresponds to phi [0° - 360°] | v corresponds to theta [0° - 180°], (or [-90° - 90°])

Args:

> **theta**
>
> > Float, int or ndarray
> >
> > Angle with positive z-axis.
> >
> > Values corresponding to 0 and 180° must be specified!
>
> **phi**
>
> > None, optional
> >
> > Float, int or ndarray
> >
> > Angle around positive z-axis starting from x-axis.
> >
> > If not None: values corresponding to 0 and 360° must be specified!
>
> **values**
>
> > None
> >
> > ndarray or mesh of values at (theta, phi) locations.
>
> **input_types**
>
> > ('array','array'), optional
> >
> > Specification of type of input of (angles,values)
>
> **method**
>
> > 'linear', optional
> >
> > Interpolation method.
> >
> > (supported scipy.interpolate.griddata methods:
> >
> > > 'nearest', 'linear', 'cubic')

theta_min

> 0, optional
>
> If 0: [0, 180]; If -90: theta range = [-90,90]

close_phi

> False, optional
>
> Make phi angles array closed (full circle).

angle_res

> 1, optional
>
> Resolution in degrees.

deg

> True, optional
>
> Type of angle input (True: degrees, False: radians).

r

> 1, optional
>
> Float, int or ndarray
>
> radius

show

> True, optional
>
> Plot results.

out

> 'values_map', optional
>
> Specifies output: "return eval(out)"

Returns:

> **returns** as specified by :out:.

luxpy.toolboxes.iolidfiles.**save_texture**(*filename*, *tex*, *bits=16*, *transpose=True*)

> Save 16 bit grayscale PNG image of uv-texture.
>
> **Args:**
>
> > **filename**
> >
> > > Filename of output image.
> >
> > **tex**
> >
> > > ndarray float uv-texture.
> >
> > **transpose**
> >
> > > True, optional
> > >
> > > If True: transpose tex (u,v) to set u as columns and v as rows
> > >
> > > in texture image.
> >
> > **Returns:**
> >
> > > **None**
> >
> > **Note:**
> >
> > Texture is rescaled to max = 1 and saved as uint16.
> >
> > –> Before using uv_map: rescale back to set 'normal' to 1.

luxpy.toolboxes.iolidfiles.**draw_lid**(*LID*, *grid_interp_method='linear'*, *theta_min=0*, *angle_res=1*, *ax=None*, *projection='2d'*, *polar_plot_Cx_planes=[0, 90]*, *use_scatter_plot=False*, *plot_colorbar=True*, *legend_on=True*, *plot_luminaire_position=True*, *plot_diagram_top=0.001*, *out='ax'*, ***plottingkwargs*)

Draw the light intensity distribution.

Args:

> **LID**
>
>> dict with IES or LDT file data.
>>
>> (obtained with iolidfiles.read_lamp_data())
>
> **grid_interp_method**
>
>> 'linear', optional
>>
>> Interpolation method for (theta,phi)-grid of normalized luminous intensity values.
>>
>> (supported scipy.interpolate.griddata methods:
>>
>>> 'nearest', 'linear', 'cubic')
>
> **theta_min**
>
>> 0, optional
>>
>> If 0: [0, 180]; If -90: theta range = [-90,90]
>
> **angle_res**
>
>> 1, optional
>>
>> Resolution in degrees.
>
> **ax**
>
>> None, optional
>>
>> If None: create new 3D-axes for plotting.
>
> **projection**
>
>> '2d', optional
>>
>> If '3d' make 3 plot
>>
>> If '2d': make polar plot(s). [not yet implemented (25/03/2021)]
>
> **polar_plot_Cx_planes**
>
>> [0,90], optional
>>
>> Plot (Cx)-(Cx+180) planes; eg. [0,90] will plot C0-C180 and C90-C270 planes in 2D polar plot.
>
> **use_scatter_plot**
>
>> False, optional
>>
>> If True: use plt.scatter for plotting intensity values in 3D plot.
>>
>> If False: use plt.plot_surface for plotting in 3D plot.
>
> **plot_colorbar**
>
>> True, optional
>>
>> Plot colorbar representing the normalized luminous intensity values in the LID 3D plot.
>
> **legend_on**
>
>> True, optional
>>
>> If True: plot legend on polar plot (no legend for 3D plot!).
>
> **plot_luminaire_position**
>
>> True, optional
>>
>> Plot the position of the luminaire (0,0,0) in the 3D graph as a red diamond.
>
> **plot_diagram_top**
>
>> 1e-3, optional
>>
>> Plot the top of the polar diagram (True).
>>
>> If None: automatic detection of non-zero intensity values in top part.

If float: automatic detection of intensity values larger than max__intensity*float in top part.

> (if smaller: don't plot top.)

out

> 'ax', optional
> string with variable to return
> default: ax handle to plot.

Returns:

> **returns**
>
> Whatever requested as determined by the string in :out:

luxpy.toolboxes.iolidfiles.**render_lid**(*LID='./data/luxpy_test_lid_file.ies', sensor_resolution=100, sensor_position=[0, -1, 0.8], sensor_n=[0, 1, -0.2], fov=(90, 90), Fd=2, luminaire_position=[0, 1.3, 2], luminaire_n=[0, 0, -1], wall_center=[0, 2, 1], wall_n=[0, -1, 0], wall_width=4, wall_height=2, wall_rho=1, floor_center=[0, 1, 0], floor_n=[0, 0, 1], floor_width=4, floor_height=2, floor_rho=1, grid_interp_method='linear', angle_res=5, theta_min=0, ax3D=None, ax2D=None, join_axes=True, legend_on=True, plot_luminaire_position=True, plot_lumiaire_rays=False, plot_luminaire_lid=True, plot_sensor_position=True, plot_sensor_pixels=True, plot_sensor_rays=False, plot_wall_edges=True, plot_wall_luminance=True, plot_wall_intersections=False, plot_floor_edges=True, plot_floor_luminance=True, plot_floor_intersections=False, out='Lv2D'*)

Render a light intensity distribution.

Args:

> **LID**
>
> > dict with IES or LDT file data or string with path/filename;
> > or String or StringIO object with IES or LDT data.
> > (dict should be obtained with iolidfiles.read_lamp_data())
>
> **sensor_resolution**
>
> > 100, optional
> > Number of sensor 'pixels' along each dimension.
>
> **sensor_position**
>
> > [0,-1,0.8], optional
> > x,y,z position of the sensor 'focal' point (is located Fd meters behind actual sensor plane)
>
> **sensor_n**
>
> > [0,1,-0.2], optional
> > Sensor plane surface normal
>
> **fov**
>
> > (90,90), optional
> > Field of view of sensor image in degrees.
>
> **Fd**

2, optional

'Focal' distance in meter. Sensor center is located Fd meter away from

:sensor_position:

luminaire_position

[0,1.3,2], optional

x,y,z position of the photometric equivalent point source

luminaire_n

[0,0,-1], optional

Orientation of lumaire LID (default points downward along z-axis away from source)

wall_center

[0,2,1], optiona

x,y,z position of the back wall

wall_n

[0,-1,0], optional

surface normal of wall

wall_width

4, optional

width of wall (m)

wall_height

2, optional

height of wall (m)

wall_rho

1, optional

Diffuse (Lambertian) reflectance of wall.

floor_center

[0,1,0], optiona

x,y,z position of the floor

floor_n

[0,0,1], optional

surface normal of floor

floor_width

4, optional

width of floor (m)

floor_height

2, optional

height of floor (m)

floor_rho

1, optional

Diffuse (Lambertian) reflectance of floor.

grid_interp_method

'linear', optional

Interpolation method for (theta,phi)-grid of normalized luminous intensity values.

(supported scipy.interpolate.griddata methods:

'nearest', 'linear', 'cubic')

theta_min

0, optional

If 0: [0, 180]; If -90: theta range = [-90,90]

Only used when generating a plot of the LID in the 3D graphs.

angle_res

1, optional

Angle resolution in degrees of LID sampling.

Only used when generating a plot of the LID in the 3D graphs.

ax3D,ax2D

None, optional

If None: create new 3D- or 2D- axes for plotting.

If join_axes == True: try and combine two axes on same figure.

If False: don't plot..

legend_on

False, optional

plot legend.

plot_luminaire_position

True, optional

Plot the position of the luminaire (0,0,0) in the graph as a red diamond.

plot_X. . .

VArious options to customize plotting. Mainly allows for plotting of additional info such as plane-ray intersection points, sensor pixels, sensor-to-plane rays, plane-to-luminaire rays, 3D plot of LID, etc.

out

'Lv2D', optional

string with variable to return

default: variable storing an grayscale image of the rendered LID.

Returns:

returns

Whatever requested as determined by the string in :out:

4.5.8 spectro/

py

- __init__.py

- spectro.py

namespace luxpy.spectro

Package for spectral measurements

Supported devices:

- JETI: specbos 1211, etc.

- OceanOptics: QEPro, QE65Pro, QE65000, USB2000, USB650,etc.

get_spd() wrapper function to measure a spectral power distribution using a spectrometer of one of the supported manufacturers.

Notes

1. For info on the input arguments of get_spd(), see help for each identically named function in each of the sub-packages.

2. The use of jeti spectrometers requires access to some dll files (delivered with this package).

3. The use of oceanoptics spectrometers requires the manual installation of pyseabreeze, as well as some other 'manual' settings. See help for oceanoptics sub-package.

luxpy.toolboxes.spectro.**init**(*manufacturer*)
    Import module for specified manufacturer. Make sure everything (drivers, external packages, ...) required is installed!

luxpy.toolboxes.spectro.**get_spd**(*manufacturer='jeti'*, *dvc=0*, *Tint=0*, *autoTint_max=None*, *close_device=True*, *out='spd'*, ***kwargs*)
    Measure a spectral power distribution using a spectrometer of one of the supported manufacturers.
    Args:
        manufacturer

            'jeti' or 'oceanoptics', optional
            Manufacturer of spectrometer (ensures the correct module is loaded).
        dvc

            0 or int or spectrometer handle, optional
            If int: function will try to initialize the spectrometer to
                obtain a handle. The int represents the device
                number in a list of all detected devices of the manufacturer.
        Tint

            0 or Float, optional
            Integration time in seconds. (if 0: find best integration time, but < autoTint_max).
        autoTint_max

            Limit Tint to this value when Tint = 0.
        close_device

            True, optional
            Close spectrometer after measurement.
            If 'dvc' not in out.split(','): always close!!!
        out

            "spd" or e.g. "spd,dvc,Errors", optional
            Requested return.
        kwargs

            For info on additional input (keyword) arguments of get_spd(),
            see help for each identically named function in each of the subpackages.
    Returns:
        spd

            ndarray with spectrum. (row 0: wavelengths, row1: values)
        dvc

            Device handle, if succesfull open (_ERROR: failure, nan: closed)

Errors

Dict with error messages.

4.5.9 sherbrooke_spectral_indices/

py

- __init__.py

- sherbrooke_spectral_indices_2013.py

namespace luxpy.sherbrooke_spectral_indices

Module for the calculation of the Melatonin Suppression Index (MSI), the Induced Photosynthesis Index (IPI) and the Star Light Index (SLI) ————————————————————————————————

spd_to_msi() calculate Melatonin Suppression Index from spectrum.

spd_to_ipi() calculate Induced Photosynthesis Index from spectrum.

spd_to_sli() calculate Star Light Index from spectrum.

References: 1. Aubé M, Roby J, Kocifaj M (2013) Evaluating Potential Spectral Impacts of Various Artificial Lights on Melatonin Suppression, Photosynthesis, and Star Visibility. PLoS ONE 8(7): e67798 https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0067798

Created on Fri Jun 11 13:46:33 2021

@author: ksmet1977 [at] gmail dot com

luxpy.toolboxes.sherbrooke_spectral_indices.**spd_to_msi**(*spd*, *force_5nm_interval=True*)

Calculate Melatonin Suppression Index from spectrum.
Args:

spd

ndarray with spectral data (first row are wavelengths)

force_5nm_interval

True, optional

If True: interpolate spd to 5nm wavelengths intervals, else: keep as in spd.

Returns:

msi

ndarray with Melatonin Suppression Index values for each input spectrum.

luxpy.toolboxes.sherbrooke_spectral_indices.**spd_to_ipi**(*spd*, *force_5nm_interval=True*)

Calculate Induced Photosynthesis Index from spectrum.
Args:

spd

ndarray with spectral data (first row are wavelengths)

force_5nm_interval

True, optional

If True: interpolate spd to 5nm wavelengths intervals, else: keep as in spd.

Returns:

msi

ndarray with Induced Photosynthesis Index values for each input spectrum.

`luxpy.toolboxes.sherbrooke_spectral_indices.`**`spd_to_sli`**(*spd*,
force_5nm_interval=True)

> Calculate Star Light Index from spectrum.
>
> **Args:**
>> **spd**
>>
>>> ndarray with spectral data (first row are wavelengths)
>>
>> **force_5nm_interval**
>>
>>> True, optional
>>> If True: interpolate spd to 5nm wavelengths intervals, else: keep as in spd.
>
> **Returns:**
>> **msi**
>>
>>> ndarray with Star Light Index values for each input spectrum.

4.5.10 spectral_mismatch_and_uncertainty/

> **py**
>
>> • __init__.py
>>
>> • detector_spectral_mismatch.py
>
> **namespace** luxpy.spectral_mismatch_and_uncertainty

Toolbox for spectral mismatch and measurement uncertainty calculations

spectral_mismatch_and_uncertainty/detector_spectral_mismatch.py

> **f1prime()** Determine the f1prime spectral mismatch index.
>
> **get_spectral_mismatch_correct_factors()** Determine the spectral mismatch factors.

Reference

1. Krüger, U. et al. GENERAL V(λ) MISMATCH - INDEX HISTORY, CURRENT STATE, NEW IDEAS (TechnoTeam)

Created on Tue Aug 31 10:46:02 2021

@author: ksmet1977 [at] gmail.com

`luxpy.toolboxes.spectral_mismatch_and_uncertainty.`**`f1prime`**(*s_detector*,   *S_C='A'*,
cieobs='1931_2',
s_target_index=2,
wlr=None,   *interp_kind='linear'*,
out='f1p')

> Determine the f1prime spectral mismatch index.
>
> **Args:**
>> **s_detector**
>>
>>> ndarray with detector spectral responsivity (first row = wavelengths)
>>
>> **S_C**

'A', optional

Standard 'calibration' illuminant.

string specifying the illuminant to use from the luxpy._CIE_ILLUMINANTS dict

or ndarray with standard illuminant spectral data.

cieobs

'1931_2', optional

string with CIE standard observer color matching functions to use (from luxpy._CMF)

or ndarray with CMFs (s_target_index > 0)

or target spectral responsivity (s_target_index == 0)

(first row contains the wavelengths).

s_target_index

2, optional

if > 0: index into CMF set (1->'xbar', 2->'ybar'='Vlambda', 3->'zbar')

if == 0: cieobs is expected to contain an ndarray with the target spectral responsivity.

wlr

None, optional

Wavelength range (None, ndarray or [start, stop, spacing]).

If None: the wavelengths of the detector are used throughout.

interp_kind

'linear', optional

Interpolation type to use when interpolating function to specified wavelength range.

out

'f1p', optional

Specify requested output of function,

   e.g. 'f1p,s_rel' also outputs the normalized target spectral responsitivity.

Returns:

f1p

ndarray (vector) with f1prime values for each of the spectral responsivities in

s_detector.

luxpy.toolboxes.spectral_mismatch_and_uncertainty.**get_spectral_mismatch_correction_factors**

Determine the spectral mismatch factors.

Args:

S_Z

ndarray with spectral power distribution of measured light source (first row =

wavelengths).

s_detector

ndarray with detector spectral responsivity (first row = wavelengths)

S_C

'A', optional

Standard 'calibration' illuminant.

string specifying the illuminant to use from the luxpy._CIE_ILLUMINANTS dict

or ndarray with standard illuminant spectral data.

cieobs

'1931_2', optional

string with CIE standard observer color matching functions to use (from luxpy._CMF)

or ndarray with CMFs (s_target_index > 0)

or target spectral responsivity (s_target_index == 0)

(first row contains the wavelengths).

s_target_index

2, optional

if > 0: index into CMF set (1->'xbar', 2->'ybar'='Vlambda', 3->'zbar')

if == 0: cieobs is expected to contain an ndarray with the target spectral responsivity.

wlr

None, optional

Wavelength range (ndarray or [start, stop, spacing]).

If None: use the wavelength range of S_Z.

interp_kind

'linear', optional

Interpolation type to use when interpolating function to specified wavelength range.

out

'F', optional

Specify requested output of function,

> e.g. 'F,f1p' also outputs the f1prime spectral mismatch index.

Returns:

F

ndarray with correction factors for each of the mesured spectra (rows)

and spectral responsivities in s_detector (columns).

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX