

---

# **LuxPy Documentation**

***Release 1.5.0***

**Kevin A.G. Smet**

**May 03, 2020**



## **CONTENTS:**



- Author: K. A.G. Smet (ksmet1977 at gmail.com)
- Version: 1.5.0
- Date: May 02, 2020
- License: GPLv3





## 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](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)

## LUXPY PACKAGE STRUCTURE

### 4.1 Utils sub-package

**py**

- `__init__.py`
- `utilities.py`
- `folder_tree.py`

**namespace** `luxpy.utils`

`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>

`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 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-compatible 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 (specified by :tshape:)

**Returns:****returns**

ndarray broadcast-compatible 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->

```
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 folder 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  
successful, otherwise throws an 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

## 4.2 Math sub-package

**py**

- `__init__.py`
- `basics.py`
- `minimizebnd.py`
- `mupolymodel.py`
- `particleswarm.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 matrix, sigma<sup>-1</sup>.

**mahalanobis2()** Evaluate the squared mahalanobis distance with center mu and shape and orientation determined by sigma<sup>-1</sup>.

**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)<sup>-1</sup>' elements cik from v-format ellipse descriptor.

**cik\_to\_v()** Calculate v-format ellipse descriptor from 2x2 'covariance matrix'<sup>-1</sup> cik.

**minimizebnd()** scipy.minimize() that allows constrained parameters on unconstrained methods(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]

**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 (wrapper 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)

**particleswarm** Module with `particleswarm()` function for global minimization using particle swarms (wrapper around `pyswarms.single.GlobalBestPSO`; module is uninimported to minimize dependencies))

---

`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'])

```
luxpy.math.normalize_3x3_matrix(M, xyz0=array([[1.0, 1.0, 1.0]]))
```

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:

$A = \text{numpy.triu}(A) + \text{numpy.triu}(A).T - \text{numpy.diag}(\text{numpy.diag}(A))$

#### 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.erfinv(y)`

Inverse of the error function erf.

Computes the inverse of the error function.

In complex domain, there is no unique complex number  $w$  satisfying  $\text{erf}(w)=z$ . This indicates a true inverse function would have multi-value. When the domain restricts to the real,  $-1 < x < 1$ , there is a unique real number satisfying  $\text{erf}(\text{erfinv}(x)) = x$ .

**y** [ndarray] Argument at which to evaluate. Domain: [-1, 1]

**erfinv** [ndarray] The inverse of erf of y, element-wise

- 1) evaluating a float number

```
>>> from scipy import special
>>> special.erfinv(0.5)
0.4769362762044698
```

- 2) evaluating a ndarray

```
>>> from scipy import special
>>> y = np.linspace(-1.0, 1.0, num=10)
>>> special.erfinv(y)
array([      -inf, -0.86312307, -0.5407314 , -0.30457019, -0.0987901 ,
         0.0987901 ,  0.30457019,  0.5407314 ,  0.86312307,       inf])
```

`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 numpy.histogram is run.

Mimicks matlab hist (uses bin centers).

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

**Returns:**

**returns**

ndarray with histogram

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

Convert Cartesian 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

Input 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 Cartesian 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)<sup>-1</sup>' 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)<sup>-1</sup>

**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'<sup>-1</sup> cik

**Args:**

**cik**  
'Nx2x2' (covariance matrix)<sup>-1</sup>

**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  $[0, 2\pi]$

**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 constrained minimization problem:  $\mathbf{a}^T \mathbf{S} \mathbf{a}$ , see ref below.)

**Returns:**

**v**  
vector with ellipse parameters [Rmax,Rmin, xc,yc, theta]

**Reference:** 1. Fitzgibbon, A.W., Pilu, M., and Fischer R.B., Direct least squares fitting of ellipses, Proc. of the 13th International 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]

`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(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 ( $p \geq 2$ ) 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  $(n_i > 20).all()$  &  $(p < 6)$  &  $(k < 6)$ : then a more appropriate chi2 test is used in 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  $\text{var1}/\text{var2}$  (with  $\text{var1} > \text{var2}$ ).

**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)

## 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
    • SPD.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',
    '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 arrays 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 are no XYZ to LMS conversion matrices defined for the 1964 10°, 1931 2° Judd corrected (1951) and 1931 2° Judd-Vos corrected (1978) cmf sets.** The Hunt-Pointer-Estevéz conversion matrix of the 1931 2° is therefore used as an approximation!
3. **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°).

4. **The 1951 scotopic 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 to np.eye().

## spectrum/spectral.py

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

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

**\_S012\_DAYLIGHTPHASE** numpy.ndarray with CIE S0,S1, S2 curves for daylight phase calculation.

**\_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

**\_CRI\_REF\_TYPE** Dict with blackbody to daylight transition (mixing) ranges for various types of reference illuminants used in color rendering index calculations.

**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 numpy.array.
2. Convert spd-like data from numpy.array to pandas.dataframe and back.
3. Interpolate spectral data.
4. Normalize spectral data.

**xyzbar()** Get color matching functions.

**vlbar()** Get Vlambda function.

**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.

**blackbody()** Calculate blackbody radiator spectrum.

**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.

**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.

**\_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/](http://capbone.com/spectral-reflectance-database/)

## 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.
- 

`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, negative_values_allowed=False, extrapol_values=None)`

Interpolate / extrapolate spectral data following standard CIE15-2018.

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

Extrapolation is always done by replicate the closest known values.

**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 if automatically chosen.
- Or :kind: can be any interpolation type supported by `scipy.interpolate.interp1d` (math.interp1d if nan's are present!!)

**negative\_values\_allowed**

False, optional

If False: negative values are clipped to zero.

**extrap\_values**

None, optional

If None: use CIE recommended 'closest value' approach when extrapolating.

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

If 'ext': use normal extrapolated values by `scipy.interpolate.interp1d`

**Returns:****returns**

ndarray of interpolated spectral data.

(.shape = (number of spectra + 1, number of wavelength in `wl_new`))

```
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, norm_type=None, norm_f=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.

**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.

**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, norm\_type=None, norm\_f=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.

**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.

**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.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 ( $Y_w = 100$ ) or absolute XYZ ( $Y = \text{Luminance}$ )

**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.  $K = 683 \text{ lm/W}$  for '1931\_2' (relative == False)

or  $K = 100/\text{sum}(\text{spd} * \text{dl})$  (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  $\text{xyz} == \text{xyzw}$ , 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  $\text{xyz.shape} = (\text{rfl.shape}[0], \text{data.shape}[0], 3)$

and with  $\text{xyzw.shape} = (\text{data.shape}[0], 3)$

**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.blackbody(*cct*, *wl3*=None)

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.

**Returns:****returns**

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

**References:**

1. CIE15:2018, “Colorimetry,” CIE, Vienna, Austria, 2018.

`luxpy.spectrum.daylightlocus` (*cct*, *force\_daylight\_below4000K=False*)  
Calculates daylight chromaticity 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.

**Returns:****returns**

(ndarray of x-coordinates, ndarray of y-coordinates)

**References:**

1. CIE15:2018, “Colorimetry,” CIE, Vienna, Austria, 2018.

`luxpy.spectrum.daylightphase` (*cct*, *wl3=None*, *force\_daylight\_below4000K=False*, *verbosity=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`.

**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.

**Returns:****returns**

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

**References:**



1. CIE15:2018, “Colorimetry,” CIE, Vienna, Austria, 2018.

```
luxpy.spectrum.cri_ref(ccts, wl3=None, ref_type='ciera', mix_range=None, cieobs='1931_2',
                      norm_type=None, norm_f=None, force_daylight_below4000K=False)
```

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 weighed mean of a Planckian and Daylight Phase spectrum.

Weighthing is done as described in IES TM30:

$$\text{SPDreference} = (\text{Te}-\text{Tb})/(\text{Te}-\text{Tb})*\text{Planckian} + (\text{T}-\text{Tb})/(\text{Te}-\text{Tb})*\text{daylight}$$

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**

luxpy.\_CIEOBS, optional

Required for the normalization of the Planckian and Daylight SPDs

when calculating a ‘mixed’ reference illuminant.

**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.

**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.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 there is some non-symmetry in the peaks)
- 'fwhms\_mid\_heights' : height at the middle of the peak

`luxpy.spectrum.spd_to_indoor (spd)`

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

### 4.3.2 basics/

py

- `__init__.py`
- `cmf.py`
- `spectral.py`
- `spectral_databases.py`

namespace luxpy

`luxpy.spectrum.basics.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.basics.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.basics.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.basics.cie_interp(data, wl_new, kind=None, negative_values_allowed=False,
                                extrap_values=None)
```

Interpolate / extrapolate spectral data following standard CIE15-2018.

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

Extrapolation is always done by replicate the closest known values.

**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 if automatically chosen.

- Or :kind: can be any interpolation type supported by scipy.interpolate.interp1d (math.interp1d if nan's are present!!)

**negative\_values\_allowed**

False, optional

If False: negative values are clipped to zero.

**extrap\_values**

None, optional

If None: use CIE recommended 'closest value' approach when extrapolating.

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

If 'ext': use normal extrapolated values by scipy.interpolate.interp1d

**Returns:**

**returns**

ndarray of interpolated spectral data.

(.shape = (number of spectra + 1, number of wavelength in wl\_new))

`luxpy.spectrum.basics.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.basics.xyzbar(cieobs='1931_2', scr='dict', wl_new=None, norm_type=None,  
                               norm_f=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.

#### **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.

**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.basics.vlbar(cieobs='1931_2', scr='dict', wl_new=None, norm_type=None,
                           norm_f=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.

**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.

**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.basics.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.basics.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.basics.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.basics.blackbody(cct, wl3=None)`

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`.

**Returns:****returns**

ndarray with blackbody radiator spectrum

(:returns:[0] contains wavelengths)

**References:**

1. CIE15:2018, “Colorimetry,” CIE, Vienna, Austria, 2018.

`luxpy.spectrum.basics.daylightlocus(cct, force_daylight_below4000K=False)`

Calculates daylight chromaticity 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.

**Returns:****returns**

(ndarray of x-coordinates, ndarray of y-coordinates)

**References:**

1. CIE15:2018, “Colorimetry,” CIE, Vienna, Austria, 2018.

`luxpy.spectrum.basics.daylightphase` (*cct*, *wl3=None*, *force\_daylight\_below4000K=False*, *verbosity=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`.

**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.

**Returns:**

**returns**

ndarray with daylight phase spectrum

(`returns[0]` contains wavelengths)

**References:**

1. CIE15:2018, “Colorimetry,” CIE, Vienna, Austria, 2018.

`luxpy.spectrum.basics.cri_ref` (*ccts*, *wl3=None*, *ref\_type='ciera'*, *mix\_range=None*, *cieobs='1931\_2'*, *norm\_type=None*, *norm\_f=None*, *force\_daylight\_below4000K=False*)

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 weighed mean of a Planckian and Daylight Phase spectrum.

Weighthing is done as described in IES TM30:

$$\text{SPD}_{\text{reference}} = (\text{Te}-\text{Tb})/(\text{Te}-\text{Tb})*\text{Planckian} + (\text{T}-\text{Tb})/(\text{Te}-\text{Tb})*\text{daylight}$$

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**

luxpy.\_CIEOBS, optional

Required for the normalization of the Planckian and Daylight SPDs when calculating a ‘mixed’ reference illuminant.

**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.

**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.basics.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 there is some non-symmetry in the peaks)
- 'fwhms\_mid\_heights' : height at the middle of the peak

`luxpy.spectrum.basics.spd_to_indoor (spd)`

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

## 4.4 Color sub-package

**py**

- `__init__.py`
- `CDATA.py`

**namespace** `luxpy`

### 4.4.1 utils/

**py**

- `__init__.py`
- `plotters.py`

**namespace** `luxpy`

**Module with functions related to plotting of color data**

**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.

---

`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*=True, *BBL*=True, *D65*=False, *EEW*=False, *cctlabs*=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.

**cctlabs**

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**

None (:show: == True)

or

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 1e19 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**

None (:show: == True)

or

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

```
luxpy.color.utils.plotBB(ccts=None, cieobs='1931_2', cspace='Yuv', axh=None, cctlabs=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 1e19 K] in 100 steps on a log10 scale

**cctlabs**

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**

None (:show: == True)  
or  
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-', **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, 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`)

**kwargs**

additional keyword arguments for use with `matplotlib.pyplot`.

**Returns:****returns**

None (:show: == True)  
or  
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([[0.0, 0.0]]), *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 ( $^{\circ}$ ), 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=True, 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**

True, 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,      di-
                                                    agram_opacity=1.0,      di-
                                                    agram_lightness=0.25,
                                                    cieobs='1931_2',    cspace='Yxy',
                                                    cspace_pars={},      show=True,
                                                    axh=None,      show_grid=True,
                                                    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**

True, 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=True, 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  $(1 + \text{abs}(\text{wavelength\_height}) * \text{spdmax})$

**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**

True, 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:

#### 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_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)
- \* `xyz_to_jabz()`, `jabz_to_xyz()`: (X,Y,Z) <-> (J,z,az,bz) [Safdar et al, 2017]

## 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. 4. 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.

`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 Yu'v' chromaticity values.

**Args:**

**xyz**

ndarray with tristimulus values

**Returns:**

**Yuv**

ndarray with CIE 1976 Yu'v' chromaticity values

(Y value refers to luminance or luminance factor)

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

Convert CIE 1976 Yu'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([[100.0, 100.0, 100.0]]),  
**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([[100.0, 100.0, 100.0]]),  
**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**

**Returns:** ndarray with tristimulus values

**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

**Returns:** CMF set to use when calculating xyzw.

**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  $L^*u^*v^*$  (CIELUV) 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([[100.0,  
                                         100.0,    100.0]]), flip_axes=False,  
                                              **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!)

**Returns:**

**Ydlep**

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

```
luxpy.color.ctf.colortransforms.Ydlep_to_xyz(Ydlep,
                                              cieobs='1931_2',
                                              xyzw=array([[100.0, 100.0, 100.0]]),
                                              flip_axes=False, **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 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 Ydlep to increase speed of loop in function.  
(single xyzw with is not flipped!)

**Returns:**

**xyz**

ndarray with tristimulus values

```
luxpy.color.ctf.colortransforms.xyz_to_srgb(xyz, gamma=2.4, **kwargs)
```

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).

```
luxpy.color.ctf.colortransforms.srgb_to_xyz(rgb, gamma=2.4, **kwargs)
```

Calculates xyz from IEC:61966 sRGB values.

**Args:**

**rgb**

ndarray with srgb values (uint8).

**gamma**

2.4, optional  
compression in sRGB

**Returns:**

**xyz**

ndarray with relative tristimulus values.

`luxpy.color.ctf.colortransforms.xyz_to_jabz(xyz, **kwargs)`

Convert XYZ tristimulus values to Jz,az,bz color coordinates.

**Args:**

**xyz**

ndarray with absolute tristimulus values (Y in  $\text{cd/m}^2$ !)

**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 =  $10000 \text{ cd/m}^2$   
(note that Jz that not exactly equal 1 for this high value, but rather for  $102900 \text{ cd/m}^2$ )

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, June 2017.

`luxpy.color.ctf.colortransforms.jabz_to_xyz(jabz, **kwargs)`

Convert Jz,az,bz color coordinates to XYZ tristimulus values.

**Args:**

**jabz**

ndarray with Jz,az,bz color coordinates

**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 =  $10000 \text{ cd/m}^2$   
(note that Jz that not exactly equal 1 for this high value, but rather for  $102900 \text{ cd/m}^2$ )

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, June, 2017.

## 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
- cctduv\_ohno\_CORM2011.py

**namespace** luxpy

**cct: Module with functions related to correlated color temperature calculations**

**\_CCT\_LUT\_PATH** Folder with Look-Up-Tables (LUT) for correlated color temperature calculation followings Ohno's method.

**\_CCT\_LUT** Dict with LUTs.

**\_CCT\_LUT\_CALC** Boolean determining whether to force LUT calculation, even if the LUT can be found in ./data/cctluts/.

**calculate\_lut()** Function that calculates the LUT for the ccts stored in ./data/cctluts/cct\_lut\_cctlist.dat or given as input argument. Calculation is performed for CMF set specified in cieobs. Adds a new (temporary) field to the \_CCT\_LUT dict.

**calculate\_luts()** Function that recalculates (and overwrites) LUTs in ./data/cctluts/ for the ccts stored in ./data/cctluts/cct\_lut\_cctlist.dat or given as input argument. Calculation is performed for all CMF sets listed in \_CMF['types'].

**xyz\_to\_cct()**

Calculates CCT, Duv from XYZ

wrapper for xyz\_to\_cct\_ohno() & xyz\_to\_cct\_search()

**xyz\_to\_duv()** Calculates Duv, (CCT) from XYZ wrapper for xyz\_to\_cct\_ohno() & xyz\_to\_cct\_search()

**cct\_to\_xyz()** Calculates xyz from CCT, Duv [100 K < CCT < 1e12]

**xyz\_to\_cct\_mcamy()**

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\_HA()**

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\_ohno()**

Calculates CCT, Duv from XYZ using a LUT following:

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

**xyz\_to\_cct\_search()** Calculates CCT, Duv from XYZ using brute-force search algorithm (between 1e2 K - 1e12 K on a log scale)

**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).

---

luxpy.color.cct.**calculate\_luts** (ccts=None)

Function that recalculates (and overwrites) LUTs in ./data/cctluts/ for the ccts stored in ./data/cctluts/cct\_lut\_cctlist.dat or given as input argument. Calculation is performed for all CMF sets listed in \_CMF['types'].

**Args:**

**ccts**

ndarray or str, optional

List of ccts for which to (re-)calculate the LUTs.

If str, ccts contains path/filename.dat to list.

**Returns:**

None

**Note:** Function writes LUTs to ./data/cctluts/ folder!

```
luxpy.color.cct.xyz_to_cct(xyzw, cieobs='1931_2', out='cct', mode='lut', wl=None, rtol=1e-05,
                           atol=0.1, force_out_of_lut=True, upper_cct_max=1000000000000.0,
                           approx_cct_temp=True, fast_search=True, cct_search_list=None)
```

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

Wrapper function for use with luxpy.colortf().

**Args:****xyzw**

ndarray of tristimulus values

**cieobs**

luxpy.\_CIEOBS, optional

CMF set used to calculate xyzw.

**mode**

'lut' or 'search', optional

Determines what method to use.

**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.

**rtol**

1e-5, float, optional

Stop brute-force 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

brute-force search and with  $dCCT$  the difference between

the present and former estimates.

**atol**

0.1, optional

Stop brute-force search when cct a absolute tolerance (K) is reached.

**upper\_cct\_max**

1e12, optional  
Limit brute-force search to this cct.

**approx\_cct\_temp**

True, optional  
If True: use xyz\_to\_cct\_HA() to get a first estimate of cct to speed up search.  
Only for 'fast' code option.

**fast\_search**

True, optional  
Use fast brute-force search, i.e. xyz\_to\_cct\_search\_fast()

**cct\_search\_list**

None, optional  
list of ccts to obtain a first guess for the cct of the input xyz when HA estimation fails due to out-of-range cct or when fast\_search == False.  
None defaults to: [50,100,500,1000,2000,3000,4000,5000,6000,10000,20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12]

**force\_out\_of\_lut**

True, optional  
If True and cct is out of range of the LUT, then switch to brute-force search method, else return numpy.nan values.

**Returns:****returns**

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

```
luxpy.color.cct.xyz_to_duv(xyzw, cieobs='1931_2', out='duv', mode='lut', wl=None, rtol=1e-05, atol=0.1, force_out_of_lut=True, upper_cct_max=1000000000000.0, approx_cct_temp=True, fast_search=True, cct_search_list=None)
```

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

Wrapper function for use with luxpy.colortf().

**Args:****xyzw**

ndarray of tristimulus values

**cieobs**

luxpy.\_CIEOBS, optional  
CMF set used to calculate xyzw.

**mode**

‘lut’ or ‘search’, optional  
Determines what method to use.

**out**

‘duv’ (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.

**rtol**

1e-5, float, optional  
Stop brute-force 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  
brute-force search and with  $dCCT$  the difference between  
the present and former estimates.

**atol**

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

**upper\_cct\_max**

1e12, optional  
Limit brute-force search to this cct.

**approx\_cct\_temp**

True, optional  
If True: use `xyz_to_cct_HA()` to get a first estimate of cct to  
speed up search.  
Only for ‘fast’ code option.

**fast\_search**

True, optional  
Use fast brute-force search, i.e. `xyz_to_cct_search_fast()`

**cct\_search\_list**

None, optional  
list of ccts to obtain a first guess for the cct of the input xyz  
when HA estimation fails due to out-of-range cct or when `fast_search == False`.  
None defaults to: [50,100,500,1000,2000,3000,4000,5000,6000,10000,  
20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12]

**force\_out\_of\_lut**

True, optional  
If True and cct is out of range of the LUT, then switch to  
brute-force search method, else return `numpy.nan` values.

**Returns:**

**returns**

ndarray with:

duv: out == 'duv' (or -1)

Optional:

duv: out == 'duv' (or -1),

cct, duv: out == 'cct,duv' (or 2),

[cct,duv]: out == "[cct,duv]" (or -2)

```
luxpy.color.cct.cct_to_xyz(ccts,    duv=None,    cieobs='1931_2',    wl=None,    mode='lut',
                           out=None,    rtol=1e-05,    atol=0.1,    force_out_of_lut=True,    up-
                           per_cct_max=1000000000000.0,    approx_cct_temp=True,
                           fast_search=True, cct_search_list=None)
```

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

Finds xyzw\_estimated by minimization of:

$$F = \text{numpy.sqrt}(((100.0 * (\text{cct\_min} - \text{cct}) / (\text{cct})) ** 2.0) + (((\text{duv\_min} - \text{duv}) / (\text{duv})) ** 2.0))$$

with cct,duv the input values and cct\_min, duv\_min calculated using luxpy.xyz\_to\_cct(xyzw\_estimated,...).

#### Args:

##### **ccts**

ndarray of cct values

##### **duv**

None or ndarray of duv values, optional

Note that duv can be supplied together with cct values in :ccts: as ndarray with shape (N,2)

##### **cieobs**

luxpy.\_CIEOBS, optional

CMF set used to calculate xyzw.

##### **mode**

'lut' or 'search', optional

Determines what method to use.

##### **out**

None (or 1), optional

If not None or 1: output a ndarray that contains estimated xyz and minimization results: (cct\_min, duv\_min, F\_min (objective fcn value))

##### **wl**

None, optional

Wavelengths used when calculating Planckian radiators.

##### **rtol**



1e-5, float, optional

Stop brute-force 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 brute-force search and with  $dCCT$  the difference between the present and former estimates.

#### **atol**

0.1, optional

Stop brute-force search when cct a absolute tolerance (K) is reached.

#### **upper\_cct\_max**

1e12, optional

Limit brute-force search to this cct.

#### **approx\_cct\_temp**

True, optional

If True: use `xyz_to_cct_HA()` to get a first estimate of cct to speed up search.

Only for 'fast' code option.

#### **fast\_search**

True, optional

Use fast brute-force search, i.e. `xyz_to_cct_search_fast()`

#### **cct\_search\_list**

None, optional

list of ccts to obtain a first guess for the cct of the input xyz

when HA estimation fails due to out-of-range cct or when `fast_search == False`.

None defaults to: [50,100,500,1000,2000,3000,4000,5000,6000,10000,20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12]

#### **force\_out\_of\_lut**

True, optional

If True and cct is out of range of the LUT, then switch to brute-force search method, else return `numpy.nan` values.

#### **Returns:**

##### **returns**

ndarray with estimated XYZ tristimulus values

**Note:** If `duv` is not supplied (`:ccts::shape` is (N,1) and `:duv:` is None), source is assumed to be on the Planckian locus.

`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_ohno(xyzw,      cieobs='1931_2',      out='cct',      wl=None,
                                rtol=1e-05,      atol=0.1,      force_out_of_lut=True,      up-
                                per_cct_max=1000000000000.0,      approx_cct_temp=True,
                                cct_search_list=None, fast_search=True)
```

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

**Args:**

**xyzw**

ndarray of tristimulus values

**cieobs**

luxpy.\_CIEOBS, optional  
CMF set used to calculate xyzw.

**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.

**rtol**

1e-5, float, optional  
Stop brute-force 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  
brute-force search and with  $dCCT$  the difference between  
the present and former estimates.

**atol**

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

**upper\_cct\_max**

1e12, optional  
Limit brute-force search to this cct.

**approx\_cct\_temp**

True, optional  
If True: use `xyz_to_cct_HA()` to get a first estimate of cct to  
speed up search.  
Only for 'fast' code option.

**fast\_search**

True, optional  
Use fast brute-force search, i.e. `xyz_to_cct_search_fast()`

**cct\_search\_list**

None, optional

list of ccts to obtain a first guess for the cct of the input xyz  
 when HA estimation fails due to out-of-range cct.  
 None defaults to: [50,100,500,1000,2000,3000,4000,5000,6000,10000,  
 20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12]

#### **force\_out\_of\_lut**

True, optional  
 If True and cct is out of range of the LUT, then switch to  
 brute-force search method, else return numpy.nan values.

#### **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:** LUTs are stored in ./data/cctluts/

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

```
luxpy.color.cct.xyz_to_cct_search(xyzw, cieobs='1931_2', out='cct', wl=None, rtol=1e-05, atol=0.1, upper_cct_max=100000000000.0, approx_cct_temp=True, fast=True, cct_search_list=None)
```

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv(distance above (> 0) or below (< 0) the Planckian locus) by a brute-force search.

Wrapper around xyz\_to\_cct\_search\_fast() and xyz\_to\_cct\_search\_fast()

#### **Args:**

##### **xyzw**

ndarray of tristimulus values

##### **cieobs**

luxpy.\_CIEOBS, optional  
 CMF set used to calculate xyzw.

##### **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.

##### **rtol**

1e-5, float, optional  
 Stop brute-force 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  
 brute-force search and with dCCT the difference between  
 the present and former estimates.

##### **atol**

0.1, optional

Stop brute-force search when cct a absolute tolerance (K) is reached.

**upper\_cct\_max**

1e12, optional

Limit brute-force search to this cct.

**cct\_search\_list**

None, optional

list of ccts to obtain a first guess for the cct of the input xyz.

None defaults to: [50,100,500,1000,2000,3000,4000,5000,6000,10000,  
20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12]

Only for 'robust' code option.

**approx\_cct\_temp**

True, optional

If True: use xyz\_to\_cct\_HA() to get a first estimate of cct to  
speed up search.

Only for 'fast' code option.

**fast**

True, optional

Use fast brute-force search, i.e. xyz\_to\_cct\_search\_fast()

**cct\_search\_list**

None, optional

list of ccts to obtain a first guess for the cct of the input xyz

when HA estimation fails due to out-of-range cct or when fast == False.

None defaults to: [50,100,500,1000,2000,3000,4000,5000,6000,10000,  
20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12]

**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)

**Notes:** 1. This function is more accurate, but slower than xyz\_to\_cct\_ohno! Note that cct must be between 50 K - 1e12 K (very large cct take a long time!!!)

```
luxpy.color.cct.xyz_to_cct_search_fast(xyzw,          cieobs='1931_2',          out='cct',  
                                       wl=None,        rtol=1e-05,        atol=0.1,        up-  
                                       per_cct_max=1000000000000.0,        ap-  
                                       prox_cct_temp=True, cct_search_list=None)
```

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv(distance above (> 0) or below (< 0) the Planckian locus) by a brute-force search.

The algorithm uses an approximate cct\_temp (HA approx., see xyz\_to\_cct\_HA)

as starting point or uses the middle of the allowed cct-range

(1e2 K - 1e12 K, higher causes overflow) on a log-scale, then constructs

a 4-step section of the blackbody (Planckian) locus on which to find the minimum distance to the 1960 uv chromaticity of the test source. If HA fails then another approximate starting point is found by generating the uv chromaticity values of a set blackbody radiators spread across the locus in a 50 K to 1e12 K range (larger CCT's cause instability of the chromaticity points due to floating point errors), looking for the closest blackbody radiator and then calculating the mean of the two surrounding ones. The default cct list is [50,100,500,1000,2000,3000,4000,5000,6000,10000,20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12].

**Args:****xyzw**

ndarray of tristimulus values

**cieobs**

luxpy.\_CIEOBS, optional  
CMF set used to calculate xyzw.

**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.

**rtol**

1e-5, float, optional  
Stop brute-force 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 brute-force search and with  $dCCT$  the difference between the present and former estimates.

**atol**

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

**upper\_cct\_max**

1e12, optional  
Limit brute-force search to this cct.

**approx\_cct\_temp**

True, optional  
If True: use `xyz_to_cct_HA()` to get a first estimate of cct to speed up search.

**cct\_search\_list**

None, optional  
list of ccts to obtain a first guess for the cct of the input xyz

when HA estimation fails due to out-of-range cct.

None defaults to: [50,100,500,1000,2000,3000,4000,5000,6000,10000,  
20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12]

**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)

**Notes:** This program is more accurate, but slower than xyz\_to\_cct\_ohno! Note that cct must be between 1e3 K - 1e20 K (very large cct take a long time!!!)

```
luxpy.color.cct.xyz_to_cct_search_robust(xyzw,          cieobs='1931_2',          out='cct',  
                                         wl=None,      rtol=1e-05,      atol=0.1,      up-  
                                         per_cct_max=1000000000000.0,  
                                         cct_search_list=None)
```

Convert XYZ tristimulus values to correlated color temperature (CCT) and Duv(distance above (> 0) or below (< 0) the Planckian locus) by a brute-force search.

The algorithm uses an approximate cct\_temp as starting point then constructs, a 4-step section of the blackbody (Planckian) locus on which to find the minimum distance to the 1960 uv chromaticity of the test source. The approximate starting point is found by generating the uv chromaticity values of a set blackbody radiators spread across the locus in a 50 K to 1e12 K range (larger CCT's cause instability of the chromaticity points due to floating point errors), looking for the closest blackbody radiator and then calculating the mean of the two surrounding ones. The default cct list is [50,100,500,1000,2000,3000,4000,5000,6000,10000, 20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12].

**Args:****xyzw**

ndarray of tristimulus values

**cieobs**

luxpy.\_CIEOBS, optional

CMF set used to calculate xyzw.

**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.

**rtol**

1e-5, float, optional  
 Stop brute-force 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  
 brute-force search and with  $dCCT$  the difference between  
 the present and former estimates.

**atol**

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

**upper\_cct\_max**

1e12, optional  
 Limit brute-force search to this cct.

**cct\_search\_list**

None, optional  
 list of ccts to obtain a first guess for the cct of the input xyz.  
 None defaults to: [50,100,500,1000,2000,3000,4000,5000,6000,10000,  
 20000,50000,1e5,1e6, 1e7, 1e8,1e9, 1e10, 1e11, 1e12]

**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)

**Notes:** 1. This function is more accurate, but slower than `xyz_to_cct_ohno`! Note that cct must be between 50 K - 1e12 K (very large cct take a long time!!!)

`luxpy.color.cct.xyz_to_cct_HA(xyzw, verbosity=1)`

Convert XYZ tristimulus values to correlated color temperature (CCT).

**Args:**

**xyzw**

ndarray of tristimulus values

**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](#)

**Notes:** According to paper small error from 3000 - 800 000 K, but a test with Planckians showed errors up to 20% around 500 000 K;  $e > 0.05$  for  $T > 200\,000$ ,  $e > 0.1$  for  $T > 300\,000$ , ...

`luxpy.color.cct.xyz_to_cct_mcamy(xyzw)`

Convert XYZ tristimulus values to correlated color temperature (CCT) using the mcamy approximation.

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

**Args:**

**xyzw**

ndarray of tristimulus values

**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_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 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-Estevéz: 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 Appearance 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, Süssstrunk S. Repairing gamut problems in CIECAM02: A progress report. *Color Res Appl* 2008;33(5), 424–426.
- ‘cat02-jiang’: cat02 modified to solve yb-problem + 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. Ssstrunk, 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:

$$D = F * (1 - (1/3.6) * \text{numpy.exp}((-La - 42)/92))$$

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:

$$D = 0.08 * \text{numpy.log10}(La1 + La0) + 0.76 - 0.45 * (La1 - La0) / (La1 + La0)$$

If 'smet2017': kwargs should contain 'xyzw' and 'Dmax'

(see Smet2017\_D for more details).

If "'? user defined", then D is calculated by:

$$D = \text{ndarray}(\text{eval}(:\text{Dtype}:))$$

**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, cieobs='1964_10')`

Calculate the degree of adaptation based on chromaticity following Smet et al. (2017)

**Args:**

xyzw

ndarray with white point data

**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~50°) of 760 cd/m<sup>2</sup>))

**cieobs**

'1964\_10', optional

CMF set used in deriving model in cited paper.

**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, catmode='1>0>2', 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)

**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.

**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()`

**mcats**

[‘cat02’], optional

List[str] or List[ndarray] of sensor space matrices for each condition pair. If `len(:mcats) == 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

#### 4.4.5 cam/

**py**

- `__init__.py`
- `colorappearancemodels.py`
- `cam_02_X.py`
- `cam15u`
- `sww2016.py`
- `cam18sl.py`

**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`, `cam16`, `ciecam97s`, `cam15u`, `cam18sl`

**\_SURROUND\_PARAMETERS** database of surround param. `c`, `Nc`, `F` and `FLL` for `ciecam02`, `cam16`, `ciecam97s` and `cam15u`.

**\_NAKA\_RUSHTON\_PARAMETERS**

database with parameters (`n`, `sig`, scaling and noise) for the Naka-Rushton function:

$$NK(x) = \text{sign}(x) * \text{scaling} * ((\text{abs}(x)**n) / ((\text{abs}(x)**n) + (\text{sig**n}))) + \text{noise}$$

**\_CAM\_02\_X\_UCS\_PARAMETERS**

database with parameters specifying the conversion from `ciecam02/cam16` to:

`cam[x]ucs` (uniform color space),

cam[x]lcd (large color diff.),  
cam[x]scd (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\_TYPE** Default CAM type str specifier.

**\_CAM\_DEFAULT\_MCAT** Default MCAT specifier.

**\_CAM\_02\_X\_DEFAULT\_CONDITIONS** Default CAM model parameters for model in  
cam.\_CAM\_DEFAULT\_TYPE

**\_CAM\_AXES** dict with list[str,str,str] containing axis labels of defined cspaces.

**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.

**cam\_structure\_ciecam02\_cam16()**

basic structure of ciecam02 and cam16 models.

Has ‘forward’ (xyz → color attributes) and ‘inverse’ (color attributes → xyz) modes.

**ciecam02()**

calculates ciecam02 output

(wrapper for cam\_structure\_ciecam02\_cam16 with specifics of ciecam02):

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

(wrapper for cam\_structure\_ciecam02\_cam16 with specifics of cam16):

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.

**camucs\_structure()** basic structure to go to ucs, lcd and scd color spaces (forward + inverse available)

**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., p. n/a–n/a.

**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.

#### **wrappers**

‘xyz\_to\_jabM\_ciecam02’, ‘jabM\_ciecam02\_to\_xyz’,  
‘xyz\_to\_jabC\_ciecam02’, ‘jabC\_ciecam02\_to\_xyz’,  
‘xyz\_to\_jabM\_cam16’, ‘jabM\_cam16\_to\_xyz’,  
‘xyz\_to\_jabC\_cam16’, ‘jabC\_cam16\_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\_qabW\_cam18sl’, ‘qabW\_cam18sl\_to\_xyz’,  
‘xyz\_to\_qabM\_cam18sl’, ‘qabM\_cam18sl\_to\_xyz’,  
‘xyz\_to\_qabS\_cam18sl’, ‘qabS\_cam18sl\_to\_xyz’,

`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_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.hue_quadrature(h, unique_hue_data=None)`

Get hue quadrature H from h.

**Args:**

**h**

float or ndarray [(N,) or (N,1)] with hue data in degrees (!).

**unique\_hue data**

None or str or dict, optional

- None:  $H = h$ .
- str: CAM specifier that gets parameters from `.cam._UNIQUE_HUE_DATA` (For supported models, see `.cam._UNIQUE_HUE_DATA['models']`)
- dict: user specified unique hue data (see `luxpy.cam._UNIQUE_HUE_DATA` for expected structure)

**Returns:**

**H**

ndarray of Hue quadrature value(s).

`luxpy.color.cam.naka_rushton(data, sig=2.0, n=0.73, scaling=1.0, noise=0.0, cam=None, direction='forward')`

Apply a Naka-Rushton response compression (n) and an adaptive shift (sig).

$$NK(x) = \text{sign}(x) * \text{scaling} * ((\text{abs}(x)**n) / ((\text{abs}(x)**n) + (\text{sig**n}))) + \text{noise}$$

**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.

**cam**

None or str, optional

Use NK parameters values specific to the color appearance model.

See `.cam._NAKA_RUSHTON_PARAMETERS['models']` for supported types.

**direction**

'forward' or 'inverse', optional

Perform either  $NK(x)$  or  $NK(x)**(-1)$ .

**Returns:****returns**

float or ndarray with NK-(de)compressed input :x:

```
luxpy.color.cam.ciecam02 (data, xyzw=array([[100.0, 100.0, 100.0]]), mcat='cat02', Yw=None, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, direction='forward', outin='J,aM,bM', yellowbluepurplecorrect=False)
```

Convert between XYZ tristimulus values and ciecam02 color appearance correlates.

Wrapper for `luxpy.cam.cam_structure_ciecam02_cam16()` designed specifically for `camtype = 'ciecam02'`.

**Args:****data**

ndarray with input tristimulus values or input color appearance correlates

Can be of shape:  $(N [, xM], x 3)$ , whereby

N refers to samples, M to light sources.

**xyzw**

`_CAM_02_X_DEFAULT_WHITE_POINT` or ndarray with tristimulus values of white point(s), optional

Can be multiple by specifying a  $M \times 3$  ndarray, instead of  $1 \times 3$ .

**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 assumed that the relative Y-tristimulus value in xyzw represents the luminance factor Yw.

**mcat**

'cat02' or str or ndarray, optional

Specifies CAT sensor space.

- None defaults to the one native to the camtype (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

**condition**

luxpy.cam.\_CAM\_02\_X\_DEFAULT\_CONDITIONS, optional

Dict with condition parameters, D, La, surround ([c,Nc,F]), Yb

Can be user defined, but dict must have same structure.

**direction**

'forward' or 'inverse', optional

- 'forward': xyz -> ciecam02

- 'inverse': ciecam02 -> xyz

(input data must be:

(J or Q, aM, bM) or

(J or Q, aC,bC) or

(J or Q, aS, bS) !!)

**outin**

'J,aM,bM' or str, optional

Str specifying the type of

input (:direction: == 'inverse') and

output (:direction: == 'forward')

**yellowbluepurplecorrect**

True or False, optional

Correct for yellow-blue and purple problems in ciecam02

(Is not used in cam16 because cat16 solves issues)

**Returns:****returns**

ndarray with color appearance correlates (:direction: == 'forward')

or

XYZ tristimulus values (:direction: == 'inverse')

**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.cam16 (data, xyzw=array([[100.0, 100.0, 100.0]]), mcat='cat16', Yw=None, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, direction='forward', outin='J,aM,bM')
```

Convert between XYZ tristimulus values and cam16 color appearance correlates.

Wrapper for `luxpy.cam.cam_structure_ciecam02_cam16()` designed specifically for `camtype = 'cam16'`.

#### Args:

##### **data**

ndarray with input tristimulus values or input color appearance correlates

Can be of shape: (N [, xM], x 3), whereby

N refers to samples, M to light sources.

##### **xyzw**

`_CAM_02_X_DEFAULT_WHITE_POINT` or ndarray with tristimulus values

of white point(s), optional

Can be multiple by specifying a Mx3 ndarray, instead of 1x3.

##### **Yw**

None, optional

Luminance factor of white point.

If None: xyz (in data) and xyzw are entered as relative tristimulus values (normalized to  $Y_w = 100$ ).

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  $Y_{wr} = 100$ ).

Yw can be < 100 for e.g. paper as white point. If Yw is None, it assumed that the relative Y-tristimulus value in xyzw represents the luminance factor Yw.

##### **mcat**

'cat16' or str or ndarray, optional

Specifies CAT sensor space.

- None defaults back to 'cat02!'.  
(others e.g. 'cat02-bs', 'cat02-jiang',  
all trying to correct gamut problems of original cat02 matrix)

- str: see `luxpy.cat._MCATS.keys()` for options  
(details on type, `?luxpy.cat`)

- ndarray: matrix with sensor primaries

##### **condition**

`luxpy.cam._CAM_02_X_DEFAULT_CONDITIONS`, optional

Dict with condition parameters, D, La, surround ([c,Nc,F]), Yb  
Can be user defined, but dict must have same structure.

**direction**

‘forward’ or ‘inverse’, optional  
- ‘forward’: xyz -> cam16  
- ‘inverse’: cam16 -> xyz  
(input data must be:  
(J or Q, aM, bM) or  
(J or Q, aC, bC) or  
(J or Q, aS, bS) !!)

**outin**

‘J,aM,bM’ or str, optional  
Str specifying the type of  
input (:direction: == ‘inverse’) and  
output (:direction: == ‘forward’)

**Returns:****returns**

ndarray with color appearance correlates (:direction: == ‘forward’)  
or  
XYZ tristimulus values (:direction: == ‘inverse’)

**References:**

..[1] 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.

```
luxpy.color.cam.cam02ucs (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None, conditions={'D':
1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, di-
rection='forward', ucstype='ucs', yellowbluepurplecorrect=False,
mcat='cat02')
```

Convert between XYZ tristimulus values and cam02ucs type color appearance correlates.

Wrapper for luxpy.cam.camucs\_structure() specifically designed for ‘ciecam02’ + ‘ucs’

**Args:****data**

ndarray with input tristimulus values or input color appearance correlates  
Can be of shape: (N [, xM], x 3), whereby  
N refers to samples, M to light sources.

**xyzw**

\_CAM\_02\_X\_DEFAULT\_WHITE\_POINT or ndarray with tristimulus values  
of white point(s), optional  
Can be multiple by specifying a Mx3 ndarray, instead of 1x3.

**Yw**

None, optional  
Luminance factor of white point.

If None: xyz (in data) and xyzw are entered as relative tristimulus values (normalized to  $Y_w = 100$ ).

If not None: input tristimulus are absolute and  $Y_w$  is used to rescale the absolute values to relative ones (relative to a reference perfect white diffuser with  $Y_{wr} = 100$ ).

$Y_w$  can be  $< 100$  for e.g. paper as white point. If  $Y_w$  is None, it assumed that the relative Y-tristimulus value in xyzw represents the luminance factor  $Y_w$ .

#### **mcats**

'cat02' or str or ndarray, optional

Specifies CAT sensor space.

- None defaults to the one native to the camtype (others e.g. 'cat02-bs', 'cat02-jiang', all trying to correct gamut problems of original cat02 matrix)
- str: see `luxpy.cat._MCATS.keys()` for options (details on type, `?luxpy.cat`)
- ndarray: matrix with sensor primaries

#### **condition**

`luxpy.cam._CAM_02_X_DEFAULT_CONDITIONS`, optional

Dict with condition parameters, D, La, surround ([c,Nc,F]), Yb

Can be user defined, but dict must have same structure.

#### **direction**

'forward' or 'inverse', optional

- 'forward': xyz -> cam02ucs

- 'inverse': cam02ucs -> xyz

(input data must be:

(J or Q, aM, bM) or

(J or Q, aC,bC) or

(J or Q, aS, bS) !!)

#### **outin**

'J,aM,bM' or str, optional

Str specifying the type of

input (:direction: == 'inverse') and

output (:direction: == 'forward')

#### **yellowbluepurplecorrect**

True or False, optional

Correct for yellow-blue and purple problems in ciecam02 (Is not used in cam16 because cat16 solves issues)

#### **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

**Returns:****returns**

ndarray with color appearance correlates (:direction: == ‘forward’)

or

XYZ tristimulus values (:direction: == ‘inverse’)

**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.cam16ucs (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None, conditions={'D':
1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, direc-
tion='forward', ucstype='ucs', mcat='cat16')
```

Convert between XYZ tristimulus values and cam16ucs type color appearance correlates.

Wrapper for luxpy.cam.camucs\_structure() specifically designed for ‘cam16’ + ‘ucs’

**Args:****data**

ndarray with input tristimulus values or input color appearance correlates

Can be of shape: (N [, xM], x 3), whereby

N refers to samples, M to light sources.

**xyzw**

\_CAM\_02\_X\_DEFAULT\_WHITE\_POINT or ndarray with tristimulus values  
of white point(s), optional

Can be multiple by specifying a Mx3 ndarray, instead of 1x3.

**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  
assumed that the relative Y-tristimulus value in xyzw represents  
the luminance factor Yw. .

**mcat**

‘cat16’ or str or ndarray, optional

Specifies CAT sensor space.

- 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

**condition**

luxpy.cam.\_CAM\_02\_X\_DEFAULT\_CONDITIONS, optional  
Dict with condition parameters, D, La, surround ([c,Nc,F]), Yb  
Can be user defined, but dict must have same structure.

**direction**

‘forward’ or ‘inverse’, optional  
-‘forward’: xyz -> cam16ucs  
-‘inverse’: cam16ucs -> xyz  
(input data must be:  
(J or Q, aM, bM) or  
(J or Q, aC,bC) or  
(J or Q, aS, bS) !!)

**outin**

‘J,aM,bM’ or str, optional  
Str specifying the type of  
input (:direction: == ‘inverse’) and  
output (:direction: == ‘forward’)

**yellowbluepurplecorrect**

True or False, optional  
Correct for yellow-blue and purple problems in ciecam02 (Is not used in cam16  
because cat16 solves issues)

**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

**Returns:****returns**

ndarray with color appearance correlates (:direction: == ‘forward’)  
or  
XYZ tristimulus values (:direction: == ‘inverse’)

**References:** 1. M. R. Luo, G. Cui, and C. Li, (2006), “Uniform colour spaces based on CIECAM02 colour appearance model,” *Color Res. Appl.*, vol. 31, no. 4, pp. 320–330. 2. 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.**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 -&gt; cam15u

- 'inverse': cam15u -&gt; 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.cam_sww16(data, dataw=None, Yb=20.0, Lw=400.0, Ccwb=None, relative=True, inputtpe='xyz', direction='forward', parameters=None, cieobs='2006_10', match_to_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,  $Y_w = 100$ )

**Lw**

400.0, optional  
Luminance ( $\text{cd/m}^2$ ) 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 ( $Y_w = 100$ )

**parameters**

None or str or dict, optional  
Dict with model parameters.

- None: defaults to luxpy.cam.\_CAM\_SWW\_2016\_PARAMETERS['JOSA']
- str: '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\_to\_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.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<sup>2</sup>) 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 throughout, 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_jabM_ciecam02(data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None,
conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, yellowbluepurplecor-
rect=None, 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([[100.0, 100.0, 100.0]]), Yw=None,
conditions={'D': 1.0, 'Dtype': None, 'La': 100.0,
'Yb': 20.0, 'surround': 'avg'}, yellowbluepurplecor-
rect=None, 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([[100.0, 100.0, 100.0]]), Yw=None,
conditions={'D': 1.0, 'Dtype': None, 'La': 100.0,
'Yb': 20.0, 'surround': 'avg'}, yellowbluepurplecor-
rect=None, 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([[100.0, 100.0, 100.0]]), Yw=None,
conditions={'D': 1.0, 'Dtype': None, 'La': 100.0,
'Yb': 20.0, 'surround': 'avg'}, yellowbluepurplecor-
rect=None, 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.xyz_to_jabM_cam16 (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None, con-
ditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'sur-
round': 'avg'}, mcat='cat16', **kwargs)
```

Wrapper function for cam16 forward mode with J,aM,bM output.

For help on parameter details: ?luxpy.cam.cam16

```
luxpy.color.cam.jabM_cam16_to_xyz (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None, con-
ditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'sur-
round': 'avg'}, mcat='cat16', **kwargs)
```

Wrapper function for cam16 inverse mode with J,aM,bM input.

For help on parameter details: ?luxpy.cam.cam16

```
luxpy.color.cam.xyz_to_jabC_cam16 (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None, con-
ditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'sur-
round': 'avg'}, mcat='cat16', **kwargs)
```

Wrapper function for cam16 forward mode with J,aC,bC output.

For help on parameter details: ?luxpy.cam.cam16

```
luxpy.color.cam.jabC_cam16_to_xyz (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, mcat='cat16', **kwargs)
```

Wrapper function for cam16 inverse mode with J,aC,bC input.

For help on parameter details: ?luxpy.cam.cam16

```
luxpy.color.cam.xyz_to_jab_cam02ucs (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, 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([[100.0, 100.0, 100.0]]), Yw=None, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, 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([[100.0, 100.0, 100.0]]), Yw=None, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, 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([[100.0, 100.0, 100.0]]), Yw=None, conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, 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([[100.0, 100.0, 100.0]]), Yw=None,
                                     conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb':
                                     20.0, 'surround': 'avg'}, 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([[100.0, 100.0, 100.0]]), Yw=None,
                                     conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb':
                                     20.0, 'surround': 'avg'}, 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.xyz_to_jab_cam16ucs (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None,
                                     conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb':
                                     20.0, 'surround': 'avg'}, mcat='cat16', **kwargs)
```

Wrapper function for cam16ucs forward mode with J,aM,bM output and ucstype = 'ucs'.

For help on parameter details: ?luxpy.cam.cam16ucs

```
luxpy.color.cam.jab_cam16ucs_to_xyz (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None,
                                     conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb':
                                     20.0, 'surround': 'avg'}, mcat='cat16', **kwargs)
```

Wrapper function for cam16ucs inverse mode with J,aM,bM input and ucstype = 'ucs'.

For help on parameter details: ?luxpy.cam.cam16ucs

```
luxpy.color.cam.xyz_to_jab_cam16lcd (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None,
                                     conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb':
                                     20.0, 'surround': 'avg'}, mcat='cat16', **kwargs)
```

Wrapper function for cam16ucs forward mode with J,aM,bM output and ucstype = 'lcd'.

For help on parameter details: ?luxpy.cam.cam16ucs

```
luxpy.color.cam.jab_cam16lcd_to_xyz (data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None,
                                     conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb':
                                     20.0, 'surround': 'avg'}, mcat='cat16', **kwargs)
```

Wrapper function for cam16ucs inverse mode with J,aM,bM input and ucstype = 'lcd'.

For help on parameter details: ?luxpy.cam.cam16ucs

```
luxpy.color.cam.xyz_to_jab_cam16scd(data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None,
                                     conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb':
                                     20.0, 'surround': 'avg'}, mcat='cat16', **kwargs)
    Wrapper function for cam16ucs forward mode with J,aM,bM output and ucstype = 'scd'.
```

For help on parameter details: ?luxpy.cam.cam16ucs

```
luxpy.color.cam.jab_cam16scd_to_xyz(data, xyzw=array([[100.0, 100.0, 100.0]]), Yw=None,
                                     conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb':
                                     20.0, 'surround': 'avg'}, mcat='cat16', **kwargs)
    Wrapper function for cam16ucs inverse mode with J,aM,bM input and ucstype = 'scd'.
```

For help on parameter details: ?luxpy.cam.cam16ucs

```
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.xyz_to_lab_cam_sww16(xyz, xyzw=None, Yb=20.0, Lw=400.0, Ccwb=None,
                                       relative=True, parameters=None, 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=None, 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.xyz_to_qabW_cam18sl(xyz, xyzb=None, Lb=[100], fov=10.0, parameters=None,
                                     **kwargs)
    Wrapper function for cam18sl forward mode with 'Q,aW,bW' output. (Note that 'Q,aW,bW' is a Cartesian
    a,b-coordinate system centered at (1,0))
```



For help on parameter details: ?luxpy.cam.cam18sl

```
luxpy.color.cam.qabW_cam18sl_to_xyz(qab, xyzb=None, Lb=[100], fov=10.0, parameters=None, **kwargs)
```

Wrapper function for cam18sl inverse mode with ‘Q,aW,bW’ input. (Note that ‘Q,aW,bW’ is a Cartesian a,b-coordinate system centered at (1,0))

For help on parameter details: ?luxpy.cam.cam18sl

```
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

#### 4.4.6 deltaE/

py

- \_\_init\_\_.py
- colordifferences.py
- discriminationellipses.py
- frielellipses.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\_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, DType='jab', avg=None, avg_axis=0, out='DEi', xyzwt=array([[100.0, 100.0, 100.0]]), xyzwr=array([[100.0, 100.0, 100.0]]), 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', yel-lowbluepurplecorrect=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\_02\_X\_DEFAULT\_TYPE, optional

Str specifier for CAM type to use, options: 'ciecam02' or 'cam16'.

### 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

Weights 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_ctype(xyzt, xyzr, dtype='xyz', tf='Yuv', DEtype='jab', avg=None,
                             avg_axis=0, out='DEi', xyzwt=None, xyzwr=None, fwftf={},
                             fwftr={}, 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 :fwtftr:

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 :fwtftr:

or else to default of cspace.

#### **tf**

\_CSPACE, optional

Color space to use for color difference calculation.

#### **fwtftr**

{}, optional

Dict with parameters for forward transform from xyz to cspace for test data.

#### **fwtfrr**

{}, optional

Dict with parameters for forward transform  
from xyz to cspace for reference data.

#### **KLCH**

None, optional

Weights 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\_02\_X\_DEFAULT\_TYPE, optional

Str specifier for CAM type to use, options: 'ciecam02' or 'cam16'.

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([[100.0, 0.33333, 0.33333]]), etype='fmc2', nsteps=10,  
                                              k_neighbours=3, average_cik=True,  
                                              Y=None)
```

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?).

**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).

**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

```
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_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([[100.0, 0.33333, 0.33333]]), 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  $Y_{xy}$ .

**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 if  $Y = 10.69$ !; 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  $T^2$  test on the centers  $Y_{xy1}$  and  $Y_{xy2}$  and their associated FMC-1/2 discrimination ellipses.

**Args:**

**$Y_{xy1}$ ,  $Y_{xy2}$**

2D ndarrays with  $[Y,]_{x,y}$  coordinate centers.

If  $Y_{xy}.shape[-1] == 2$ :  $Y$  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:  $Y_i = 10.69$  and overrides values in  $Y_{xyi}$ .

**ellipsoid**

True, optional

If True: return ellipsoids, else return ellipses (only if  $cspace == 'Y_{xy}'$ )!

**cspace**

' $Y_{xy}$ ', optional

Return coefficients for  $Y_{xy}$ -ellipses/ellipsoids (' $Y_{xy}$ ') 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  $g_{ij}^{-1} = S_i/n_i$  is determined using the FMC-1 or FMC-2 equations (see refs. 1 & 2). 2. Calculate sum of covariance matrices:  $SIG = S1/n1 + S2/n2 = g_{ij1}^{-1} + g_{ij2}^{-1}$  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

## 4.4.7 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<sup>2</sup>), ‘L1000’ (1000 cd/m<sup>2</sup>) and ‘L2000’ (2000 cd/m<sup>2</sup>).

#### **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 chromaticity 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.

#### 4.4.8 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`
  - `ciewrappers.py`
  - `ieswrappers.py`
  - `cri2012.py`
  - `mcri.py`
  - `cqs.py`
- `/iestm30/`
  - `__init__.py`
  - `ies_tm30_metrics.py`
  - `ies_tm30_graphics.py`
- `/VFPX/`
  - `__inint__.py`
  - `vectorshiftmodel.py`
  - `pixelshiftmodel.py`
  - `VF_PX_models.py`

**namespace** luxpy.cri

**cri:** sub-package supporting 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’,
- ‘cri2012’, ‘cri2012-hl17’, ‘cri2012-hl1000’, ‘cri2012-real210’,
- ‘mcri’,
- ‘cqs-v7.5’, ‘cqs-v9.0’

**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:

$$R_{fi,a} = 100 - c_1 * DE_{i,a} \quad (c_1 = 4.6)$$

### log\_scale()

Log-based color rendering index scale from Davis & Ohno (2009):

$$R_{fi,a} = 10 * \ln(\exp((100 - c_1 * DE_{i,a})/10) + 1)$$

### psy\_scale()

Psychometric based color rendering index scale from Smet et al. (2013):

$$R_{fi,a} = 100 * (2 / (\exp(c_1 * \text{abs}(DE_{i,a})^{**}(c_2 + 1)))^{**} c_3)$$

## utils/helpers.py

**gamut\_slicer()** Slices the gamut in nhbins slices and provides normalization of test gamut to reference gamut.

**jab\_to\_rg()** Calculates gamut area index, Rg.

### jab\_to\_rhi()

Calculate hue bin measures:

Rfhi (local (hue bin) color fidelity)

Rcshi (local chroma shift)

Rhshi (local hue shift)

**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, Reshi, 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\_cri2012()  
spd\_to\_cri2012\_h117()  
spd\_to\_cri2012\_h11000()  
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

## 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/iestm30\_metrics.py**

**spd\_to\_ies\_tm30\_metrics()** Calculates IES TM30 metrics from spectral data.

**iestm30/iestm30\_graphics.py**

**plot\_cri\_graphics()** Plot graphical information on color rendition properties.

**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:

$$R_{fi,a} = 100 - c_1 * DE_{i,a} \quad (c_1 = 4.6)$$

**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):

$$R_{fi,a} = 10 * \ln(\exp((100 - c_1 * DE_{i,a})/10) + 1).$$
**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=[0.018181818181818, 1.5, 2.0],
                           scale_max=100.0)
```

Psychometric based color rendering index scale from CRI2012:

$$R_{fi,a} = 100 * (2 / (\exp(c_1 * \text{abs}(DE_{i,a}))^{**}(c_2) + 1))) ** c_3.$$
**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.gamut_slicer(jab_test, jab_ref, out='jabt,jabr', nhbins=None, start_hue=0.0,
                             normalize_gamut=True, normalized_chroma_ref=100,
                             close_gamut=False)
```

Slices the gamut in hue bins.

**Args:**



**jab\_test**

ndarray with Cartesian color coordinates (e.g. Jab) of the samples under the test SPD

**jab\_ref**

ndarray with Cartesian color coordinates (e.g. Jab)  
of the samples under the reference SPD

**out**

'jabt,jabr' or str, optional  
Specifies which variables to output as ndarray

**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.

**start\_hue**

0.0 or float, optional  
Hue angle to start bin slicing

**normalize\_gamut**

True or False, optional  
True normalizes the gamut of test to that of ref.  
(perfect agreement results in circle).

**normalized\_chroma\_ref**

100.0 or float, optional  
Controls the size (chroma/radius) of the normalization circle/gamut.

**close\_gamut**

False or True, optional  
True appends the first jab coordinates to the end of the output  
(for plotting closed gamuts)

**Returns:****returns**

ndarray with average jabt,jabr of each hue bin.  
(.shape = (number of hue bins, 3))

(or outputs whatever is specified in :out:)

```
luxpy.color.cri.jab_to_rg(jabt, jabr, max_scale=100, ordered_and_sliced=False, nhbins=None,
                          start_hue=0.0, normalize_gamut=True, normalized_chroma_ref=100,
                          out='Rg,jabt,jabr')
```

Calculates gamut area index, Rg.

**Args:****jabt**

ndarray with Cartesian color coordinates (e.g. Jab)  
of the samples under the test SPD

**jabr**

ndarray with Cartesian color coordinates (e.g. Jab)  
of the samples under the reference SPD

**max\_scale**

100.0, optional  
Value of Rg when Rf = max\_scale (i.e. DEavg = 0)

**ordered\_and\_sliced**

False or True, optional  
- False: Hue ordering will be done with lux.cri.gamut\_slicer().  
- True: user is responsible for hue-ordering and closing gamut  
(i.e. first element in :jab: equals the last).

**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.

**start\_hue**

0.0 or float, optional  
Hue angle to start bin slicing

**normalize\_gamut**

True or False, optional  
True normalizes the gamut of test to that of ref.  
(perfect agreement results in circle).

**normalized\_chroma\_ref**

100.0 or float, optional  
Controls the size (chroma/radius) of the normalization circle/gamut

**out**

'Rg,jabt,jabr' or str, optional  
Specifies which variables to output as ndarray

**Returns:****Rg**

float or ndarray with gamut area indices Rg.

`luxpy.color.cri.jab_to_rhi(jabt, jabr, DEi, cri_type='ies-tm30', start_hue=None, nhbins=None,  
scale_factor=None, scale_fcn=None, use_bin_avg_DEi=True)`

Calculate hue bin measures: Rfhi, Rcshi and Rhshi.

Rfhi: local (hue bin) color fidelity

Rcshi: local chroma shift

Rhshi: local hue shift

(See IES TM30)

**Args:****jabt**

ndarray with jab coordinates under test SPD

**jabr**

ndarray with jab coordinates under reference SPD

**DEi**

ndarray with DEi (from gamut\_slicer()).

**use\_bin\_avg\_DEi**

True, optional

Note that following IES-TM30 DEi 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 DEi (False).

**nhbins**

int, number of hue bins to slice gamut  
(None use the one specified in :cri\_type: dict).

**start\_hue**

float (°), hue at which to start slicing

**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 Rfhi, Rcshi and Rhshi

**References:** 1. IES TM30, Method for Evaluating Light Source Color Rendition. New York, NY: The Illuminating Engineering Society of North America.

`luxpy.color.cri.jab_to_DEi(jabt, jabr, out='DEi', avg=None)`

Calculates color differences (~fidelity), DEi, of Jab input.

**Args:****jabt**

ndarray with Cartesian color coordinates (e.g. Jab)  
of the samples under the test SPD

**jabr**

ndarray with Cartesian color coordinates (e.g. Jab)  
of the samples under the reference SPD

**avg**

None, optional

If None: don't calculate average, else: avg must be function handle

**out**

'DEi' or str, optional

Specifies requested output (e.g. 'DEi,DEa')

**Returns:**

**returns**

float or ndarray with DEi for :out: 'DEi'

Other output is also possible by changing the :out: str value.

```
luxpy.color.cri.spd_to_DEi (SPD, 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:**

**SPD**

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 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 radiations,  
'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.spd_to_rg (SPD, 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)
```

Calculates the color gamut index, Rg, of spectral data.

**Args:****SPD**

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 radiations,  
'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.

#### 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  $\rightarrow (100 - \text{scale\_factor} * \text{DEi})$ ,
    - \* `log_scale`  $\rightarrow$  (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 specied  
(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'.

#### Returns:

##### returns

float or ndarray with Rg for :out: 'Rg'

Other output is also possible by changing the :out: str value.

**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_cri (SPD, cri_type='ies-tm30', out='Rf', wl=None, sample-
                           set=None, ref_type=None, cieobs=None, avg=None,
                           scale=None, opt_scale_factor=False, cspace=None, catf=None,
                           cri_specific_pars=None, rg_pars=None)
```

Calculates the color rendering fidelity index, Rf, of spectral data.

#### Args:

##### SPD

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 radiations,  
      '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.

#### 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 specied  
(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**

True or False, optional

True: optimize scaling-factor, else do nothing and use value of scaling-factor in :scale: dict.

**Returns:****returns**

float or ndarray with Rf for :out: 'Rf'

Other output is also possible by changing the :out: str value.

**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.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-tm30-18')`

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\_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 (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\_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 (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\_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 (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\_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_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\_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_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\_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_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)`

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  $L_a = (Y_b/100)*(E/\pi)$  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  $L_a$  ( $\text{cd/m}^2$ ).

**Yb**

20.0, optional  
Luminance factor of background. (used when calculating  $L_a$  from E)  
If None, E contains  $L_a$  ( $\text{cd/m}^2$ ).

**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

**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  $Q_a$  ( $Q_{ai}$ ) or  $Q_f$  ( $Q_{fi}$ ) or  $Q_p$  ( $Q_{pi}$ ) 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.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, axtype='polar', ax=None,
                             force_CVG_layout=False)
```

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.

**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.

**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, axtype='polar', ax=None,  
                                         force_CVG_layout=False)
```

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.

**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.

**Returns:****returns**

gcf(), gca(), list with rgb colors for hue bins (for use in other plotting fncs)

```
luxpy.color.cri.spd_to_ies_tm30_metrics(SPD, cri_type=None, hbins=16, start_hue=0.0,
                                         scalef=100, vf_model_type='M6',
                                         vf_pcolorshift={'Cref': 40, 'href': array([3.7835,
3.3161, 2.8272, 1.9093, 5.2787, 4.3081,
0.37762, 6.2055, 1.4564, 0.88927]), '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:****data**

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 themselves 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**

dict with color rendering data:

- 'SPD' : ndarray test SPDs
- 'bjabt' : ndarray with binned jab data under test SPDs
- 'bjabr' : ndarray with binned jab data under reference SPDs
- 'cct' : ndarray with CCT of test SPD
- 'duv' : ndarray with distance to blackbody locus of test SPD
- 'Rf' : ndarray with general color fidelity indices
- 'Rg' : ndarray with gamut area indices
- 'Rfi' : ndarray with specific color fidelity indices
- 'Rfhi' : ndarray with local (hue binned) fidelity indices
- 'Rcshi' : ndarray with local chroma shifts indices
- 'Rhshi' : ndarray with local hue shifts indices
- 'Rt' : ndarray with general metamerism uncertainty index Rt
- 'Rti' : ndarray with specific metamerism uncertainty indices Rti
- 'Rfhi\_vf' : ndarray with local (hue binned) fidelity indices  
obtained from VF model predictions at color space  
pixel coordinates
- 'Rcshi\_vf' : ndarray with local chroma shifts indices  
(same as above)
- 'Rhshi\_vf' : ndarray with local hue shifts indices  
(same as above)

```
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([3.7835, 3.3161,  
                                   2.8272, 1.9093, 5.2787, 4.3081, 0.37762, 6.2055, 1.4564,  
                                   0.88927]), '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 graphical information on color rendition properties.

**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**

False or 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 metamerism 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 themselves 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.

**Returns:****returns**

(data,  
[plt.gca(),ax\_spd, ax\_CVG, ax\_locC, ax\_locH, ax\_VF],  
cmap )

:data: dict with color rendering data  
with keys:

- 'SPD' : ndarray test SPDs
- 'bjabt': ndarray with binned jab data under test SPDs
- 'bjabr': ndarray with binned jab data under reference SPDs
- 'cct' : ndarray with CCT of test SPD
- 'duv' : ndarray with distance to blackbody locus of test SPD
- 'Rf' : ndarray with general color fidelity indices
- 'Rg' : ndarray with gamut area indices
- 'Rfi' : ndarray with specific color fidelity indices
- 'Rfhi' : ndarray with local (hue binned) fidelity indices
- 'Reshi': ndarray with local chroma shifts indices
- 'Rhshi': ndarray with local hue shifts indices
- 'Rt' : ndarray with general metamer uncertainty index Rt
- 'Rti' : ndarray with specific metamer uncertainty indices Rti
- 'Rfhi\_vf' : ndarray with local (hue binned) fidelity indices  
obtained from VF model predictions at color space

pixel coordinates

- 'Rcshi\_vf': ndarray with local chroma shifts indices  
(same as above)
- 'Rhshi\_vf': ndarray with local hue shifts indices  
(same as above)

: [...]: list with handles to current figure and 5 axes.

: cmap: list with rgb colors for hue bins (for use in other plotting fcns)

#### 4.4.9 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  
Nddarray specifying the a-coordinates at which to apply the model.

**bxr**

`np.arange(-_VF_MAXR, _VF_MAXR+_VF_DELTAR, _VF_DELTAR)`, optional  
Nddarray 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, 'yellowbluepurplecorrect':
None}, sampleset=None, pool=False, pcolorshift={'Cref': 40, 'href': array([0.31416,
0.94248, 1.5708, 2.1991, 2.8274, 3.4558, 4.0841,
4.7124, 5.3407, 5.969]), '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 themselves 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 + metamerism shift)
  - \* Rt (metamerism uncertainty index)
  - \* Rfi (specific color fidelity indices)
  - \* Rti (specific metamerism 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',    deter-
                                              mine_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 separately.

**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'
- 'felddata': 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([0.31416, 0.94248, 1.5708, 2.1991,
                                             2.8274, 3.4558, 4.0841, 4.7124, 5.3407,
                                             5.969]), '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 themselves 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([360.0,
361.0, 362.0, 363.0, 364.0, 365.0, 366.0, 367.0,
368.0, 369.0, 370.0, 371.0, 372.0, 373.0, 374.0,
375.0, 376.0, 377.0, 378.0, 379.0, 380.0, 381.0,
382.0, 383.0, 384.0, 385.0, 386.0, 387.0, 388.0,
389.0, 390.0, 391.0, 392.0, 393.0, 394.0, 395.0,
396.0, 397.0, 398.0, 399.0, 400.0, 401.0, 402.0,
403.0, 404.0, 405.0, 406.0, 407.0, 408.0, 409.0,
410.0, 411.0, 412.0, 413.0, 414.0, 415.0, 416.0,
417.0, 418.0, 419.0, 420.0, 421.0, 422.0, 423.0,
424.0, 425.0, 426.0, 427.0, 428.0, 429.0, 430.0,
431.0, 432.0, 433.0, 434.0, 435.0, 436.0, 437.0,
438.0, 439.0, 440.0, 441.0, 442.0, 443.0, 444.0,
445.0, 446.0, 447.0, 448.0, 449.0, 450.0, 451.0,
452.0, 453.0, 454.0, 455.0, 456.0, 457.0, 458.0,
459.0, 460.0, 461.0, 462.0, 463.0, 464.0, 465.0,
466.0, 467.0, 468.0, 469.0, 470.0, 471.0, 472.0,
473.0, 474.0, 475.0, 476.0, 477.0, 478.0, 479.0,
480.0, 481.0, 482.0, 483.0, 484.0, 485.0, 486.0,
487.0, 488.0, 489.0, 490.0, 491.0, 492.0, 493.0,
494.0, 495.0, 496.0, 497.0, 498.0, 499.0, 500.0,
501.0, 502.0, 503.0, 504.0, 505.0, 506.0, 507.0,
508.0, 509.0, 510.0, 511.0, 512.0, 513.0, 514.0,
515.0, 516.0, 517.0, 518.0, 519.0, 520.0, 521.0,
522.0, 523.0, 524.0, 525.0, 526.0, 527.0, 528.0,
529.0, 530.0, 531.0, 532.0, 533.0, 534.0, 535.0,
536.0, 537.0, 538.0, 539.0, 540.0, 541.0, 542.0,
543.0, 544.0, 545.0, 546.0, 547.0, 548.0, 549.0,
550.0, 551.0, 552.0, 553.0, 554.0, 555.0, 556.0,
557.0, 558.0, 559.0, 560.0, 561.0, 562.0, 563.0,
564.0, 565.0, 566.0, 567.0, 568.0, 569.0, 570.0,
571.0, 572.0, 573.0, 574.0, 575.0, 576.0, 577.0,
578.0, 579.0, 580.0, 581.0, 582.0, 583.0, 584.0,
585.0, 586.0, 587.0, 588.0, 589.0, 590.0, 591.0,
592.0, 593.0, 594.0, 595.0, 596.0, 597.0, 598.0,
599.0, 600.0, 601.0, 602.0, 603.0, 604.0, 605.0,
606.0, 607.0, 608.0, 609.0, 610.0, 611.0, 612.0,
613.0, 614.0, 615.0, 616.0, 617.0, 618.0, 619.0,
620.0, 621.0, 622.0, 623.0, 624.0, 625.0, 626.0,
627.0, 628.0, 629.0, 630.0, 631.0, 632.0, 633.0,
634.0, 635.0, 636.0, 637.0, 638.0, 639.0, 640.0,
641.0, 642.0, 643.0, 644.0, 645.0, 646.0, 647.0,
648.0, 649.0, 650.0, 651.0, 652.0, 653.0, 654.0,
655.0, 656.0, 657.0, 658.0, 659.0, 660.0, 661.0,
662.0, 663.0, 664.0, 665.0, 666.0, 667.0, 668.0,
669.0, 670.0, 671.0, 672.0, 673.0, 674.0, 675.0,
676.0, 677.0, 678.0, 679.0, 680.0, 681.0, 682.0,
683.0, 684.0, 685.0, 686.0, 687.0, 688.0, 689.0,
690.0, 691.0, 692.0, 693.0, 694.0, 695.0, 696.0,
697.0, 698.0, 699.0, 700.0, 701.0, 702.0, 703.0,
704.0, 705.0, 706.0, 707.0, 708.0, 709.0, 710.0,
711.0, 712.0, 713.0, 714.0, 715.0, 716.0, 717.0,
718.0, 719.0, 720.0, 721.0, 722.0, 723.0, 724.0,
725.0, 726.0, 727.0, 728.0, 729.0, 730.0, 731.0,
732.0, 733.0, 734.0, 735.0, 736.0, 737.0, 738.0,
739.0, 740.0, 741.0, 742.0, 743.0, 744.0, 745.0,
746.0, 747.0, 748.0, 749.0, 750.0, 751.0, 752.0,
753.0, 754.0, 755.0, 756.0, 757.0, 758.0, 759.0,
760.0, 761.0, 762.0, 763.0, 764.0, 765.0, 766.0,
767.0, 768.0, 769.0, 770.0, 771.0, 772.0, 773.0,
```

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)

## 4.5 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 (TN003:2015) photobiological quantities

(Eesc, Eemc, Eelc, Eez, Eer and Esc, Emc, Elc, Ez, Er)

Photore- ceptor	Photopigment (la- bel, )	Spectral ciency s()	effi- -	Quantity (-opic irradi- ance)	Q-symbol (Ee,)	Unit sym- bol
s-cone	photopsin (sc)	cyanolabe		cyanopic	Ee,sc	W.m2
m-cone	photopsin (mc)	chlorolabe		chloropic	Ee,mc	W.m2
l-cone	photopsin (lc)	erythrolabe		erythropic	Ee,lc	W.m2
ipRGC	melanopsin (z)	melanopic		melanopic	Ee,z	W.m2
rod	rhodopsin (r)	rhodopic		rhodopic	Ee,r	W.m2

CIE recommends that the -opic irradiance is determined by convolving the spectral irradiance,  $E_e(\lambda)$  (Wm<sup>2</sup>), for each wavelength, with the action spectrum,  $s(\lambda)$ , where  $s(\lambda)$  is normalized to one at its peak:

$$E_{e,\lambda} = E_e(\lambda) s(\lambda) d\lambda$$

where the corresponding units are Wm<sup>2</sup> in each case.

The equivalent luminance is calculated as:

$$E_v = K_m \int E_{e,\lambda} s(\lambda) d\lambda \quad V(\lambda) d\lambda / s(\lambda) d\lambda$$

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() 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 alpha-actinic action spectra. (stored in file:  
   './data/cie\_tn003\_2015\_SI\_action\_spectra.dat')

**spd\_to\_aopicE()** Calculate alpha-opic irradiance (Ee,) and equivalent luminance (E) values  
   for the l-cone, m-cone, s-cone, rod and iprgc () photoreceptor cells following CIE  
   technical note TN 003:2015.

**References:** 1. 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](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 Observation Index (COI) [ASNZS  
   1680.2.5-1995]

**Reference:** AS/NZS1680.2.5 (1997). INTERIOR LIGHTING PART 2.5: HOSPITAL AND MEDICAL TASKS.

`luxpy.toolboxes.photbiochem.spd_to_aopicE(sid, Ee=None, E=None, Q=None,  
   cieobs='1931_2', sid_units='W/m2',  
   out='Eeas,Eas')`

Calculate alpha-opic irradiance (Ee,) and equivalent luminance (E) values for the l-cone, m-cone, s-cone, rod and iprgc () photoreceptor cells following CIE technical note TN 003:2015.

**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:

**out**

'Eas, Eas' or str, optional

Determines values to return.

**Returns:****returns**

(Eas, Eas) with Eas and Eas resp. numpy.ndarrays with the  
-opic irradiance and equivalent illuminance values  
of all spectra in :sid: in SI-units.

(other choice can be set using :out:)

```
luxpy.toolboxes.photbiochem.spd_to_COI_ASNZS1680 (S=None, tf='lab', cieobs='1931_2',  
                                                    out='COI,cct',                               extrapo-  
                                                    late_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 AS/NZS 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, E=None, sum\_sources=False, interpolate\_sources=True*)

Calculate Circadian Stimulus (CS) and Circadian Light [LRC: Rea et al 2012].

**Args:**

**El**

ndarray, optional  
Defaults to D65  
light source spectral irradiance distribution

**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 and CLa 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 CIE15-2004, "Colorimetry").

**Returns:**

**CS**

ndarray with Circadian stimulus values

**CLa**

ndarray with Circadian Light values

**Notes:** 1. The original 2012 (Eq. 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 correction 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.

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.

## 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](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](https://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/extrapolation 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\_shift\_L**

0, optional  
Std Dev. in peak wavelength shift [nm] of L-cone.

**var\_shift\_L**

0, optional  
Std Dev. in peak wavelength shift [nm] of M-cone.

**var\_shift\_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](https://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/extrapolation 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](https://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 absorbance 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/extrapolation 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](https://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 was 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](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_shift_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, normal-
                                       ize_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/extrapolation 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\_shift\_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](https://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='indy', K=683, M=array([[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]))
```

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`
- `spdoptimizer2020.py`

**namespace** `luxpy.spdbuild/`

### Module for building and optimizing SPDs

`spdbuilder.py`

### Functions

**gaussian\_spd()** Generate Gaussian spectrum.

**butterworth\_spd()** Generate Butterworth based spectrum.

**mono\_led\_spd()** Generate monochromatic LED spectrum based on a Gaussian or butterworth profile or according to Ohno (Opt. Eng. 2005).

**spd\_builder()** Build spectrum based on Gaussians, monochromatic and/or phosphor 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.

**spd\_builder()** Build spectrum based on Gaussians, monochromatic and/or phosphor 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)

spdbuilder2020.py

This module differs from spdbuild.py in the spdoptimizer function, that 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. In contrast to spdbuild.py, it only supports the '3mixer' algorithms for calculating the mixing contributions of the primaries.

## Functions

**gaussian\_prim\_constructor** constructs a gaussian based primary set.

**\_setup\_wlrr** Setup the wavelength range for use in prim\_constructor.

**\_extract\_prim\_optimization\_parameters** Extract the primary parameters from the optimization vector x and the prim\_constructor\_parameter\_defs dict.

**\_start\_optimization\_tri** Start optimization of \_fitnessfcn for n primaries using the specified minimize\_method. (see notes in docstring on specifications for the user-defined minimization fcn)

**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.

## Notes

1. See examples below (in `spdbuiler2020.__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:

$$g = \exp(-0.5*((wl - peakwl)/fwhm)**2)$$

```
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):

$$lz = (1 + ((n*(wl - peakwl)/fwhm)**2))**(-2)$$

$$\text{with } n = 2*(2**0.5-1)**0.5$$

```
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 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:

$$g = \exp(-0.5*((wl - peakwl)/fwhm)**2)$$

Lorentzian (2nd order):

$$lz = (1 + ((n*(wl - peakwl)/fwhm)**2))**(-2)$$

$$\text{with } n = 2*(2**0.5-1)**0.5$$

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:

$$\begin{aligned} \text{phosphor\_spd} = & (\text{strength\_ph1} * \text{mono\_led\_spd}(\text{peakwl\_ph1}, \dots, \text{strength\_shoulder} = 1) \\ & + \text{strength\_ph2} * \text{mono\_led\_spd}(\text{peakwl\_ph2}, \dots, \text{strength\_shoulder} = 1)) \\ & / (\text{strength\_ph1} + \text{strength\_ph2}) \end{aligned}$$

else:

$$\text{phosphor\_spd} = (\text{strength\_ph1} * \text{mono\_led\_spd}(\text{peakwl\_ph1}, \dots, \text{strength\_shoulder} = 1))$$

+ (1-strength\_ph1)\*mono\_led\_spd(peakwl\_ph2, ..., strength\_shoulder = 1))

2)  $S = (\text{mono\_led\_spd}() + \text{strength\_ph} * (\text{phosphor\_spd} / \text{phosphor\_spd.max()})) / (1 + \text{strength\_ph})$

3)  $\text{piecewise\_fcn} = S$  for  $wl < \text{peakwl}$  and 1 for  $wl \geq \text{peakwl}$

4)  $\text{phosphor\_led\_spd} = S * \text{piecewise\_fcn}$

**Args:****peakwl**

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, optional  
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  $(1 - \text{strength\_ph1})$

`: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\_pieewise\_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_pieewise_fcn=False, ver-
                                       bosity=0, out='spd', **kwargs)
```

Build spectrum based on Gaussian, monochromatic and/or phosphor 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, optional

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.

#### **fw hm\_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  $(1 - \text{strength\_ph1})$

: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 phosphor\_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 weights (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 weights 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:**  $Y_{xyt}$ ,  $Y_{xy1}$ , ... can contain multiple rows, referring to single mixture.

`luxpy.toolboxes.spdbuild.colormixer` ( $Y_{xyt}=None$ ,  $Y_{xyi}=None$ ,  $n=4$ ,  $pair\_strengths=None$ ,  
 $source\_order=None$ )

Calculate fluxes required to obtain a target chromaticity when (additively) mixing  $N$  light sources.

**Args:**

**$Y_{xyt}$**

ndarray with target  $Y_{xy}$  chromaticities.

Defaults to equi-energy white.

**$Y_{xyi}$**

ndarray with  $Y_{xy}$  chromaticities of light sources  $i = 1$  to  $n$ .

**$n$**

4 or int, optional

Number of source components to randomly generate when  $Y_{xyi}$  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.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([[100.0, 0.33333,
0.33333]]), 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.

Weights 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. fcn for the optimized spectrum.

```
luxpy.toolboxes.spdbuild.get_optim_pars_dict (target=array([[100.0,          0.33333,
0.33333]]),          tar_type='Yxy',
cieobs='1931_2', optimizer_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], al-
low_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, trian-
gle_strengths=None, peakwl_min=[400],
peakwl_max=[700],    fwhm_min=[5],
fwhm_max=[300], 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,          al-  
                                                    low_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,          al-  
                                                    low_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([[100.0, 0.33333, 0.33333]]),
                                         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=[300], 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.

Weights 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.

```
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={},
                                         decimals=5, obj_fcn=None, obj_fcn_pars={},
                                         obj_fcn_weights=None, obj_tar_vals=None,
                                         triangle_strengths_bnds=None, minimize_method=None, minimize_opts={}, x0=None,
                                         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.

**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.

Weights 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.

options:

- 'nelder-mead': Nelder-Mead simplex local optimization  
using the luxpy.math.minimizebnd wrapper  
with method set to 'Nelder-Mead'.
- 'particleswarm': Pseudo-global optimizer using particle swarms  
(using wrapper luxpy.math.particleswarm)
- 'demo' : Differential Evolutionary Multiobjective Optimatizer  
(using math.DEMO.demo\_opt)
- 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 }
- 'particleswarm' : { 'iters': 100, 'n\_particles': 10, 'ftol': -np.inf, 'ps\_opts' : { 'c1': 0.5, 'c2': 0.3, 'w':0.9} }
- 'demo' : { 'F': 0.5, 'CR': 0.3, 'kmax': 300, 'mu': 100, 'display': True }
- 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.

#### **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 particleswarm and demo 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,
                                                    prim_constructor_parameter_types,
                                                    **prim_constructor_parameter_defs)
```

Construct a set of n gaussian primaries with wavelengths wlr using the input in x and in kwargs.

**Args:**

**x**

ndarray (M x n) 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, 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**

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.

**Returns:**

**spd**

ndarray with spectrum of nprim primaries (1st row = wavelengths)

**Example on how to create constructor:**

```
`def gaussian_prim_constructor(x, nprims, wlr,`
` prim_constructor_parameter_types,`
` **prim_constructor_parameter_defs):`
` `
` # Extract the primary parameters from x and`
` prim_constructor_parameter_defs:`
` pars = _extract_prim_optimization_parameters(x, nprims,`
` prim_constructor_parameter_types, prim_constructor_parameter_defs)`
` # setup wavelengths:`
` wlr = _setup_wlr(wlr)`
` `
` # Collect parameters from pars dict:`
` return`
` np.vstack((wlr, np.exp(-( (pars['peakwl']-wlr.T)/pars['fwhm']) **2).T)))`
```

`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._setup_wlr(wlr)
    Setup the wavelength range for use in prim_constructor.
```

```
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.
```

```
luxpy.toolboxes.spdbuild._start_optimization_tri(_fitnessfcn, n, fargs_dict,
                                                  bnds, par_opt_types, minimize_method, minimize_opts,
                                                  pareto=None, x0=None, verbosity=1, out='results')
    Start optimization of _fitnessfcn for n primaries using the specified minimize_method.
```

Notes on **minimize\_method**:

**Notes on minimize\_method:**

1. Implemented: 'particleswarm', 'demo', 'nelder-mead'
2. **if not isinstance(minimize\_method, str):**

then it should contain an optimizer function with the following interface:

```
results = minimize_method(fitnessfcn, Nparameters, args = {},
                          bounds = (lb, ub), verbosity = 1)
```

With 'results' a dictionary containing various variables related to the optimization. It **MUST** contain a key 'x\_final' containing the final optimized parameters. bnds must be [lowerbounds, upperbounds] with x-bounds ndarrays with values for each parameter.

args is an argument with a dictionary containing the values for the fitnessfcn. Pareto specifies

whether the output of the fitnessfcn should be the Root-Sum-of-Squares (True) of all weighted objective function values or not (False). Individual function values are required by true multi-objective optimizers.

```
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 primis.

**Args:**

**nprim**

None, optional

If not None: generate nprim random primis (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=[None], 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)
```

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',  tri-
angle_strengths_bnds=None,
prim_constructor=<luxpy.toolboxes.spdbuild.spdoptimizer2020.
object>,  prims=None,
obj_fcn=<luxpy.toolboxes.spdbuild.spdoptimizer2020.ObjFcn
object>,  mini-
mizer=<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  $n!/(n-3)! \cdot 3!$  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.

**`_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:

#### 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_refl()`** 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', *ref\_spd*=None, *D*=None, *cieobs*='1931\_2', *cspace*='xyz', *cspace\_tf*={}, *interp\_type*='nd', *k\_neighbours*=4, *show*=True, *verbosity*=0, *show\_ref\_img*=True, *stack\_test\_ref*=12, *write\_to\_file*=None)

Render image under specified light source spd.

**Args:****img**

None or str or ndarray with uint8 rgb image.

None load a default image.

**spd**

ndarray, optional  
Light source spectrum for rendering

**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.

**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

**Returns:****returns**

img\_hyp, img\_ren,  
ndarrays with hyperspectral image and rendered images

```
luxpy.toolboxes.hypspcim.xyz_to_rfl(xyz, rfl=None, out='rfl_est', refspd=None, D=None,
                                     cieobs='1931_2', cspace='xyz', cspace_tf={},
                                     interp_type='nd', k_neighbours=4, verbosity=0)
```

Approximate spectral reflectance of xyz based on nd-dimensional linear interpolation or k nearest neighbour interpolation of samples from a standard reflectance set.

**Args:****xyz**

ndarray with tristimulus values of target points.

**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.

**Returns:****returns**

:rfl\_est:

ndarrays with estimated reflectance spectra.

### 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>>*, *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.

**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 tristimulus 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 tristimulus 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 tristimulus 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>>*,  
*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 tristimulus 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 RGB values to a spectrum using a smits like conversion as implemented in mitsuba (July 10, 2019)

**convert()** Convert an array of RGB values to a spectrum (wrapper around `rgb_to_spec_smits()`, future: implement other methods)

```
luxpy.toolboxes.rgb2spec.convert(rgb, 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

**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.

**Returns:** If None: use `_BASESPEC_SMITS`.

**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', bitdepth=8, wlr=[360.0, 830.0, 1.0], rgb2spec=None)
```

Convert an array of RGB values to a spectrum using a Smits like conversion as implemented in Mitsuba.

**Args:**

**rgb**

ndarray of list of 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(filename, 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:****filename**

Filename of IES file.

**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 file\_ext == 'ies': | dict\_keys( | ['filename', '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 file\_ext == 'ldt': | dict\_keys( | ['filename', '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'] | )

### 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.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

|

luxpy.color, ??  
luxpy.color.cam, ??  
luxpy.color.cat, ??  
luxpy.color.cct, ??  
luxpy.color.cri, ??  
luxpy.color.cri.VFPX, ??  
luxpy.color.ctf.colortf, ??  
luxpy.color.ctf.colortransforms, ??  
luxpy.color.deltaE, ??  
luxpy.color.utils, ??  
luxpy.color.whiteness, ??  
luxpy.math, ??  
luxpy.math.DEMO, ??  
luxpy.math.vec3, ??  
luxpy.spectrum, ??  
luxpy.spectrum.basics, ??  
luxpy.toolboxes.dispcal, ??  
luxpy.toolboxes.hypspcim, ??  
luxpy.toolboxes.indvcmf, ??  
luxpy.toolboxes.iolidfiles, ??  
luxpy.toolboxes.photbiochem, ??  
luxpy.toolboxes.rgb2spec, ??  
luxpy.toolboxes.spdbuild, ??  
luxpy.toolboxes.spectro, ??  
luxpy.utils, ??