LuxPy Documentation

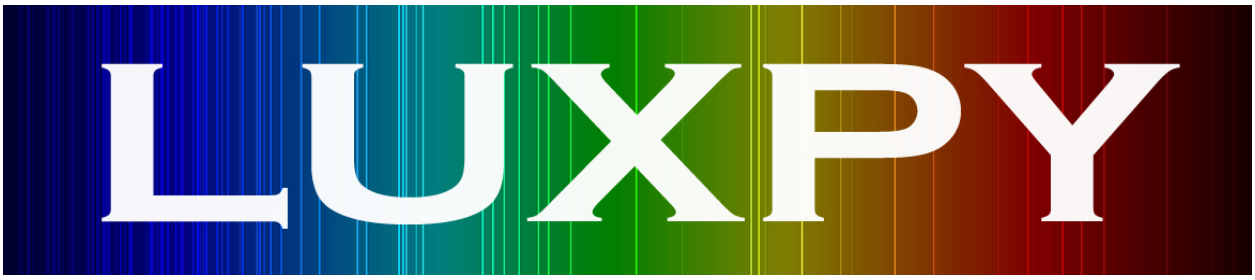
Release 1.5.0

Kevin A.G. Smet

May 02, 2020

CONTENTS:

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



CHAPTER

ONE

LICENSE: GPLV3

CHAPTER

TWO

INSTALLATION

2.1 Install luxpy

1. **Install miniconda**
    - download the installer from: https://conda.io/miniconda.html or https://repo.continuum.io/miniconda/)
    - e.g. https://repo.continuum.io/miniconda/Miniconda3-latest-Windows-x86_64.exe
    - Make sure 'conda.exe' can be found on the windows system path, if necessary do a manual add.
2. Create a virtual environment with full anaconda distribution by typing the following at the commandline:

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

3. Activate the virtual environment:

    ```
    >> activate py36
    ```

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

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

5a. Install luxpy package from pypi:

```
>> pip install luxpy
```

5b. Install luxpy package from anaconda:

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

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

2.2 Use of LuxPy package in Spyder IDE

6. Install spyder in py36 environment:

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

7. Run spyder

```
>> spyder
```

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

```
import luxpy as lx
```

2.3 Use of LuxPy package in Jupyter notebook

6. Install jupyter in py36 environment:

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

7. Start jupyter notebook:

```
>> jupyter notebook
```

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

9. To import LuxPy package type:

```
import luxpy as lx
```

CHAPTER

THREE

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)

CHAPTER

FOUR

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.**odict**

alias of collections.OrderedDict

class luxpy.utils.**Axes3D**(*fig*, *rect=None*, **args*, *azim=-60*, *elev=30*, *zscale=None*, *sharez=None*, *proj_type='persp'*, ***kwargs*)

3D axes object.

name = '3d'

_shared_z_axes = <matplotlib.cbook.Grouper object>

set_axis_off()

Turn the x- and y-axis off.

This affects the axis lines, ticks, ticklabels, grid and axis labels.

set_axis_on()

Turn the x- and y-axis on.

This affects the axis lines, ticks, ticklabels, grid and axis labels.

have_units()

Return *True* if units are set on the *x*, *y*, or *z* axes

convert_zunits(*z*)

For artists in an axes, if the zaxis has units support, convert *z* using zaxis unit type

New in version 1.2.1.

_process_unit_info(*xdata=None*, *ydata=None*, *zdata=None*, *kwargs=None*)
Look for unit *kwargs* and update the axis instances as necessary

set_top_view()

_init_axis()
Init 3D axes; overrides creation of regular X/Y axes

get_zaxis()
Return the `ZAxis` (*~.axis3d.Axis*) instance.

_get_axis_list()

unit_cube(*vals=None*)

tunit_cube(*vals=None*, *M=None*)

tunit_edges(*vals=None*, *M=None*)

draw(*renderer*)
Draw everything (plot lines, axes, labels)

get_axis_position()

_on_units_changed(*scalex=False*, *scaley=False*, *scalez=False*)
Callback for processing changes to axis units.

Currently forces updates of data limits and view limits.

update_datalim(*xys*, ***kwargs*)
Extend the *~.Axes.dataLim* BBox to include the given points.

If no data is set currently, the BBox will ignore its limits and set the bound to be the bounds of the xydata (*xys*). Otherwise, it will compute the bounds of the union of its current data and the data in *xys*.

xys [2D array-like] The points to include in the data limits BBox. This can be either a list of (x, y) tuples or a Nx2 array.

updatex, updatey [bool, optional, default *True*] Whether to update the x/y limits.

get_autoscale_on()
Get whether autoscaling is applied for all axes on plot commands

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

get_autoscalez_on()
Get whether autoscaling for the z-axis is applied on plot commands

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

set_autoscale_on(*b*)
Set whether autoscaling is applied on plot commands

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

b : bool

set_autoscalez_on(*b*)
Set whether autoscaling for the z-axis is applied on plot commands

New in version 1.1.0.

b : bool

set_zmargin(*m*)
Set padding of Z data limits prior to autoscaling.

m times the data interval will be added to each end of that interval before it is used in autoscaling.

accepts: float in range 0 to 1

New in version 1.1.0.

margins (**margins*, *x=None*, *y=None*, *z=None*, *tight=True*)
Convenience method to set or retrieve autoscaling margins.

signatures:: margins()

returns xmargin, ymargin, zmargin

```
margins(margin)

margins(xmargin, ymargin, zmargin)

margins(x=xmargin, y=ymargin, z=zmargin)

margins(..., tight=False)
```

All forms above set the xmargin, ymargin and zmargin parameters. All keyword parameters are optional. A single positional argument specifies xmargin, ymargin and zmargin. Passing both positional and keyword arguments for xmargin, ymargin, and/or zmargin is invalid.

The *tight* parameter is passed to `autoscale_view()`, which is executed after a margin is changed; the default here is *True*, on the assumption that when margins are specified, no additional padding to match tick marks is usually desired. Setting *tight* to *None* will preserve the previous setting.

Specifying any margin changes only the autoscaling; for example, if *xmargin* is not None, then *xmargin* times the X data interval will be added to each end of that interval before it is used in autoscaling.

New in version 1.1.0.

autoscale (*enable=True*, *axis='both'*, *tight=None*)
Convenience method for simple axis view autoscaling. See `matplotlib.axes.Axes.autoscale()` for full explanation. Note that this function behaves the same, but for all three axes. Therefore, 'z' can be passed for *axis*, and 'both' applies to all three axes.

New in version 1.1.0.

auto_scale_xyz (*X*, *Y*, *Z=None*, *had_data=None*)

autoscale_view (*tight=None*, *scalex=True*, *scaley=True*, *scalez=True*)
Autoscale the view limits using the data limits. See `matplotlib.axes.Axes.autoscale_view()` for documentation. Note that this function applies to the 3D axes, and as such adds the *scalez* to the function arguments.

Changed in version 1.1.0: Function signature was changed to better match the 2D version. *tight* is now explicitly a kwarg and placed first.

Changed in version 1.2.1: This is now fully functional.

get_w_lims ()
Get 3D world limits.

_determine_lims (*xmin=None*, *xmax=None*, **args*, ***kwargs*)

set_xlim3d (*left=None*, *right=None*, *emit=True*, *auto=False*, ***, *xmin=None*, *xmax=None*)
Set 3D x limits.

See `matplotlib.axes.Axes.set_xlim()` for full documentation.

set_xlim(*left=None*, *right=None*, *emit=True*, *auto=False*, ***, *xmin=None*, *xmax=None*)
Set 3D x limits.

See `matplotlib.axes.Axes.set_xlim()` for full documentation.

set_ylim3d(*bottom=None*, *top=None*, *emit=True*, *auto=False*, ***, *ymin=None*, *ymax=None*)
Set 3D y limits.

See `matplotlib.axes.Axes.set_ylim()` for full documentation.

set_ylim(*bottom=None*, *top=None*, *emit=True*, *auto=False*, ***, *ymin=None*, *ymax=None*)
Set 3D y limits.

See `matplotlib.axes.Axes.set_ylim()` for full documentation.

set_zlim3d(*bottom=None*, *top=None*, *emit=True*, *auto=False*, ***, *zmin=None*, *zmax=None*)
Set 3D z limits.

See `matplotlib.axes.Axes.set_ylim()` for full documentation

set_zlim(*bottom=None*, *top=None*, *emit=True*, *auto=False*, ***, *zmin=None*, *zmax=None*)
Set 3D z limits.

See `matplotlib.axes.Axes.set_ylim()` for full documentation

get_xlim3d()
Return the x-axis view limits.

left, right [(float, float)] The current x-axis limits in data coordinates.

set_xlim set_xbound, get_xbound invert_xaxis, xaxis_inverted

The x-axis may be inverted, in which case the *left* value will be greater than the *right* value.

Changed in version 1.1.0: This function now correctly refers to the 3D x-limits

get_xlim()
Return the x-axis view limits.

left, right [(float, float)] The current x-axis limits in data coordinates.

set_xlim set_xbound, get_xbound invert_xaxis, xaxis_inverted

The x-axis may be inverted, in which case the *left* value will be greater than the *right* value.

Changed in version 1.1.0: This function now correctly refers to the 3D x-limits

get_ylim3d()
Return the y-axis view limits.

bottom, top [(float, float)] The current y-axis limits in data coordinates.

set_ylim set_ybound, get_ybound invert_yaxis, yaxis_inverted

The y-axis may be inverted, in which case the *bottom* value will be greater than the *top* value.

Changed in version 1.1.0: This function now correctly refers to the 3D y-limits.

get_ylim()
Return the y-axis view limits.

bottom, top [(float, float)] The current y-axis limits in data coordinates.

set_ylim set_ybound, get_ybound invert_yaxis, yaxis_inverted

The y-axis may be inverted, in which case the *bottom* value will be greater than the *top* value.

Changed in version 1.1.0: This function now correctly refers to the 3D y-limits.

get_zlim3d()
Get 3D z limits.

get_zlim()
Get 3D z limits.

get_zscale()

set_xscale(*value*, ***kwargs*)
Set the x-axis scale.

value [{"linear"}] The axis scale type to apply. 3D axes currently only support linear scales; other scales yield nonsensical results.

****kwargs** Keyword arguments are nominally forwarded to the scale class, but none of them is applicable for linear scales.

set_yscale(*value*, ***kwargs*)
Set the y-axis scale.

value [{"linear"}] The axis scale type to apply. 3D axes currently only support linear scales; other scales yield nonsensical results.

****kwargs** Keyword arguments are nominally forwarded to the scale class, but none of them is applicable for linear scales.

set_zscale(*value*, ***kwargs*)
Set the z-axis scale.

value [{"linear"}] The axis scale type to apply. 3D axes currently only support linear scales; other scales yield nonsensical results.

****kwargs** Keyword arguments are nominally forwarded to the scale class, but none of them is applicable for linear scales.

set_zticks(**args*, ***kwargs*)
Set z-axis tick locations. See `matplotlib.axes.Axes.set_yticks()` for more details.

Note: Minor ticks are not supported.

New in version 1.1.0.

get_zticks(*minor=False*)
Return the z ticks as a list of locations See `matplotlib.axes.Axes.get_yticks()` for more details.

Note: Minor ticks are not supported.

New in version 1.1.0.

get_zmajorticklabels()
Get the ztick labels as a list of Text instances

New in version 1.1.0.

get_zminorticklabels()
Get the ztick labels as a list of Text instances

Note: Minor ticks are not supported. This function was added only for completeness.

New in version 1.1.0.

set_zticklabels(**args*, ***kwargs*)
Set z-axis tick labels. See `matplotlib.axes.Axes.set_yticklabels()` for more details.

Note: Minor ticks are not supported by Axes3D objects.

New in version 1.1.0.

get_zticklabels(*minor=False*)
Get ztick labels as a list of Text instances. See `matplotlib.axes.Axes.get_yticklabels()` for more details.

Note: Minor ticks are not supported.

New in version 1.1.0.

zaxis_date(*tz=None*)
Sets up z-axis ticks and labels that treat the z data as dates.

tz is a timezone string or `tzinfo` instance. Defaults to rc value.

Note: This function is merely provided for completeness. Axes3D objects do not officially support dates for ticks, and so this may or may not work as expected.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

get_zticklines()
Get ztick lines as a list of Line2D instances. Note that this function is provided merely for completeness. These lines are re-calculated as the display changes.

New in version 1.1.0.

clabel(**args*, ***kwargs*)
This function is currently not implemented for 3D axes. Returns *None*.

view_init(*elev=None*, *azim=None*)
Set the elevation and azimuth of the axes in degrees (not radians).

This can be used to rotate the axes programmatically.

'elev' stores the elevation angle in the z plane (in degrees). 'azim' stores the azimuth angle in the x,y plane (in degrees).

if elev or azim are None (default), then the initial value is used which was specified in the *Axes3D* constructor.

set_proj_type(*proj_type*)
Set the projection type.

proj_type [str] Type of projection, accepts 'persp' and 'ortho'.

get_proj()
Create the projection matrix from the current viewing position.

elev stores the elevation angle in the z plane azim stores the azimuth angle in the x,y plane

dist is the distance of the eye viewing point from the object point.

mouse_init(*rotate_btn=1*, *zoom_btn=3*)
Initializes mouse button callbacks to enable 3D rotation of the axes. Also optionally sets the mouse buttons for 3D rotation and zooming.

rotate_btn [int or list of int] The mouse button or buttons to use for 3D rotation of the axes; defaults to 1.

zoom_btn [int or list of int] The mouse button or buttons to use to zoom the 3D axes; defaults to 3.

can_zoom()
Return *True* if this axes supports the zoom box button functionality.

3D axes objects do not use the zoom box button.

can_pan()
Return *True* if this axes supports the pan/zoom button functionality.

3D axes objects do not use the pan/zoom button.

cla()
Clear axes

disable_mouse_rotation()
Disable mouse button callbacks.

_button_press(*event*)

_button_release(*event*)

format_zdata(*z*)
Return *z* string formatted. This function will use the `fmt_zdata` attribute if it is callable, else will fall back on the zaxis major formatter

format_coord(*xd*, *yd*)
Given the 2D view coordinates attempt to guess a 3D coordinate. Looks for the nearest edge to the point and then assumes that the point is at the same z location as the nearest point on the edge.

_on_move(*event*)
Mouse moving

button-1 rotates by default. Can be set explicitly in mouse_init(). button-3 zooms by default. Can be set explicitly in mouse_init().

set_zlabel(*zlabel*, *fontdict=None*, *labelpad=None*, ***kwargs*)
Set zlabel. See doc for `set_ylabel()` for description.

get_zlabel()
Get the z-label text string.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

get_frame_on()
Get whether the 3D axes panels are drawn.

set_frame_on(*b*)
Set whether the 3D axes panels are drawn.

b : bool

grid(*b=True*, ***kwargs*)
Set / unset 3D grid.

Note: Currently, this function does not behave the same as `matplotlib.axes.Axes.grid()`, but it is intended to eventually support that behavior.

Changed in version 1.1.0: This function was changed, but not tested. Please report any bugs.

ticklabel_format(*, *style=''*, *scilimits=None*, *useOffset=None*, *axis='both'*)
Convenience method for manipulating the ScalarFormatter used by default for linear axes in Axed3D objects.

See `matplotlib.axes.Axes.ticklabel_format()` for full documentation. Note that this version applies to all three axes of the Axes3D object. Therefore, the *axis* argument will also accept a value of 'z' and the value of 'both' will apply to all three axes.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

locator_params(*axis='both'*, *tight=None*, ***kwargs*)
Convenience method for controlling tick locators.

See `matplotlib.axes.Axes.locator_params()` for full documentation. Note that this is for Axes3D objects, therefore, setting *axis* to 'both' will result in the parameters being set for all three axes. Also, *axis* can also take a value of 'z' to apply parameters to the z axis.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

tick_params(*axis='both'*, ***kwargs*)
Convenience method for changing the appearance of ticks and tick labels.

See `matplotlib.axes.Axes.tick_params()` for more complete documentation.

The only difference is that setting *axis* to 'both' will mean that the settings are applied to all three axes. Also, the *axis* parameter also accepts a value of 'z', which would mean to apply to only the z-axis.

Also, because of how Axes3D objects are drawn very differently from regular 2D axes, some of these settings may have ambiguous meaning. For simplicity, the 'z' axis will accept settings as if it was like the 'y' axis.

Note: Axes3D currently ignores some of these settings.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

invert_zaxis()
Invert the z-axis.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

zaxis_inverted()
Returns True if the z-axis is inverted.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

get_zbound()
Returns the z-axis numerical bounds where:

```
lowerBound < upperBound
```

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

set_zbound(*lower=None*, *upper=None*)
Set the lower and upper numerical bounds of the z-axis. This method will honor axes inversion regardless of parameter order. It will not change the `_autoscaleZon` attribute.

New in version 1.1.0: This function was added, but not tested. Please report any bugs.

text (*x*, *y*, *z*, *s*, *zdir=None*, ***kwargs*)
Add text to the plot. kwargs will be passed on to Axes.text, except for the *zdir* keyword, which sets the direction to be used as the z direction.

text3D (*x*, *y*, *z*, *s*, *zdir=None*, ***kwargs*)
Add text to the plot. kwargs will be passed on to Axes.text, except for the *zdir* keyword, which sets the direction to be used as the z direction.

text2D (*x*, *y*, *s*, *fontdict=None*, *withdash=<deprecated parameter>*, ***kwargs*)
Add text to the axes.

Add the text *s* to the axes at location *x*, *y* in data coordinates.

x, y [scalars] The position to place the text. By default, this is in data coordinates. The coordinate system can be changed using the *transform* parameter.

s [str] The text.

fontdict [dictionary, optional, default: None] A dictionary to override the default text properties. If fontdict is None, the defaults are determined by your rc parameters.

withdash [boolean, optional, default: False] Creates a *~matplotlib.text.TextWithDash* instance instead of a *~matplotlib.text.Text* instance.

text [*.Text*] The created *.Text* instance.

****kwargs** [*~matplotlib.text.Text* properties.] Other miscellaneous text parameters.

Individual keyword arguments can be used to override any given parameter:

```
>>> text(x, y, s, fontsize=12)
```

The default transform specifies that text is in data coords, alternatively, you can specify text in axis coords (0,0 is lower-left and 1,1 is upper-right). The example below places text in the center of the axes:

```
>>> text(0.5, 0.5, 'matplotlib', horizontalalignment='center',
...      verticalalignment='center', transform=ax.transAxes)
```

You can put a rectangular box around the text instance (e.g., to set a background color) by using the keyword *bbox*. *bbox* is a dictionary of *~matplotlib.patches.Rectangle* properties. For example:

```
>>> text(x, y, s, bbox=dict(facecolor='red', alpha=0.5))
```

plot (*xs*, *ys*, **args*, *zdir='z'*, ***kwargs*)
Plot 2D or 3D data.

xs [1D array-like] x coordinates of vertices.

ys [1D array-like] y coordinates of vertices.

zs [scalar or 1D array-like] z coordinates of vertices; either one for all points or one for each point.

zdir [{'x', 'y', 'z'}] When plotting 2D data, the direction to use as z ('x', 'y' or 'z'); defaults to 'z'.

****kwargs** Other arguments are forwarded to *matplotlib.axes.Axes.plot*.

plot3D (*xs*, *ys*, **args*, *zdir='z'*, ***kwargs*)
Plot 2D or 3D data.

xs [1D array-like] x coordinates of vertices.

ys [1D array-like] y coordinates of vertices.

zs [scalar or 1D array-like] z coordinates of vertices; either one for all points or one for each point.

zdir [{'x', 'y', 'z'}] When plotting 2D data, the direction to use as z ('x', 'y' or 'z'); defaults to 'z'.

****kwargs** Other arguments are forwarded to *matplotlib.axes.Axes.plot*.

plot_surface(*X*, *Y*, *Z*, **args*, *norm=None*, *vmin=None*, *vmax=None*, *lightsource=None*, ***kwargs*)
Create a surface plot.

By default it will be colored in shades of a solid color, but it also supports color mapping by supplying the *cmap* argument.

Note: The *rcount* and *ccount* kwargs, which both default to 50, determine the maximum number of samples used in each direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points.

X, Y, Z [2d arrays] Data values.

rcount, ccount [int] Maximum number of samples used in each direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points. Defaults to 50.

New in version 2.0.

rstride, cstride [int] Downsampling stride in each direction. These arguments are mutually exclusive with *rcount* and *ccount*. If only one of *rstride* or *cstride* is set, the other defaults to 10.

'classic' mode uses a default of `rstride = cstride = 10` instead of the new default of `rcount = ccount = 50`.

color [color-like] Color of the surface patches.

cmap [Colormap] Colormap of the surface patches.

facecolors [array-like of colors.] Colors of each individual patch.

norm [Normalize] Normalization for the colormap.

vmin, vmax [float] Bounds for the normalization.

shade [bool] Whether to shade the facecolors. Defaults to True. Shading is always disabled when *cmap* is specified.

lightsource [*~matplotlib.colors.LightSource*] The lightsource to use when *shade* is True.

****kwargs** Other arguments are forwarded to *.Poly3DCollection*.

_generate_normals(*polygons*)
Takes a list of polygons and return an array of their normals.

Normals point towards the viewer for a face with its vertices in counterclockwise order, following the right hand rule.

Uses three points equally spaced around the polygon. This normal of course might not make sense for polygons with more than three points not lying in a plane, but it's a plausible and fast approximation.

polygons: list of (M_i, 3) array_like, or (..., M, 3) array_like A sequence of polygons to compute normals for, which can have varying numbers of vertices. If the polygons all have the same number of vertices and array is passed, then the operation will be vectorized.

normals: (..., 3) array_like A normal vector estimated for the polygon.

_shade_colors(*color*, *normals*, *lightsource=None*)
Shade *color* using normal vectors given by *normals*. *color* can also be an array of the same length as *normals*.

plot_wireframe(*X*, *Y*, *Z*, **args*, ***kwargs*)
Plot a 3D wireframe.

Note: The *rcount* and *ccount* kwargs, which both default to 50, determine the maximum number of samples used in each direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points.

X, Y, Z [2d arrays] Data values.

rcount, ccount [int] Maximum number of samples used in each direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points. Setting a count to zero causes the data to be not sampled in the corresponding direction, producing a 3D line plot rather than a wireframe plot. Defaults to 50.

New in version 2.0.

rstride, cstride [int] Downsampling stride in each direction. These arguments are mutually exclusive with *rcount* and *ccount*. If only one of *rstride* or *cstride* is set, the other defaults to 1. Setting a stride to zero causes the data to be not sampled in the corresponding direction, producing a 3D line plot rather than a wireframe plot.

'classic' mode uses a default of `rstride = cstride = 1` instead of the new default of `rcount = ccount = 50`.

****kwargs** Other arguments are forwarded to *.Line3DCollection*.

plot_trisurf(**args*, *color=None*, *norm=None*, *vmin=None*, *vmax=None*, *lightsource=None*, ***kwargs*)
Plot a triangulated surface.

The (optional) triangulation can be specified in one of two ways; either:

```
plot_trisurf(triangulation, ...)
```

where triangulation is a `Triangulation` object, or:

```
plot_trisurf(X, Y, ...)
plot_trisurf(X, Y, triangles, ...)
plot_trisurf(X, Y, triangles=triangles, ...)
```

in which case a Triangulation object will be created. See `Triangulation` for a explanation of these possibilities.

The remaining arguments are:

```
plot_trisurf(..., Z)
```

where *Z* is the array of values to contour, one per point in the triangulation.

X, Y, Z [array-like] Data values as 1D arrays.

color Color of the surface patches.

cmap A colormap for the surface patches.

norm [Normalize] An instance of Normalize to map values to colors.

vmin, vmax [scalar, optional, default: None] Minimum and maximum value to map.

shade [bool] Whether to shade the facecolors. Defaults to True. Shading is always disabled when *cmap* is specified.

lightsource [*~matplotlib.colors.LightSource*] The lightsource to use when *shade* is True.

****kwargs** All other arguments are passed on to `Poly3DCollection`

New in version 1.2.0.

_3d_extend_contour(*cset*, *stride=5*)
Extend a contour in 3D by creating

add_contour_set(*cset*, *extend3d=False*, *stride=5*, *zdir='z'*, *offset=None*)

add_contourf_set(*cset*, *zdir='z'*, *offset=None*)

contour(*X*, *Y*, *Z*, **args*, *extend3d=False*, *stride=5*, *zdir='z'*, *offset=None*, ***kwargs*)
Create a 3D contour plot.

X, Y, Z [array-likes] Input data.

extend3d [bool] Whether to extend contour in 3D; defaults to False.

stride [int] Step size for extending contour.

zdir [{'x', 'y', 'z'}] The direction to use; defaults to 'z'.

offset [scalar] If specified, plot a projection of the contour lines at this position in a plane normal to zdir

***args, ****kwargs** Other arguments are forwarded to *matplotlib.axes.Axes.contour*.

matplotlib.contour.QuadContourSet

contour3D(*X*, *Y*, *Z*, **args*, *extend3d=False*, *stride=5*, *zdir='z'*, *offset=None*, ***kwargs*)
Create a 3D contour plot.

X, Y, Z [array-likes] Input data.

extend3d [bool] Whether to extend contour in 3D; defaults to False.

stride [int] Step size for extending contour.

zdir [{'x', 'y', 'z'}] The direction to use; defaults to 'z'.

offset [scalar] If specified, plot a projection of the contour lines at this position in a plane normal to zdir

***args, ****kwargs** Other arguments are forwarded to *matplotlib.axes.Axes.contour*.

matplotlib.contour.QuadContourSet

tricontour(**args*, *extend3d=False*, *stride=5*, *zdir='z'*, *offset=None*, ***kwargs*)
Create a 3D contour plot.

Changed in version 1.3.0: Added support for custom triangulations

---

Note: This method currently produces incorrect output due to a longstanding bug in 3D PolyCollection rendering.

X, Y, Z [array-likes] Input data.

extend3d [bool] Whether to extend contour in 3D; defaults to False.

stride [int] Step size for extending contour.

zdir [{'x', 'y', 'z'}] The direction to use; defaults to 'z'.

offset [scalar] If specified, plot a projection of the contour lines at this position in a plane normal to zdir

***args, ****kwargs** Other arguments are forwarded to *matplotlib.axes.Axes.tricontour*.

matplotlib.tri.tricontour.TriContourSet

contourf(*X*, *Y*, *Z*, **args*, *zdir='z'*, *offset=None*, ***kwargs*)
Create a 3D filled contour plot.

X, Y, Z [array-likes] Input data.

zdir [{'x', 'y', 'z'}] The direction to use; defaults to 'z'.

offset [scalar] If specified, plot a projection of the contour lines at this position in a plane normal to zdir

***args, ****kwargs** Other arguments are forwarded to *matplotlib.axes.Axes.contourf*.

matplotlib.contour.QuadContourSet

New in version 1.1.0: The *zdir* and *offset* parameters.

contourf3D(*X*, *Y*, *Z*, **args*, *zdir='z'*, *offset=None*, ***kwargs*)
Create a 3D filled contour plot.

X, Y, Z [array-likes] Input data.

zdir [{'x', 'y', 'z'}] The direction to use; defaults to 'z'.

offset [scalar] If specified, plot a projection of the contour lines at this position in a plane normal to zdir

***args, ****kwargs** Other arguments are forwarded to *matplotlib.axes.Axes.contourf*.

matplotlib.contour.QuadContourSet

New in version 1.1.0: The *zdir* and *offset* parameters.

tricontourf(**args*, *zdir='z'*, *offset=None*, ***kwargs*)
Create a 3D filled contour plot.

Note: This method currently produces incorrect output due to a longstanding bug in 3D PolyCollection rendering.

X, Y, Z [array-likes] Input data.

zdir [{'x', 'y', 'z'}] The direction to use; defaults to 'z'.

offset [scalar] If specified, plot a projection of the contour lines at this position in a plane normal to zdir

***args, ****kwargs** Other arguments are forwarded to *matplotlib.axes.Axes.tricontourf*.

matplotlib.tri.tricontour.TriContourSet

New in version 1.1.0: The *zdir* and *offset* parameters.

Changed in version 1.3.0: Added support for custom triangulations

add_collection3d(*col*, *zs=0*, *zdir='z'*)
Add a 3D collection object to the plot.

2D collection types are converted to a 3D version by modifying the object and adding z coordinate information.

Supported are:

- PolyCollection
- LineCollection
- PatchCollection

scatter (*xs*, *ys*, *zs=0*, *zdir='z'*, *s=20*, *c=None*, *depthshade=True*, **args*, ***kwargs*)
Create a scatter plot.

xs, ys [array-like] The data positions.

zs [float or array-like, optional, default: 0] The z-positions. Either an array of the same length as *xs* and *ys* or a single value to place all points in the same plane.

zdir [{'x', 'y', 'z', '-x', '-y', '-z'}, optional, default: 'z'] The axis direction for the *zs*. This is useful when plotting 2D data on a 3D Axes. The data must be passed as *xs*, *ys*. Setting *zdir* to 'y' then plots the data to the x-z-plane.

See also /gallery/mplot3d/2dcollections3d.

s [scalar or array-like, optional, default: 20] The marker size in points**2. Either an array of the same length as *xs* and *ys* or a single value to make all markers the same size.

c [color, sequence, or sequence of color, optional] The marker color. Possible values:

- A single color format string.
- A sequence of color specifications of length n.
- A sequence of n numbers to be mapped to colors using *cmap* and *norm*.
- A 2-D array in which the rows are RGB or RGBA.

For more details see the *c* argument of *~.axes.Axes.scatter*.

depthshade [bool, optional, default: True] Whether to shade the scatter markers to give the appearance of depth. Each call to `scatter()` will perform its depthshading independently.

****kwargs** All other arguments are passed on to *~.axes.Axes.scatter*.

paths : *~matplotlib.collections.PathCollection*

scatter3D (*xs*, *ys*, *zs=0*, *zdir='z'*, *s=20*, *c=None*, *depthshade=True*, **args*, ***kwargs*)
Create a scatter plot.

xs, ys [array-like] The data positions.

zs [float or array-like, optional, default: 0] The z-positions. Either an array of the same length as *xs* and *ys* or a single value to place all points in the same plane.

zdir [{'x', 'y', 'z', '-x', '-y', '-z'}, optional, default: 'z'] The axis direction for the *zs*. This is useful when plotting 2D data on a 3D Axes. The data must be passed as *xs*, *ys*. Setting *zdir* to 'y' then plots the data to the x-z-plane.

See also /gallery/mplot3d/2dcollections3d.

s [scalar or array-like, optional, default: 20] The marker size in points**2. Either an array of the same length as *xs* and *ys* or a single value to make all markers the same size.

c [color, sequence, or sequence of color, optional] The marker color. Possible values:

- A single color format string.
- A sequence of color specifications of length n.
- A sequence of n numbers to be mapped to colors using *cmap* and *norm*.
- A 2-D array in which the rows are RGB or RGBA.

For more details see the *c* argument of *~.axes.Axes.scatter*.

depthshade [bool, optional, default: True] Whether to shade the scatter markers to give the appearance of depth. Each call to `scatter()` will perform its depthshading independently.

****kwargs** All other arguments are passed on to *~.axes.Axes.scatter*.

paths : *~matplotlib.collections.PathCollection*

bar(*left*, *height*, *zs=0*, *zdir='z'*, **args*, ***kwargs*)
Add 2D bar(s).

left [1D array-like] The x coordinates of the left sides of the bars.

height [1D array-like] The height of the bars.

zs [scalar or 1D array-like] Z coordinate of bars; if a single value is specified, it will be used for all bars.

zdir [{'x', 'y', 'z'}] When plotting 2D data, the direction to use as z ('x', 'y' or 'z'); defaults to 'z'.

****kwargs** Other arguments are forwarded to *matplotlib.axes.Axes.bar*.

mpl_toolkits.mplot3d.art3d.Patch3DCollection

bar3d(*x*, *y*, *z*, *dx*, *dy*, *dz*, *color=None*, *zsort='average'*, *shade=True*, **args*, ***kwargs*)
Generate a 3D barplot.

This method creates three dimensional barplot where the width, depth, height, and color of the bars can all be uniquely set.

x, y, z [array-like] The coordinates of the anchor point of the bars.

dx, dy, dz [scalar or array-like] The width, depth, and height of the bars, respectively.

color [sequence of valid color specifications, optional] The color of the bars can be specified globally or individually. This parameter can be:

- A single color value, to color all bars the same color.
- An array of colors of length N bars, to color each bar independently.
- An array of colors of length 6, to color the faces of the bars similarly.
- An array of colors of length 6 * N bars, to color each face independently.

When coloring the faces of the boxes specifically, this is the order of the coloring:

1. -Z (bottom of box)
2. +Z (top of box)
3. -Y
4. +Y
5. -X
6. +X

zsort [str, optional] The z-axis sorting scheme passed onto *~.art3d.Poly3DCollection*

shade [bool, optional (default = True)] When true, this shades the dark sides of the bars (relative to the plot's source of light).

****kwargs** Any additional keyword arguments are passed onto *~.art3d.Poly3DCollection*.

collection [*~.art3d.Poly3DCollection*] A collection of three dimensional polygons representing the bars.

set_title (*label*, *fontdict=None*, *loc='center'*, ***kwargs*)
Set a title for the axes.

Set one of the three available axes titles. The available titles are positioned above the axes in the center, flush with the left edge, and flush with the right edge.

label [str] Text to use for the title

fontdict [dict] A dictionary controlling the appearance of the title text, the default *fontdict* is:

```
{'fontsize': rcParams['axes.titlesize'],
 'fontweight' : rcParams['axes.titleweight'],
 'verticalalignment': 'baseline',
 'horizontalalignment': loc}
```

loc [{'center', 'left', 'right'}, str, optional] Which title to set, defaults to 'center'

pad [float] The offset of the title from the top of the axes, in points. Default is `None` to use rcParams['axes.titlepad'].

text [`Text`] The matplotlib text instance representing the title

****kwargs** [~*matplotlib.text.Text* properties] Other keyword arguments are text properties, see `Text` for a list of valid text properties.

quiver (*X*, *Y*, *Z*, *U*, *V*, *W*, */*, *length=1*, *arrow_length_ratio=.3*, *pivot='tail'*, *normalize=False*, ***kwargs*)
Plot a 3D field of arrows.

The arguments could be array-like or scalars, so long as they they can be broadcast together. The arguments can also be masked arrays. If an element in any of argument is masked, then that corresponding quiver element will not be plotted.

X, Y, Z [array-like] The x, y and z coordinates of the arrow locations (default is tail of arrow; see *pivot* kwarg)

U, V, W [array-like] The x, y and z components of the arrow vectors

length [float] The length of each quiver, default to 1.0, the unit is the same with the axes

arrow_length_ratio [float] The ratio of the arrow head with respect to the quiver, default to 0.3

pivot [{'tail', 'middle', 'tip'}] The part of the arrow that is at the grid point; the arrow rotates about this point, hence the name *pivot*. Default is 'tail'

normalize [bool] When True, all of the arrows will be the same length. This defaults to False, where the arrows will be different lengths depending on the values of u,v,w.

****kwargs** Any additional keyword arguments are delegated to `LineCollection`

quiver3D (**args*, *length=1*, *arrow_length_ratio=0.3*, *pivot='tail'*, *normalize=False*, ***kwargs*)
ax.quiver(X, Y, Z, U, V, W, /, length=1, arrow_length_ratio=.3, pivot='tail', normalize=False, **kwargs)

Plot a 3D field of arrows.

The arguments could be array-like or scalars, so long as they they can be broadcast together. The arguments can also be masked arrays. If an element in any of argument is masked, then that corresponding quiver element will not be plotted.

X, Y, Z [array-like] The x, y and z coordinates of the arrow locations (default is tail of arrow; see *pivot* kwarg)

U, V, W [array-like] The x, y and z components of the arrow vectors

length [float] The length of each quiver, default to 1.0, the unit is the same with the axes

arrow_length_ratio [float] The ratio of the arrow head with respect to the quiver, default to 0.3

pivot [{'tail', 'middle', 'tip'}] The part of the arrow that is at the grid point; the arrow rotates about this point, hence the name *pivot*. Default is 'tail'

normalize [bool] When True, all of the arrows will be the same length. This defaults to False, where the arrows will be different lengths depending on the values of u,v,w.

****kwargs** Any additional keyword arguments are delegated to `LineCollection`

voxels([*x*, *y*, *z*], */*, *filled*, *facecolors=None*, *edgecolors=None*, ***kwargs*)
Plot a set of filled voxels

All voxels are plotted as 1x1x1 cubes on the axis, with filled[0,0,0] placed with its lower corner at the origin. Occluded faces are not plotted.

New in version 2.1.

filled [3D np.array of bool] A 3d array of values, with truthy values indicating which voxels to fill

x, y, z [3D np.array, optional] The coordinates of the corners of the voxels. This should broadcast to a shape one larger in every dimension than the shape of *filled*. These can be used to plot non-cubic voxels.

If not specified, defaults to increasing integers along each axis, like those returned by `indices()`. As indicated by the `/` in the function signature, these arguments can only be passed positionally.

facecolors, edgecolors [array_like, optional] The color to draw the faces and edges of the voxels. Can only be passed as keyword arguments. This parameter can be:

- A single color value, to color all voxels the same color. This can be either a string, or a 1D rgb/rgba array
- `None`, the default, to use a single color for the faces, and the style default for the edges.
- A 3D ndarray of color names, with each item the color for the corresponding voxel. The size must match the voxels.
- A 4D ndarray of rgb/rgba data, with the components along the last axis.

shade [bool] Whether to shade the facecolors. Defaults to True. Shading is always disabled when *cmap* is specified.

New in version 3.1.

lightsource [*~matplotlib.colors.LightSource*] The lightsource to use when *shade* is True.

New in version 3.1.

****kwargs** Additional keyword arguments to pass onto `Poly3DCollection()`

faces [dict] A dictionary indexed by coordinate, where `faces[i,j,k]` is a *Poly3DCollection* of the faces drawn for the voxel `filled[i,j,k]`. If no faces were drawn for a given voxel, either because it was not asked to be drawn, or it is fully occluded, then `(i,j,k) not in faces`.

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

Args:

data

tuple, list, ndarray

Returns:

returns

ndarray with .ndim >= 2

luxpy.utils.**np3d**(*data*)

Make a tuple, list or numpy array at least a 3d numpy array.

Args:

data

tuple, list, ndarray

Returns:

returns

ndarray with .ndim >= 3

luxpy.utils.**np2dT**(*data*)

Make a tuple, list or numpy array at least a 2D numpy array and transpose.

Args:

data

tuple, list, ndarray

Returns:

returns

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

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

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

Args:

data

tuple, list, ndarray

Returns:

returns

ndarray with .ndim >= 3 and with first two axes
transposed (axis=3 is kept the same).

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

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

Args:

db

dict

Returns:

returns

dict with the values of specific keys overwritten by the
not-None values of corresponding args of a function fcn.

Example usage:

```
_db = {'c' : 'c1', 'd' : 10, 'e' : {'e1':'hello', 'e2':1000}}

def test_put_args_in_db(a, b, db = None, c = None,d = None,e = None):

    args = locals().copy() # get dict with keyword input arguments to
                # function 'test_put_args_in_db'

        db = put_args_in_db(db,args) # overwrite non-None args in db copy.

    if db is not None: # unpack db for further use
                c,d,e = [db[x] for x in sorted(db.keys())]

        print(' a : {}'.format(a))
        print(' b : {}'.format(b))
        print(' db: {}'.format(db))
        print(' c : {}'.format(c))
        print(' d : {}'.format(d))
        print(' e : {}'.format(e))
        print('_db: {}'.format(_db))
```

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

Convert dict to vec and vice versa.

Args:

vec

list or vector array, optional

dic

dict, optional

vsize

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

keys

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

Returns:

returns

x, vsize
x is an array, if vec is None
x is a dict, if vec is not None

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

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

Args:

data

- str with path to file containing data

- ndarray with data
- pandas.dataframe with data

kind

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

columns

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

header

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

sep

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

datatype'

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

copy

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

verbosity

True, False, optional
Print warning when inferring headers from file.

Returns:

returns

data as ndarray or pandas.dataframe

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

Easy input of of keys and values into dict.

Args:

keys

iterable list[str,...] of keys

values

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

ordered

True, False, optional

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

Returns:

returns

(ordered) dict

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

Create a meshed block from x and y.

(Similar to meshgrid, but axis = 0 is retained).
To enable fast blockwise calculation.

Args:

x

ndarray with ndim == 2

y

ndarray with ndim == 2

Returns:

X,Y

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

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

Split data on last axis

Args:

data

ndarray

Returns:

returns

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

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

Join data on last axis.

Args:

data

tuple (ndarray, ndarray, ...)

Returns:

returns

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

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

Broadcasts shapes of data to a target_shape.

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

Args:

data

ndarray

target_shape

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

expand_2d_to_3d

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

axis0_repeats

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

axis1_repeats

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

Returns:

returns

reshaped ndarray

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

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

Args:

x

ndarray

tshape

tuple with target shape

add_axis

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

equal_shape

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

Returns:

returns

ndarray broadcast-compatable with tshape.

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

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

Args:

filename

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

df

dataframe to save to workbook

sheet_name

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

startrow

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

truncate_sheet

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

to_excel_kwargs

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

Returns: None

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

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

Show luxpy foler tree.

Args:

omit

List of folders and file-extensions to omit.

Returns: None

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

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

Args:

string

string with package or module name

try_pip_install

False, optional
True: try pip installing it using subprocess

Returns:

success

True if importable, False if not.

luxpy.utils.**get_function_kwargs**(*f*)
Get dictionary of a function's keyword arguments and their default values.

Args:

f

function name

Returns:

dict

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

luxpy.utils.**profile_fcn**(*fcn*, *profile=True*, *sort_stats='tottime'*, *output_file=None*)
Profile or time a function fcn.

Args:

fcn

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

profile

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

sort_stats

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

output_file

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

Return:

ps

Profiler output

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 matric, sigmainv.

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

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

geomean() Calculates geometric mean along axis.

polyarea()

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

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

cart2pol() Converts Cartesian to polar coordinates.

pol2cart() Converts polar to Cartesian coordinates.

cart2spher() Converts Cartesian to spherical coordinates.

spher2cart() Converts spherical to Cartesian coordinates.

magnitude_v() Calculates magnitude of vector.

angle_v1v2() Calculates angle between two vectors.

histogram()

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

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

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

minimizebnd() scipy.minimize() that allows contrained 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]b

fit_ellipse() Fit an ellipse to supplied data points.

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

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

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

ndinterp1_scipy() Perform n-dimensional interpolation using Delaunay triangulation (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 uninmported 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([[1.0000e+00*, *1.0000e+00*, *1.0000e+00]])*)
Normalize 3x3 matrix M to xyz0 – > [1,1,1]

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

Args:

M

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

xyz0

2darray, optional

Returns:

returns

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

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

Args:

A

ndarray

atol

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

rtol

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

method

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

forcesymm

True or False, optional
If A is not symmetric, force symmetry using:
A = numpy.triu(A) + numpy.triu(A).T - numpy.diag(numpy.diag(A))

Returns:

returns

ndarray with positive-definite matrix.

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

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

Args:

A

ndarray

atol

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

rtol

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

Returns:

returns

Bool
True: the array is symmetric within the given tolerance

luxpy.math.**check_posdef**(*A*, *atol=1e-09*, *rtol=1e-09*)
Checks positive definiteness of a matrix via Cholesky.

Args:

A

ndarray

atol

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

rtol

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

Returns:

returns

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

luxpy.math.**positive_arctan**(*x*, *y*, *htype='deg'*)
Calculate positive angle (0°-360° or 0 - 2*pi rad.) from x and y.

Args:

x

ndarray of x-coordinates

y

ndarray of y-coordinates

htype

'deg' or 'rad', optional
- 'deg': hue angle between 0° and 360°
- 'rad': hue angle between 0 and 2pi radians

Returns:

returns

ndarray of positive angles.

luxpy.math.**line_intersect**(*a1*, *a2*, *b1*, *b2*)
Line intersections of series of two line segments a and b.

Args:

a1

ndarray (.shape = (N,2)) specifying end-point 1 of line a

a2

ndarray (.shape = (N,2)) specifying end-point 2 of line a

b1

ndarray (.shape = (N,2)) specifying end-point 1 of line b

b2

ndarray (.shape = (N,2)) specifying end-point 2 of line b

Note: N is the number of line segments a and b.

Returns:

returns

ndarray with line-intersections (.shape = (N,2))

References:

1. https://stackoverflow.com/questions/3252194/numpy-and-line-intersections

luxpy.math.**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 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 erf(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 nump.histogram is run.
Mimicks matlab hist (uses bin centers).

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

Returns:

returns

ndarray with histogram

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

Convert Cartesion to polar coordinates.

Args:

theta

float or ndarray with theta-coordinates

r

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

htype

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

Returns:

returns

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

luxpy.math.**cart2pol**(*x*, *y=None*, *htype='deg'*)
Convert Cartesion to polar coordinates.

Args:

x

float or ndarray with x-coordinates

y

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

htype

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

Returns:

returns

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

luxpy.math.**spher2cart**(*theta*, *phi*, *r=1.0*, *deg=True*)
Convert spherical to cartesian coordinates.

Args:

theta

Float, int or ndarray
Angle with positive z-axis.

phi

Float, int or ndarray
Angle around positive z-axis starting from x-axis.

r

1, optional
Float, int or ndarray
radius

Returns:

x, y, z

tuple of floats, ints or ndarrays
Cartesian coordinates

luxpy.math.**cart2spher**(*x*, *y*, *z*, *deg=True*)
Convert cartesian to spherical coordinates.

Args:

x, y, z

tuple of floats, ints or ndarrays
Cartesian coordinates

Returns:

theta

Float, int or ndarray
Angle with positive z-axis.

phi

Float, int or ndarray
Angle around positive z-axis starting from x-axis.

r

1, optional
Float, int or ndarray
radius

luxpy.math.**bvgpdf**(*x*, *y=None*, *mu=None*, *sigmainv=None*)
Evaluate bivariate Gaussian probability density function (BVGPDF)

Args:

x

scalar or list or ndarray (.ndim = 1 or 2) with
x(y)-coordinates at which to evaluate bivariate Gaussian PD.

y

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

mu

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

sigmainv

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

Returns:

returns

ndarray with magnitude of BVGPDF(x,y)

luxpy.math.**mahalanobis2**(*x*, *y=None*, *z=None*, *mu=None*, *sigmainv=None*)

Evaluate the squared mahalanobis distance

Args:

x

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

y

None or scalar or list or ndarray (.ndim = 1) with y-coordinates at which to evaluate the mahalanobis distance squared, optional.

If :y: is None, :x: should be a 2d array.

z

None or scalar or list or ndarray (.ndim = 1) with z-coordinates at which to evaluate the mahalanobis distance squared, optional.

If :z: is None & :y: is None, then :x: should be a 2d array.

mu

None or ndarray (.ndim = 1) with center coordinates of the mahalanobis ellipse, optional.

None defaults to zeros(2) or zeros(3).

sigmainv

None or ndarray with 'inverse covariance matrix', optional

Determines the shape and orientation of the PD.

None default to np.eye(2) or eye(3).

Returns:

returns

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

luxpy.math.**dot23**(*A*, *B*, *keepdims=False*)

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

Args:

A

ndarray (.shape = (M,N))

B

ndarray (.shape = (N,K,L))

Returns:

returns

ndarray (.shape = (M,K,L))

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

Calculate root-mean-square along axis.

Args:

data

list of values or ndarray

axis

0, optional
Axis along which to calculate rms.

keepdims

False or True, optional
Keep original dimensions of array.

Returns:

returns

ndarray with rms values.

luxpy.math.**geomean**(*data*, *axis=0*, *keepdims=False*)
Calculate geometric mean along axis.

Args:

data

list of values or ndarray

axis

0, optional
Axis along which to calculate geomean.

keepdims

False or True, optional
Keep original dimensions of array.

Returns:

returns

ndarray with geomean values.

luxpy.math.**polyarea**(*x*, *y*)
Calculates area of polygon.

First coordinate should also be last.

Args:

x

ndarray of x-coordinates of polygon vertices.

y

ndarray of x-coordinates of polygon vertices.

Returns:

returns

float (area or polygon)

luxpy.math.**magnitude_v**(*v*)
Calculates magnitude of vector.

Args:

v

ndarray with vector

Returns:

magnitude

ndarray

luxpy.math.**angle_v1v2**(*v1*, *v2*, *htype='deg'*)
Calculates angle between two vectors.

Args:

v1

ndarray with vector 1

v2

ndarray with vector 2

htype

'deg' or 'rad', optional
Requested angle type.

Returns:

ang

ndarray

luxpy.math.**v_to_cik**(*v*, *inverse=False*)
Calculate 2x2 '(covariance matrix)^-1' elements cik

Args:

v

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

inverse

If True: return inverse of cik.

Returns:

cik

'Nx2x2' (covariance matrix)^-1

Notes:

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

luxpy.math.**cik_to_v**(*cik*, *xyc=None*, *inverse=False*)
Calculate v-format ellipse descriptor from 2x2 'covariance matrix'^-1 cik

Args:

cik

'Nx2x2' (covariance matrix)^-1

inverse

If True: input is inverse of cik.

Returns:

v

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

Notes:

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

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

Floating point modulus

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

Args:

x

angle to restrict

y

end of interval [0, y] to restrict to

Returns:

r floating point modulus

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

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

Args:

data

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

alpha

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

Return:

data

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

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

Fit an ellipse to supplied data points.

Args:

xy

coordinates of points to fit (Nx2 array)

center_on_mean_xy

False, optional
Center ellipse on mean of xy
(otherwise it might be offset due to solving
the contrained minization problem: aT*S*a, see ref below.)

Returns:

v

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

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

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

Fit covariance ellipse to xy data.

Args:

xy

coordinates of points to fit (Nx2 array)

alpha

0.05, optional
alpha significance level
(e.g alpha = 0.05 for 95% confidence ellipse)

pdf

chi2, optional
- 'chi2': Rescale using Chi2-distribution
- 't': Rescale using Student t-distribution
- 'norm': Rescale using normal-distribution
- None: don't rescale using pdf, use alpha as scalefactor (cfr. alpha* 1SD or alpha * 1SE)

SE

False, optional
If false, fit standard error ellipse at alpha significance level
If true, fit standard deviation ellipse at alpha significance level

robust

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

robust_alpha

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

Returns:

v

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

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

Args:

X

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

ni

None, optional

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

verbosity

0, optional
If 1: print results.

robust

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

robust_alpha

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

Returns:

statistic

F or chi2 value (see len(dfs))

pval

p-value

df

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

Notes:

1. If p==1: Reduces to Bartlett's test for equal variances.
2. If (ni>20).all() & (p<6) & (k<6): then a more appropriate chi2 test is used in a some cases.

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

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

Args:

X,Y

ndarrays with data.

verbosity

0, optional
If 1: print results.

Returns:

tval

statistic

pval

p-value

df

degree of freedom.

ratio

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

Note:

1. Based on Gardner, R.C. (2001). Psychological Statistics Using SPSS for Windows. New Jersey, Prentice Hall.
2. Python port from matlab code by Janne Kauttonen (https://nl.mathworks.com/matlabcentral/fileexchange/67910-pitmanmorgantest-x-y; accessed Sep 26, 2019)

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

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*, *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.`**`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 (Yw = 100) or absolute XYZ (Y = Luminance)

rfl

ndarray with spectral reflectance functions.
Will be interpolated if wavelengths do not match those of :data:

cieobs

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

K

None, optional
e.g. K = 683 lm/W for '1931_2' (relative == False)
or K = 100/sum(spd*dl) (relative == True)

out

None or 1 or 2, optional
Determines number and shape of output. (see :returns:)

cie_std_dev_obs

None or str, optional
- None: don't use CIE Standard Deviate Observer function.
- 'f1': use F1 function.

Returns:

returns

If rfl is None:
If out is None: ndarray of xyz values
(.shape = (data.shape[0],3))
If out == 1: ndarray of xyz values
(.shape = (data.shape[0],3))
If out == 2: (ndarray of xyz, ndarray of xyzw) values
Note that xyz == xyzw, with (.shape = (data.shape[0],3))
If rfl is not None:
If out is None: ndarray of xyz values
(.shape = (rfl.shape[0],data.shape[0],3))
If out == 1: ndarray of xyz values

(.shape = (rfl.shape[0]+1,data.shape[0],3))
The xyzw values of the light source spd are the first set
of values of the first dimension. The following values
along this dimension are the sample (rfl) xyz values.
If out == 2: (ndarray of xyz, ndarray of xyzw) values
with xyz.shape = (rfl.shape[0],data.shape[0],3)
and with xyzw.shape = (data.shape[0],3)

References:

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

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

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

Args:

data

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

cieobs

luxpy._CIEOBS, optional
Determines the color matching function set used in the
calculation of LER. For cieobs = '1931_2' the ybar CMF curve equals
the CIE 1924 Vlambda curve.

K

None, optional
e.g. K = 683 lm/W for '1931_2'

Returns:

ler

ndarray of LER values.

References:

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

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

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

Args:

data

ndarray with spectral data

ptype

'ru' or str, optional
str: - 'ru': in radiometric units
- 'pu': in photometric units
- 'pusa': in photometric units with Km corrected
to standard air (cfr. CIE TN003-2015)
- 'qu': in quantal energy units

cieobs

_CIEOBS or str, optional
Type of cmf set to use for photometric units.

Returns:

returns:

ndarray with normalized spectral data (SI units)

luxpy.spectrum.**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 weigthed mean of a Planckian and Daylight Phase spectrum.
Weighthing is done as described in IES TM30:
SPDreference = (Te-T)/(Te-Tb)*Planckian+(T-Tb)/(Te-Tb)*daylight
with Tb and Te are resp. the starting and end CCTs of the
mixing range and whereby the Planckian and Daylight SPDs
have been normalized for equal luminous flux.
If None: use the default specified for :ref_type:.

Can be a ndarray with shape[0] > 1, in which different mixing
ranges will be used for cct in :ccts:.

cieobs

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 their 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 weigthed mean of a Planckian and Daylight Phase spectrum.
Weighthing is done as described in IES TM30:
SPDreference = (Te-T)/(Te-Tb)*Planckian+(T-Tb)/(Te-Tb)*daylight
with Tb and Te are resp. the starting and end CCTs of the
mixing range and whereby the Planckian and Daylight SPDs
have been normalized for equal luminous flux.
If None: use the default specified for :ref_type:.
Can be a ndarray with shape[0] > 1, in which different mixing
ranges will be used for cct in :ccts:.

cieobs

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 their 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*, *cctlabels=False*, *axh=None*, *show=True*, *cspace_pars={}*, *formatstr='k-'*, *diagram_colors=False*, *diagram_samples=100*, *diagram_opacity=1.0*, *diagram_lightness=0.25*, ***kwargs*)

Plot spectrum locus for cieobs in cspace.

Args:

DL

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

BBL

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

D65

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

EEW

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

cctlabels

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

axh

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

show

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

cieobs

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

cspace

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

formatstr

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

cspace_pars

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

diagram_colors

False, optional
True: plot colored chromaticity diagram.

diagram_samples

256, optional
Sampling resolution of color space.

diagram_opacity

1.0, optional
Sets opacity of chromaticity diagram

diagram_lightness

0.25, optional
Sets lightness of chromaticity diagram

kwargs

additional keyword arguments for use with matplotlib.pyplot.

Returns:

returns

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*, *cctlabels=True*, *show=True*, *cspace_pars={}*, *formatstr='k-'*, ***kwargs*)

Plot blackbody locus.

Args:

ccts

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

cctlabels

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

axh

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

show

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

cieobs

luxpy._CIEOBS or str, optional

Determines CMF set to calculate spectrum locus or other.

cspace

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

formatstr

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

cspace_pars

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

kwargs

additional keyword arguments for use with matplotlib.pyplot.

Returns:

returns

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

Plot one or more concentric circles.

Args:

center

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

radii

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

angles

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

color

'k', optional
Color for plotting.

linestyle

'–', optional
Linestyle of circles.

out

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

luxpy.color.utils.**plotellipse**(*v*, *cspace_in='Yxy'*, *cspace_out=None*, *nsamples=100*, *show=True*, *axh=None*, *line_color='darkgray'*, *line_style=':'*, *line_width=1*, *line_marker=''*, *line_markersize=4*, *plot_center=False*, *center_marker='o'*, *center_color='darkgray'*, *center_markersize=4*, *show_grid=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*, *diagram_opacity=1.0*, *diagram_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+abs(wavelength_height)*spdmax)

wavelength_opacity

1.0, optional
Sets opacity of wavelength rectangle.

wavelength_lightness

1.0, optional
Sets lightness of wavelength rectangle.

wavelength_height

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

axh

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

show

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

cieobs

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

show_grid

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) <-> (Jz,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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, ***kwargs*)

Convert XYZ tristimulus values CIE 1964 U*V*W* color space.

Args:

xyz

ndarray with tristimulus values

xyzw

ndarray with tristimulus values of white point, optional
(Defaults to luxpy._COLORTF_DEFAULT_WHITE_POINT)

Returns:

wuv

ndarray with W*U*V* values

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

Convert CIE 1964 U*V*W* color space coordinates to XYZ tristimulus values.

Args:

wuv

ndarray with W*U*V* values

xyzw

ndarray with tristimulus values of white point, optional
(Defaults to luxpy._COLORTF_DEFAULT_WHITE_POINT)

Returns:

xyz

ndarray with tristimulus values

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

Convert XYZ tristimulus values to XYZ tristimulus values.

Args:

xyz

ndarray with tristimulus values

Returns:

xyz

ndarray with tristimulus values

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

Convert XYZ tristimulus values to LMS cone fundamental responses.

Args:

xyz

ndarray with tristimulus values

cieobs

_CIEOBS or str, optional

M

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

Returns:

lms

ndarray with LMS cone fundamental responses

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

Convert LMS cone fundamental responses to XYZ tristimulus values.

Args:

lms

ndarray with LMS cone fundamental responses

cieobs

_CIEOBS or str, optional

M

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

Returns:

xyz

ndarray with tristimulus values

luxpy.color.ctf.colortransforms.**xyz_to_lab**(*xyz*, *xyzw=None*, *cieobs='1931_2'*, ***kwargs*)
Convert XYZ tristimulus values to CIE 1976 L*a*b* (CIELAB) coordinates.

Args:

xyz

ndarray with tristimulus values

xyzw

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

cieobs

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

Returns:

lab

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

luxpy.color.ctf.colortransforms.**lab_to_xyz**(*lab*, *xyzw=None*, *cieobs='1931_2'*, ***kwargs*)
Convert CIE 1976 L*a*b* (CIELAB) color coordinates to XYZ tristimulus values.

Args:

lab

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

xyzw

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

cieobs

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

Returns:

xyz

ndarray with tristimulus values

luxpy.color.ctf.colortransforms.**xyz_to_luv**(*xyz*, *xyzw=None*, *cieobs='1931_2'*, ***kwargs*)
Convert XYZ tristimulus values to CIE 1976 L*u*v* (CIELUV) coordinates.

Args:

xyz

ndarray with tristimulus values

xyzw

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

cieobs

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

Returns:

luv

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

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

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

Args:

luv

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

xyzw

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

cieobs

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

Returns:

xyz

ndarray with tristimulus values

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

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

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

Args:

xyz

ndarray with tristimulus values

cieobs

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

scaling

list of scaling factors for r and b dimensions.

M

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

Returns:

Vrb

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

Reference:

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

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

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

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

Args:

Vrb

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

cieobs

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

scaling

list of scaling factors for r and b dimensions.

M

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

Minverted

False, optional
Bool that determines whether M should be inverted.

Returns:

xyz

ndarray with tristimulus values

Reference:

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

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

Convert XYZ tristimulus values to IPT color coordinates.

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

Args:

xyz

ndarray with tristimulus values

xyzw

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

cieobs

luxpy._CIEOBS, optional
CMF set to use when calculating xyzw for rescaling M
(only when not None).

M

None, optional
None defaults to xyz to lms conversion matrix determined by :cieobs:

Returns:

ipt

ndarray with IPT color coordinates

Note:

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

Reference:

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

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

Convert XYZ tristimulus values to IPT color coordinates.

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

Args:

ipt

ndarray with IPT color coordinates

xyzw

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

cieobs

luxpy._CIEOBS, optional
CMF set to use when calculating xyzw for rescaling Mxyz2lms
(only when not None).

M

None, optional
None defaults to xyz to lms conversion matrix determined by:cieobs:

Returns:

xyz

ndarray with tristimulus values

Note:

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

Reference:

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

luxpy.color.ctf.colortransforms.**xyz_to_Ydlep**(*xyz*, *cieobs='1931_2'*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *flip_axes=False*, ***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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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 narray with tristimulus values of a single (!) native white point, optional
None defaults to xyz of CIE D65 using the :cieobs: observer.

cieobs

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

flip_axes

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

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 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 cd/m^2
(note that Jz that not exactly equal 1 for this high value, but rather for 102900 cd/m2)

2b. az, bz represent respectively a red-green and a yellow-blue opponent axis
(but note that a D65 shows a small offset from (0,0))

Reference: 1. Safdar, M., Cui, G., Kim,Y. J., and Luo, M. R. (2017). Perceptually uniform color space for image signals including high dynamic range and wide gamut. Opt. Express, vol. 25, no. 13, pp. 15131–15151, 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 cd/m^2
(note that Jz that not exactly equal 1 for this high value, but rather for 102900 cd/m2)

2b. az, bz represent respectively a red-green and a yellow-blue opponent axis
(but note that a D65 shows a small offset from (0,0))

Reference: 1. Safdar, M., Cui, G., Kim,Y. J., and Luo, M. R. (2017). Perceptually uniform color space for image signals including high dynamic range and wide gamut. Opt. Express, vol. 25, no. 13, pp. 15131–15151, 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 fuond 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 (temprorary) 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 calculated 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 calculated 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*, *upper_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 = numpy.sqrt(((100.0*(cct_min - cct)/(cct))**2.0)
+ (((duv_min - duv)/(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 calculated 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*, *upper_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 calculated 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=1000000000000.0*, *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 calculated 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*, *upper_cct_max=1000000000000.0*, *approx_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 calculated 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*, *upper_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 calculated 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 mccamy 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-Estevez: R. W. G. Hunt, The Reproduction of Colour: Sixth Edition, 6th ed. Chichester, UK: John Wiley & Sons Ltd, 2004.
- 'cat02': from ciecam02: CIE159-2004, "A Colour Apperance Model for Color Management System: CIECAM02," CIE, Vienna, 2004.
- 'cat02-bs': cat02 adjusted to solve yellow-blue problem (last line = [0 0 1]): Brill MH, Süsstrunk S. Repairing gamut problems in CIECAM02: A progress report. Color Res Appl 2008;33(5), 424–426.
- 'cat02-jiang': cat02 modified to solve yb-probem + purple problem: Jun Jiang, Zhifeng Wang,M. Ronnier Luo,Manuel Melgosa,Michael H. Brill,Changjun Li, Optimum solution of the CIECAM02 yellow–blue and purple problems, Color Res Appl 2015: 40(5), 491-503.
- 'kries'
- 'judd-1945': from CIE16-2004, Eq.4, a23 modified from 0.1 to 0.1020 for increased accuracy
- 'bfd': bradford transform : G. D. Finlayson and S. Susstrunk, "Spectral sharpening and the Bradford transform," 2000, vol. Proceeding, pp. 236–242.
- 'sharp': sharp transform: S. Süsstrunk, J. Holm, and G. D. Finlayson, "Chromatic adaptation performance of different RGB sensors," IS&T/SPIE Electronic Imaging 2001: Color Imaging, vol. 4300. San Jose, CA, January, pp. 172–183, 2001.
- 'cmc': C. Li, M. R. Luo, B. Rigg, and R. W. G. Hunt, "CMC 2000 chromatic adaptation transform: CMCCAT2000," Color Res. Appl., vol. 27, no. 1, pp. 49–58, 2002.
- 'ipt': F. Ebner and M. D. Fairchild, "Development and testing of a color space (IPT) with improved hue uniformity," in IS&T 6th Color Imaging Conference, 1998, pp. 8–13.
- 'lms':
- 'bianco': S. Bianco and R. Schettini, "Two new von Kries based chromatic adaptation transforms found by numerical optimization," Color Res. Appl., vol. 35, no. 3, pp. 184–192, 2010.
- 'bianco-pc': S. Bianco and R. Schettini, "Two new von Kries based chromatic adaptation transforms found by numerical optimization," Color Res. Appl., vol. 35, no. 3, pp. 184–192, 2010.
- 'cat16': C. Li, Z. Li, Z. Wang, Y. Xu, M. R. Luo, G. Cui, M. Melgosa, M. H. Brill, and M. Pointer, "Comprehensive color solutions: CAM16, CAT16, and CAM16-UCS," Color Res. Appl., p. n/a–n/a.

check_dimensions() Check if dimensions of data and xyzw match.

get_transfer_function()

Calculate the chromatic adaptation diagonal matrix transfer function Dt.

Default = 'vonkries' (others: 'rlab', see Fairchild 1990)

smet2017_D()

Calculate the degree of adaptation based on chromaticity.

Smet, K.A.G.*, Zhai, Q., Luo, M.R., Hanselaer, P., (2017), Study of chromatic adaptation using memory color matches, Part II: colored illuminants. Opt. Express, 25(7), pp. 8350-8365

get_degree_of_adaptation()

Calculates the degree of adaptation.

D passes either right through or D is calculated following some D-function (Dtype) published in literature (cat02, cat16, cmccat, smet2017) or set manually.

parse_x1x2_parameters() local helper function that parses input parameters and makes them the target_shape for easy calculation

apply() Calculate corresponding colors by applying a von Kries chromatic adaptation transform (CAT), i.e. independent rescaling of 'sensor sensitivity' to data to adapt from current adaptation conditions (1) to the new conditions (2).

`luxpy.color.cat.`**`check_dimensions`**(*data*, *xyzw*, *caller='cat.apply()'*)

Check if dimensions of data and xyzw match.

Does nothing when they do, but raises error if dimensions don't match.

Args:

data

ndarray with color data.

xyzw

ndarray with white point tristimulus values.

caller

str with caller function for error handling, optional

Returns:

returns

ndarray with input color data,

Raises error if dimensions don't match.

`luxpy.color.cat.`**`get_transfer_function`**(*cattype='vonkries'*, *catmode='1>0>2'*, *lmsw1=None*, *lmsw2=None*, *lmsw0=array([[100, 100, 100]])*, *D10=1.0*, *D20=1.0*, *La1=100.0*, *La2=100.0*, *La0=100.0*)

Calculate the chromatic adaptation diagonal matrix transfer function Dt.

Args:

cattype

'vonkries' (others: 'rlab', see Farchild 1990), optional

catmode

'1>0>2, optional

-'1>0>2': Two-step CAT

from illuminant 1 to baseline illuminant 0 to illuminant 2.
-'1>0': One-step CAT
from illuminant 1 to baseline illuminant 0.
-'0>2': One-step CAT
from baseline illuminant 0 to illuminant 2.

lmsw1

None, depending on :catmode: optional

lmsw2

None, depending on :catmode: optional

lmsw0

_WHITE_POINT, optional

D10

1.0, optional
Degree of adaptation for ill. 1 to ill. 0

D20

1.0, optional
Degree of adaptation for ill. 2 to ill. 0

La1

luxpy._LA, optional
Adapting luminance under ill. 1

La2

luxpy._LA, optional
Adapting luminance under ill. 2

La0

luxpy._LA, optional
Adapting luminance under baseline ill. 0

Returns:

Dt

ndarray (diagonal matrix)

luxpy.color.cat.**get_degree_of_adaptation**(*Dtype=None*, ***kwargs*)

Calculates the degree of adaptation according to some function published in literature.

Args:

Dtype

None, optional
If None: kwargs should contain 'D' with value.
If 'manual: kwargs should contain 'D' with value.
If 'cat02' or 'cat16': kwargs should contain keys 'F' and 'La'.
Calculate D according to CAT02 or CAT16 model:
D = F*(1-(1/3.6)*numpy.exp((-La-42)/92))
If 'cmc': kwargs should contain 'La', 'La0'(or 'La2') and 'order'
for 'order' = '1>0': 'La' is set La1 and 'La0' to La0.
for 'order' = '0>2': 'La' is set La0 and 'La0' to La1.

for 'order' = '1>2': 'La' is set La1 and 'La2' to La0.
D is calculated as follows:
D = 0.08*numpy.log10(La1+La0)+0.76-0.45*(La1-La0)/(La1+La0)
If 'smet2017': kwargs should contain 'xyzw' and 'Dmax'
(see Smet2017_D for more details).
If "? user defined", then D is calculated by:
D = ndarray(eval(:Dtype:))

Returns:

D

ndarray with degree of adaptation values.

Notes:

1. D passes either right through or D is calculated following some D-function (Dtype) published in literature.
2. D is limited to values between zero and one
3. If kwargs do not contain the required parameters, an exception is raised.

luxpy.color.cat.**smet2017_D**(*xyzw*, *Dmax=None*, *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~50Â°) of 760 cd/mÂ^2))

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()

mcat

['cat02'], optional
List[str] or List[ndarray] of sensor space matrices for each
condition pair. If len(:mcat:) == 1, the same matrix is used.

normxyz0

None, optional
Set of xyz tristimulus values to normalize the sensor space matrix to.

outtype

'xyz' or 'lms', optional
- 'xyz': return corresponding tristimulus values
- 'lms': return corresponding sensor space excitation values
(e.g. for further calculations)

Returns:

returns

ndarray with corresponding colors

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) = sign(x) * scaling * ((abs(x)**n) / ((abs(x)**n) + (sig**n))) + 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) = sign(x) * scaling * ((abs(x)**n) / ((abs(x)**n) + (sig**n))) + noise

Args:

data

float or ndarray

sig

2.0, optional
Semi-saturation constant. Value for which NK(:data:) is 1/2

n

0.73, optional
Compression power.

scaling

1.0, optional
Maximum value of NK-function.

noise

0.0, optional
Cone excitation noise.

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([[1.0000e+02*, *1.0000e+02*, *1.0000e+02]])*, *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 tristsimulus 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 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

'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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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 tristsimulus 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 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 back 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 -> 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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *direction='forward'*, *ucstype='ucs'*, *yellowbluepurplecorrect=False*, *mcat='cat02'*)

Convert between XYZ tristsimulus 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 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 -> 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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *direction='forward'*, *ucstype='ucs'*, *mcat='cat16'*)

Convert between XYZ tristsimulus 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 -> cam15u

-'inverse': cam15u -> xyz

outin

'Q,aW,bW' or str, optional

'Q,aW,bW' (brightness and opponent signals for amount-of-neutral)

other options: 'Q,aM,bM' (colorfulness) and 'Q,aS,bS' (saturation)

Str specifying the type of

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

parameters

None or dict, optional
Set of model parameters.
- None: defaults to luxpy.cam._CAM15U_PARAMETERS
(see references below)

Returns:

returns

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

References: 1. M. Withouck, K. A. G. Smet, W. R. Ryckaert, and P. Hanselaer, "Experimental driven modelling of the color appearance of unrelated self-luminous stimuli: CAM15u," Opt. Express, vol. 23, no. 9, pp. 12045–12064, 2015. 2. M. Withouck, K. A. G. Smet, and P. Hanselaer, (2015), "Brightness prediction of different sized unrelated self-luminous stimuli," Opt. Express, vol. 23, no. 10, pp. 13455–13466.

luxpy.color.cam.**cam_sww16**(*data*, *dataw=None*, *Yb=20.0*, *Lw=400.0*, *Ccwb=None*, *relative=True*, *inputtype='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, Yw = 100)

Lw

400.0, optional
Luminance (cd/m^2) of white point.

Ccwb

None, optional
Degree of cognitive adaptation (white point balancing)

If None: use [..,..] from parameters dict.

relative

True or False, optional
True: xyz tristimulus values are relative (Yw = 100)

parameters

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 channging 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^2) value(s) of background(s) calculated using the CIE 2006 10° CMFs
(only used in case datab == None and the background is assumed to be an Equal-Energy-White)

fov

10.0, optional
Field-of-view of stimulus (for size effect on brightness)

inputtpe

'xyz' or 'spd', optional
Specifies the type of input:
tristimulus values or spectral data for the forward mode.

direction

'forward' or 'inverse', optional
-'forward': xyz -> cam18sl
-'inverse': cam18sl -> xyz

outin

'Q,aS,bS' or str, optional
'Q,aS,bS' (brightness and opponent signals for saturation)
other options: 'Q,aM,bM' (colorfulness)
(Note that 'Q,aW,bW' would lead to a Cartesian
a,b-coordinate system centered at (1,0))
Str specifying the type of
input (:direction: == 'inverse') and
output (:direction: == 'forward')

parameters

None or dict, optional

Set of model parameters.
- None: defaults to luxpy.cam._CAM18SL_PARAMETERS
(see references below)

Returns:

returns

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

Notes:

* Instead of using the CIE 1964 10° CMFs in some places of the model,
the CIE 2006 10° CMFs are used througout, making it more self_consistent.
This has an effect on the k scaling factors (now different those in CAM15u)
and the illuminant E normalization for use in the chromatic adaptation transform.
(see future erratum to Hermans et al., 2018)
* The paper also used an equation for the amount of white W, which is
based on a Q value not expressed in 'bright' ('cA' = 0.937 instead of 123).
This has been corrected for in the luxpy version of the model, i.e.
_CAM18SL_PARAMETERS['cW'][0] has been changed from 2.29 to 1/11672.
(see future erratum to Hermans et al., 2018)
* Default output was 'Q,aW,bW' prior to March 2020, but since this
is an a,b Cartesian system centered on (1,0), the default output
has been changed to 'Q,aS,bS'.

References: 1. Hermans, S., Smet, K. A. G., & Hanselaer, P. (2018). "Color appearance model for self-luminous stimuli." Journal of the Optical Society of America A, 35(12), 2000–2009.

luxpy.color.cam.**xyz_to_jabM_ciecam02**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *yellowbluepurplecorrect=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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *yellowbluepurplecorrect=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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *yellowbluepurplecorrect=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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *yellowbluepurplecorrect=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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': '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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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,aM,bM input.

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

luxpy.color.cam.**xyz_to_jabC_cam16**(*data*, *xyzw=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Yw=None*, *conditions={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': '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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *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
- frieleellipses.py
- macadamellipses.py

namespace luxpy.deltaE

Module for color difference calculations

process_DEi() Process color difference input DEi for output (helper fnc).

DE_camucs() Calculate color appearance difference DE using camucs type model.

DE_2000() Calculate DE2000 color difference.

DE_cspace() Calculate color difference DE in specific color space.

get_macadam_ellipse() Estimate n-step MacAdam ellipse at CIE x,y coordinates

get_gij_fmc() Get gij matrices describing the discrimination ellipses for Yxy using FMC-1 or FMC-2.

get_fmc_discrimination_ellipse() Get n-step discrimination ellipse(s) in v-format (R,r, xc, yc, theta) for Yxy using FMC-1 or FMC-2.

`luxpy.color.deltaE.`**`deltaH`**(*h1*, *C1*, *h2=None*, *C2=None*, *htype='deg'*)

Compute a hue difference, dH = 2*C1*C2*sin(dh/2)

Args:

h1

hue for sample 1 (or hue difference if h2 is None)

C1

chroma of sample 1 (or prod C1*C2 if C2 is None)

h2

hue angle of sample 2 (if None, then h1 contains a hue difference)

C2

chroma of sample 2

htype

'deg' or 'rad', optional
- 'deg': hue angle between 0° and 360°
- 'rad': hue angle between 0 and 2pi radians

Returns:

returns

ndarray of deltaH values.

luxpy.color.deltaE.**DE_camucs**(*xyzt*, *xyzr*, *DEtype='jab'*, *avg=None*, *avg_axis=0*, *out='DEi'*, *xyzwt=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *xyzwr=array([[1.0000e+02, 1.0000e+02, 1.0000e+02]])*, *Ywt=None*, *conditionst={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *Ywr=None*, *conditionsr={'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}*, *camtype='ciecam02'*, *ucstype='ucs'*, *mcat=None*, *outin='J, aM, bM'*, *yellowbluepurplecorrect=False*, ***kwargs*)

Calculate color appearance difference DE using camucs type model.

Args:

xyzt

ndarray with tristimulus values of test data.

xyzr

ndarray with tristimulus values of reference data.

DEtype

'jab' or str, optional

Options:

- 'jab' : calculates full color difference over all 3 dimensions.
- 'ab' : calculates chromaticity difference.
- 'j' : calculates lightness or brightness difference (depending on :outin:).
- 'j,ab': calculates both 'j' and 'ab' options and returns them as a tuple.

avg

None, optional

None: don't calculate average DE, otherwise use function handle in :avg:.

avg_axis

axis to calculate average over, optional

out

'DEi' or str, optional

Requested output.

camtype

luxpy.cam._CAM_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
Weigths for L, C, H
None: default to [1,1,1]

avg

None, optional
None: don't calculate average DE,
otherwise use function handle in :avg:.

avg_axis

axis to calculate average over, optional

out

'DEi' or str, optional
Requested output.

Note: For the other input arguments, see specific color space used.

Returns:

returns

ndarray with DEi [, DEa] or other as specified by :out:

References: 1. Sharma, G., Wu, W., & Dalal, E. N. (2005). The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. Color Research & Application, 30(1), 21–30.

`luxpy.color.deltaE.`**`DE_cspace`**(*xyzt*, *xyzr*, *dtype='xyz'*, *tf='Yuv'*, *DEtype='jab'*, *avg=None*, *avg_axis=0*, *out='DEi'*, *xyzwt=None*, *xyzwr=None*, *fwtft={}*, *fwtfr={}*, *KLCH=None*, *camtype='ciecam02'*, *ucstype='ucs'*)

Calculate color difference DE in specific color space.

Args:

xyzt

ndarray with tristimulus values of test data.

xyzr

ndarray with tristimulus values of reference data.

dtype

'xyz' or 'jab', optional
Specifies data type in :xyzt: and :xyzr:.

xyzwt

None or ndarray, optional
White point tristimulus values of test data
None defaults to the one set in :fwtft:
or else to the default of cspace.

xyzwr

None or ndarray, optional
Whitepoint tristimulus values of reference data
None defaults to the one set in non-empty :fwtfr:
or else to default of cspace.

tf

_CSPACE, optional
Color space to use for color difference calculation.

fwtft

{}, optional
Dict with parameters for forward transform from xyz to cspace for test data.

fwtfr

{}, optional
Dict with parameters for forward transform
from xyz to cspace for reference data.

KLCH

None, optional
Weigths for L, C, H
None: default to [1,1,1]

KLCH is not used when tf == 'camucs'.

DEtype

'jab' or str, optional
Options:
- 'jab' : calculates full color difference over all 3 dimensions.
- 'ab' : calculates chromaticity difference.
- 'j' : calculates lightness or brightness difference
  (depending on :outin:).
- 'j,ab': calculates both 'j' and 'ab' options
  and returns them as a tuple.

avg

None, optional
None: don't calculate average DE,
otherwise use function handle in :avg:.

avg_axis

axis to calculate average over, optional

out

'DEi' or str, optional
Requested output.

camtype

luxpy.cam._CAM_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([[1.0000e+02, 3.3333e-01, 3.3333e-01]])*, *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([[1.0000e+02, 3.3333e-01, 3.3333e-01]])*, *etype='fmc2'*, *Y=None*, *nsteps=10*)

Get discrimination ellipse(s) in v-format (R,r, xc, yc, theta) for Yxy using FMC-1 or FMC-2.

Args:

Yxy

2D ndarray with [Y,]x,y coordinate centers.
If Yxy.shape[-1]==2: Y is added using the value from the Y-input argument.

etype

'fmc2', optional
Type of FMC color discrimination equations to use (see references below).
options: 'fmc1', fmc2'

Y

None, optional
Only affects FMC-2 (see note below).
If not None: Y = 10.69 and overrides values in Yxy.

nsteps

10, optional
Set multiplication factor for ellipses
(nsteps=1 corresponds to approximately 1 MacAdam step,
for FMC-2, Y also has to be 10.69, see note below).

Note:

1. FMC-2 is almost identical to FMC-1 is Y = 10.69!; see [2]

References:

1. Chickering, K.D. (1967), Optimization of the MacAdam-Modified 1965 Friele Color-Difference Formula, 57(4), p.537-541
2. Chickering, K.D. (1971), FMC Color-Difference Formulas: Clarification Concerning Usage, 61(1), p.118-122

luxpy.color.deltaE.**discrimination_hotelling_t2**(*Yxy1*, *Yxy2*, *etype='fmc2'*, *ellipsoid=True*, *Y1=None*, *Y2=None*, *cspace='Yxy'*)

Check 'significance' of difference using Hotelling's T2 test on the centers Yxy1 and Yxy2 and their associate FMC-1/2 discrimination ellipses.

Args:

Yxy1, Yxy2

2D ndarrays with [Y,]x,y coordinate centers.
If Yxy.shape[-1]==2: Y is added using the value from the Y-input argument.

etype

'fmc2', optional
Type of FMC color discrimination equations to use (see references below).
options: 'fmc1', fmc2'

Y1, Y2

None, optional
Only affects FMC-2 (see note below).
If not None: Yi = 10.69 and overrides values in Yxyi.

ellipsoid

True, optional
If True: return ellipsoids, else return ellipses (only if cspace == 'Yxy')!

cspace

'Yxy', optional
Return coefficients for Yxy-ellipses/ellipsoids ('Yxy') or XYZ ellipsoids ('xyz')

Returns:

p

Chi-square based p-value

T2

T2 test statistic (= mahalanobis distance on summed standard error cov. matrices)

Steps: 1. For each center coordinate, the standard error covariance matrix gij^-1 = Si/ni is determined using the FMC-1 or FMC-2 equations (see refs. 1 & 2). 2. Calculate sum of covariance matrices: SIG = S1/n1 + S2/n2 = gij1^-1 + gij2^-1 3. These are then used in Hotelling's T2 test: T2 = (xy1 - xy2).T*(SIG^-1)*(xy1_xy2) 4. The T2 statistic is then tested against a Chi-square distribution with 2 or 3 degrees of freedom.

References:

1. Chickering, K.D. (1967), Optimization of the MacAdam-Modified 1965 Friele Color-Difference Formula, 57(4):537-541
2. Chickering, K.D. (1971), FMC Color-Difference Formulas: Clarification Concerning Usage, 61(1):118-122

4.4.7 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^2), 'L1000' (1000 cd/m^2) and 'L2000' (2000 cd/m^2).

Returns:

N

ndarray with calculated neutrality

References: 1. Smet, K., Deconinck, G., & Hanselaer, P., (2014), Chromaticity of unique white in object mode. Optics Express, 22(21), 25830–25841.

2. Smet, K.A.G., Zhai, Q., Luo, M.R., Hanselaer, P., (2017), Study of chromatic adaptation using memory color matches, Part II: colored illuminants, Opt. Express, 25(7), pp. 8350-8365.

`luxpy.color.whiteness.`**`cct_to_neutral_loci_smet2018`**(*cct*, *nlocitype='uw'*, *out='duv,D'*)

Calculate the most neutral appearing Duv10 in and the degree of neutrality for a specified CCT using the models in Smet et al. (2018).

Args:

cct10

ndarray CCT

nlocitype

'uw', optional
'uw': use unique white models published in Smet et al. (2014).
'ca': use degree of chromatic adaptation model from Smet et al. (2017).

out

'duv,D', optional
Specifies requested output (other options: 'duv', 'D').

Returns:

duv

ndarray with most neutral Duv10 value corresponding to the cct input.

D

ndarray with the degree of neutrality at (cct, duv).

References: 1. Smet, K.A.G., (2018), Two Neutral White Illumination Loci Based on Unique White Rating and Degree of Chromatic Adaptation. LEUKOS, 14(2), 55–67.

Notes:

1. Duv is specified in the CIE 1960 u10v10 chromatity diagram as the models were developed using CIE 1964 10° tristimulus, chromaticity and CCT values.
2. The parameter +0.0172 in Eq. 4b should be -0.0172.

4.4.8 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 suppporting color rendition calculations (colorrendition.py)

utils/init_cri_defaults_database.py

_CRI_TYPE_DEFAULT Default cri_type.

_CRI_DEFAULTS

default parameters for color fidelity and gamut area metrics (major dict has 9 keys (04-Jul-2017): sampleset [str/dict], ref_type [str], cieobs [str], avg [fcn handle], scale [dict], cspace [dict], catf [dict], rg_pars [dict], cri_specific_pars [dict])

- **Supported cri-types:**
  - 'ciera','ciera-8','ciera-14','cierf',
  - 'iesrf','iesrf-tm30-15','iesrf-tm30-18',
  - '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:
Rfi,a = 100 - c1*DEi,a. (c1 = 4.6)

log_scale()

Log-based color rendering index scale from Davis & Ohno (2009):
Rfi,a = 10 * ln(exp((100 - c1*DEi,a)/10) + 1)

psy_scale()

Psychometric based color rendering index scale from Smet et al. (2013):
Rfi,a = 100 * (2 / (exp(c1*abs(DEi,a)**(c2) + 1))) ** c3

utils/helpers.py

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, Rcshi, Rhshi, cct, duv, ...

utils/graphics.py

plot_hue_bins() Makes basis plot for Color Vector Graphic (CVG).

plot_ColorVectorGraphic() Plots Color Vector Graphic (see IES TM30).

indices/indices.py

wrapper_functions_for_fidelity_type_metrics

spd_to_ciera(): CIE 13.3 1995 version
spd_to_ciera_133_1995(): CIE 13.3 1995 version
spd_to_cierf(): latest version
spd_to_cierf_224_2017(): CIE224-2017 version

spd_to_iesrf(): latest version
spd_to_iesrf_tm30(): latest version
spd_to_iesrf_tm30_15(): TM30-15 version
spd_to_iesrf_tm30_18(): TM30-18 version

spd_to_cri2012()
spd_to_cri2012_hl17()
spd_to_cri2012_hl1000()
spd_to_cri2012_real210()

wrapper_functions_for_gamut_area_metrics

spd_to_iesrg(): latest version
spd_to_iesrg_tm30(): latest version
spd_to_iesrg_tm30_15(): TM30-15 version
spd_to_iesrg_tm30_18(): TM30-18 version

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:

Rfi,a = 100 - c1*DEi,a. (c1 = 4.6)

Args:

data

float or list[floats] or ndarray

scale_factor

[4.6] or list[float] or ndarray, optional
Rescales color differences before subtracting them from :scale_max:

scale_max

100.0, optional
Maximum value of linear scale

Returns:

returns

float or list[floats] or ndarray

References: 1. CIE13.3-1995, "Method of Measuring and Specifying Colour Rendering Properties of Light Sources," CIE, Vienna, Austria, 1995.,ISBN 978 3 900734 57 2

luxpy.color.cri.**log_scale**(*data, scale_factor=[6.73], scale_max=100.0*)

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

Rfi,a = 10 * ln(exp((100 - c1*DEi,a)/10) + 1).

Args:

data

float or list[floats] or ndarray

scale_factor

[6.73] or list[float] or ndarray, optional
Rescales color differences before subtracting them from :scale_max:
Note that the default value is the one from cie-224-2017.

scale_max

100.0, optional
Maximum value of linear scale

Returns:

returns

float or list[floats] or ndarray

References: 1. W. Davis and Y. Ohno, "Color quality scale," (2010), Opt. Eng., vol. 49, no. 3, pp. 33602–33616. 2. CIE224:2017. CIE 2017 Colour Fidelity Index for accurate scientific use. Vienna, Austria: CIE. (2017).

luxpy.color.cri.**psy_scale**(*data, scale_factor=[0.01818181818181818, 1.5, 2.0], scale_max=100.0*)

Psychometric based color rendering index scale from CRI2012:

Rfi,a = 100 * (2 / (exp(c1*abs(DEi,a)**(c2) + 1))) ** c3.

Args:

data

float or list[floats] or ndarray

scale_factor

[1/55, 3/2, 2.0] or list[float] or ndarray, optional
Rescales color differences before subtracting them from :scale_max:

Note that the default value is the one from (Smet et al. 2013, LRT).

scale_max

100.0, optional
Maximum value of linear scale

Returns:

returns

float or list[floats] or ndarray

References: 1. Smet, K., Schanda, J., Whitehead, L., & Luo, R. (2013). CRI2012: A proposal for updating the CIE colour rendering index. Lighting Research and Technology, 45, 689–709.

`luxpy.color.cri.`**`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 radiatiors,
'DL': daylightphase,
'ciera': used in CIE CRI-13.3-1995,
'cierf': used in CIE 224-2017,
'iesrf': used in TM30-15, ...
- ndarray: user defined reference SPD

cieobs

None or dict, optional
Specifies which CMF sets to use for the calculation of the sample
XYZs and the CCT (for reference illuminant calculation).
None defaults to the one specified in :cri_type: dict.
- key: 'xyz': str specifying CMF set for calculating xyz
of samples and white
- key: 'cct': str specifying CMF set for calculating cct

cspace

None or dict, optional
Specifies which color space to use.
None defaults to the one specified in :cri_type: dict.
- key: 'type': str specifying color space used to calculate
color differences in.
- key: 'xyzw': None or ndarray with white point of color space
If None: use xyzw of test / reference (after chromatic
adaptation, if specified)
- other keys specify other possible parameters needed for color
space calculation,
see lx.cri._CRI_DEFAULTS['iesrf']['cspace'] for details.

catf

None or dict, optional
Perform explicit CAT before converting to color space coordinates.
- None: don't apply a cat (other than perhaps the one built

into the colorspace)
- dict: with CAT parameters:
- key: 'D': ndarray with degree of adaptation
- key: 'mcat': ndarray with sensor matrix specification
- key: 'xyzw': None or ndarray with white point
None: use xyzw of reference otherwise transform both
test and ref to xyzw

cri_specific_pars

None or dict, optional
Specifies other parameters specific to type of cri
(e.g. maxC for CQS calculations)
- None: default to the one specified in :cri_type: dict.
- dict: user specified parameters.
For its use, see for example:
luxpy.cri._CRI_DEFAULTS['mcri']['cri_specific_pars']

Returns:

returns

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

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

`luxpy.color.cri.`**`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 radiatiors,
'DL': daylightphase,
'ciera': used in CIE CRI-13.3-1995,
'cierf': used in CIE 224-2017,
'iesrf': used in TM30-15, ...
- ndarray: user defined reference SPD

cieobs

None or dict, optional
Specifies which CMF sets to use for the calculation of the sample
XYZs and the CCT (for reference illuminant calculation).
None defaults to the one specified in :cri_type: dict.
- key: 'xyz': str specifying CMF set for calculating xyz
of samples and white
- key: 'cct': str specifying CMF set for calculating cct

cspace

None or dict, optional
Specifies which color space to use.
None defaults to the one specified in :cri_type: dict.
- key: 'type': str specifying color space used to calculate
color differences in.
- key: 'xyzw': None or ndarray with white point of color space
If None: use xyzw of test / reference (after chromatic
adaptation, if specified)
- other keys specify other possible parameters needed for color
space calculation,
see lx.cri._CRI_DEFAULTS['iesrf']['cspace'] for details.

catf

None or dict, optional
Perform explicit CAT before converting to color space coordinates.
- None: don't apply a cat (other than perhaps the one built
into the colorspace)
- dict: with CAT parameters:

- key: 'D': ndarray with degree of adaptation
- key: 'mcat': ndarray with sensor matrix specification
- key: 'xyzw': None or ndarray with white point
None: use xyzw of reference otherwise transform both
test and ref to xyzw

cri_specific_pars

None or dict, optional
Specifies other parameters specific to type of cri
(e.g. maxC for CQS calculations)
- None: default to the one specified in :cri_type: dict.
- dict: user specified parameters.
For its use, see for example:
luxpy.cri._CRI_DEFAULTS['mcri']['cri_specific_pars']

rg_pars

None or dict, optional
Dict containing specifying parameters for slicing the gamut.
Dict structure:
{'nhbins' : None, 'start_hue' : 0,
'normalize_gamut' : False, 'normalized_chroma_ref': 100.0}
- key: 'nhbins': int, number of hue bins to slice gamut
(None use the one specified in :cri_type: dict).
- key: 'start_hue': float (°), hue at which to start slicing
- key: 'normalize_gamut': True or False:
normalize gamut or not before calculating a gamut
area index Rg.
- key: 'normalized_chroma_ref': 100.0 or float, optional
Controls the size (chroma/radius)
of the normalization circle/gamut.

avg

None or fcn handle, optional
Averaging function (handle) for color differences, DEi
(e.g. numpy.mean, .math.rms, .math.geomean)
None use the one specified in :cri_type: dict.

scale

None or dict, optional
Specifies scaling of color differences to obtain CRI.
- None use the one specified in :cri_type: dict.
- dict: user specified dict with scaling parameters.
- key: 'fcn': function handle to type of cri scale,
e.g.
* linear()_scale –> (100 - scale_factor*DEi),
* log_scale –> (cfr. Ohno's CQS),
* psy_scale (Smet et al.'s cri2012,See: LRT 2013)
- key: 'cfactor': factors used in scaling function,

If None:

Scaling factor value(s) will be optimized to minimize the rms between the Rf's of the requested metric and the target metric specified in:

- key: 'opt_cri_type': str
  * str: one of the preset _CRI_DEFAULTS
  * dict: user speciied
  (dict must contain all keys as normal)
  Note that if key not in :scale: dict, then 'opt_cri_type' is added with default setting = 'ciera'.
- key: 'opt_spd_set': ndarray with set of light source spds used to optimize cfactor.
  Note that if key not in :scale: dict, then default = 'F1-F12'.

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 radiatiors,
'DL': daylightphase,
'ciera': used in CIE CRI-13.3-1995,
'cierf': used in CIE 224-2017,
'iesrf': used in TM30-15, ...
- ndarray: user defined reference SPD

cieobs

None or dict, optional
Specifies which CMF sets to use for the calculation of the sample
XYZs and the CCT (for reference illuminant calculation).
None defaults to the one specified in :cri_type: dict.
- key: 'xyz': str specifying CMF set for calculating xyz
of samples and white
- key: 'cct': str specifying CMF set for calculating cct

cspace

None or dict, optional
Specifies which color space to use.
None defaults to the one specified in :cri_type: dict.
- key: 'type': str specifying color space used to calculate
color differences in.
- key: 'xyzw': None or ndarray with white point of color space
If None: use xyzw of test / reference (after chromatic
adaptation, if specified)
- other keys specify other possible parameters needed for color
space calculation,
see lx.cri._CRI_DEFAULTS['iesrf']['cspace'] for details.

catf

None or dict, optional
Perform explicit CAT before converting to color space coordinates.
- None: don't apply a cat (other than perhaps the one built

into the colorspace)
- dict: with CAT parameters:
- key: 'D': ndarray with degree of adaptation
- key: 'mcat': ndarray with sensor matrix specification
- key: 'xyzw': None or ndarray with white point
None: use xyzw of reference otherwise transform both
test and ref to xyzw

cri_specific_pars

None or dict, optional
Specifies other parameters specific to type of cri
(e.g. maxC for CQS calculations)
- None: default to the one specified in :cri_type: dict.
- dict: user specified parameters.
For its use, see for example:
luxpy.cri._CRI_DEFAULTS['mcri']['cri_specific_pars']

rg_pars

None or dict, optional
Dict containing specifying parameters for slicing the gamut.
Dict structure:
{'nhbins' : None, 'start_hue' : 0,
'normalize_gamut' : False, 'normalized_chroma_ref': 100.0}
- key: 'nhbins': int, number of hue bins to slice gamut
(None use the one specified in :cri_type: dict).
- key: 'start_hue': float (°), hue at which to start slicing
- key: 'normalize_gamut': True or False:
normalize gamut or not before calculating a gamut
area index Rg.
- key: 'normalized_chroma_ref': 100.0 or float, optional
Controls the size (chroma/radius)
of the normalization circle/gamut.

avg

None or fcn handle, optional
Averaging function (handle) for color differences, DEi
(e.g. numpy.mean, .math.rms, .math.geomean)
None use the one specified in :cri_type: dict.

scale

None or dict, optional
Specifies scaling of color differences to obtain CRI.
- None use the one specified in :cri_type: dict.
- dict: user specified dict with scaling parameters.
- key: 'fcn': function handle to type of cri scale,
e.g.
* linear()_scale –> (100 - scale_factor*DEi),
* log_scale –> (cfr. Ohno's CQS),
* psy_scale (Smet et al.'s cri2012,See: LRT 2013)
- key: 'cfactor': factors used in scaling function,

If None:

Scaling factor value(s) will be optimized to minimize the rms between the Rf's of the requested metric and the target metric specified in:

- key: 'opt_cri_type': str
  * str: one of the preset _CRI_DEFAULTS
  * dict: user speciied
  (dict must contain all keys as normal)
  Note that if key not in :scale: dict, then 'opt_cri_type' is added with default setting = 'ciera'.
- key: 'opt_spd_set': ndarray with set of light source spds used to optimize cfactor.
  Note that if key not in :scale: dict, then default = 'F1-F12'.

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 La = (Yb/100)*(E/pi) to then calculate D
following the 'cat02' model).
If None: the degree is determined by :D:
If (:E: is not None) & (:Yb: is None): :E: is assumed to contain
the adapting field luminance La (cd/m^2).

Yb

20.0, optional
Luminance factor of background. (used when calculating La from E)
If None, E contains La (cd/m^2).

out

'Rm' or str, optional
Specifies requested output (e.g. 'Rm,Rmi,cct,duv')

wl

None, optional
Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
None: default to no interpolation

Returns:

returns

float or ndarray with MCRI Rm for :out: 'Rm'
Other output is also possible by changing the :out: str value.

References: 1. K.A.G. Smet, W.R. Ryckaert, M.R. Pointer, G. Deconinck, P. Hanselaer,(2012) "A memory colour quality metric for white light sources," Energy Build., vol. 49, no. C, pp. 216–225.

luxpy.color.cri.**spd_to_cqs**(*SPD*, *version='v9.0'*, *out='Qa'*, *wl=None*)

Calculates CQS Qa (Qai) or Qf (Qfi) or Qp (Qpi) for versions v9.0 or v7.5.

Args:

SPD

ndarray with spectral data (can be multiple SPDs,
first axis are the wavelengths)

version

'v9.0' or 'v7.5', optional

out

'Qa' or str, optional
Specifies requested output (e.g. 'Qa,Qai,Qf,cct,duv')

wl

None, optional
Wavelengths (or [start, end, spacing]) to interpolate the SPDs to.
None: default to no interpolation

Returns:

returns

float or ndarray with CQS Qa for :out: 'Qa'
Other output is also possible by changing the :out: str value.

References: 1. W. Davis and Y. Ohno, "Color quality scale," (2010), Opt. Eng., vol. 49, no. 3, pp. 33602–33616.

luxpy.color.cri.**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 fcns)

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([3.7835e+00, 3.3161e+00, 2.8272e+00, 1.9093e+00, 5.2787e+00, 4.3081e+00, 3.7762e-01, 6.2055e+00, 1.4564e+00, 8.8927e-01]), 'labels': array(['5B', '5BG', '5G', '5GY', '5P', '5PB', '5R', '5RP', '5Y', '5YR'], dtype=object), 'sig': 0.3}, scale_vf_chroma_to_sample_chroma=False*)

Calculates IES TM30 metrics from spectral data.

Args:

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 themselved OR by calculating the dCoverC and dH at resp. 5 and 6 hues. :VF_pcolorshift: specifies these hues and chroma level.

scale_vf_chroma_to_sample_chroma

False, optional
Scale chroma of reference and test vf fields such that average of binned reference chroma equals that of the binned sample chroma before calculating hue bin metrics.

Returns:

data

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 metameric uncertainty index Rt
- 'Rti' : ndarray with specific metameric 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([3.7835e+00, 3.3161e+00, 2.8272e+00, 1.9093e+00, 5.2787e+00, 4.3081e+00, 3.7762e-01, 6.2055e+00, 1.4564e+00, 8.8927e-01]), 'labels': array(['5B', '5BG', '5G', '5GY', '5P', '5PB', '5R', '5RP', '5Y', '5YR'], dtype=object), 'sig': 0.3}, vf_color='k', vf_bin_labels=array(['5B', '5BG', '5G', '5GY', '5P', '5PB', '5R', '5RP', '5Y', '5YR'], dtype=object), vf_plot_bin_colors=True, scale_vf_chroma_to_sample_chroma=False, plot_VF=True, plot_CF=False, plot_SF=False*)

Plot 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 metameric uncertainty.

vf_pcolorshift

_VF_PCOLORSHIFT or user defined dict, optional
The polynomial models of degree 5 and 6 can be fully specified or summarized by the model parameters themselved OR by calculating the dCoverC and dH at resp. 5 and 6 hues. :VF_pcolorshift: specifies these hues and chroma level.

vf_color

'k', optional
For plotting the vector fields.

vf_plot_bin_colors

True, optional
Colorize hue bins of VF graph.

scale_vf_chroma_to_sample_chroma

False, optional
Scale chroma of reference and test vf fields such that average of binned reference chroma equals that of the binned sample chroma before calculating hue bin metrics.

vf_bin_labels

see :bin_labels:
Set VF model hue-bin labels.

plot_CF

False, optional
Plot circle fields.

plot_VF

True, optional
Plot vector fields.

plot_SF

True, optional

Plot sample shifts.

Returns:

returns

(data,
[plt.gcf(),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
- 'Rcshi': ndarray with local chroma shifts indices
- 'Rhshi': ndarray with local hue shifts indices
- 'Rt' : ndarray with general metameric uncertainty index Rt
- 'Rti' : ndarray with specific metameric 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
Ndarray specifying the a-coordinates at which to apply the model.

bxr

np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR), optional
Ndarray specifying the b-coordinates at which to apply the model.

make_grid

True, optional
True: generate a 2d-grid from :axr:, :bxr:.

limit_grid_radius

0, optional
A value of zeros keeps grid as specified by axr,bxr.
A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

color

'k', optional
For plotting the vector field.
If :color: == 0, no plot will be generated.

Returns:

returns

If :color: == 0: ndarray of axt,bxt,axr,bxr
Else: handle to axes used for plotting.

luxpy.color.cri.VFPX.**VF_colorshift_model**(*S*, *cri_type='iesrf'*, *model_type='M6'*, *cspace={'Yw': None, 'conditions': {'D': 1.0, 'Dtype': None, 'La': 100.0, 'Yb': 20.0, 'surround': 'avg'}, 'mcat': 'cat02', 'type': 'jab_cam02ucs', 'xyzw': None, 'yellowbluepurplecorrect': None}*, *sampleset=None*, *pool=False*, *pcolorshift={'Cref': 40, 'href': array([3.1416e-01, 9.4248e-01, 1.5708e+00, 2.1991e+00, 2.8274e+00, 3.4558e+00, 4.0841e+00, 4.7124e+00, 5.3407e+00, 5.9690e+00]), 'sig': 0.3}*, *vfcolor='k'*, *verbosity=0*)

Applies full vector field model calculations to spectral data.

Args:

S

nump.ndarray with spectral data.

cri_type

_VF_CRI_DEFAULT or str or dict, optional
Specifies type of color fidelity model to use.
Controls choice of ref. ill., sample set, averaging, scaling, etc.
See luxpy.cri.spd_to_cri for more info.

modeltype

_VF_MODEL_TYPE or 'M6' or 'M5', optional
Specifies degree 5 or degree 6 polynomial model in ab-coordinates.

cspace

_VF_CSPACE or dict, optional
Specifies color space. See _VF_CSPACE_EXAMPLE for example structure.

sampleset

None or str or ndarray, optional
Sampleset to be used when calculating vector field model.

pool

False, optional
If :S: contains multiple spectra, True pools all jab data before
modeling the vector field, while False models a different field
for each spectrum.

pcolorshift

default dict (see below) or user defined dict, optional
Dict containing the specification input
for apply_poly_model_at_hue_x().
Default dict = {'href': np.arange(np.pi/10,2*np.pi,2*np.pi/10),
'Cref' : _VF_MAXR,
'sig' : _VF_SIG,
'labels' : '#'}
The polynomial models of degree 5 and 6 can be fully specified or
summarized by the model parameters themselved OR by calculating the

dCoverC and dH at resp. 5 and 6 hues.

vfcolor

'k', optional
For plotting the vector fields.

verbosity

0, optional
Report warnings or not.

Returns:

returns

list[dict] (each list element refers to a different test SPD)
with the following keys:
- 'Source': dict with ndarrays of the S, cct and duv of source spd.
- 'metrics': dict with ndarrays for:
  * Rf (color fidelity: base + metameric shift)
  * Rt (metameric uncertainty index)
  * Rfi (specific color fidelity indices)
  * Rti (specific metameric uncertainty indices)
  * cri_type (str with cri_type)
- 'Jab': dict with with ndarrays for Jabt, Jabr, DEi
- 'dC/C_dH_x_sig' :
  np.vstack((dCoverC_x,dCoverC_x_sig,dH_x,dH_x_sig)).T
  See get_poly_model() for more info.
- 'fielddata': dict with dicts containing data on the calculated
  vector-field and circle-fields:
  * 'vectorfield' : {'axt': vfaxt, 'bxt' : vfbxt,
    'axr' : vfaxr, 'bxr' : vfbxr},
  * 'circlefield' : {'axt': cfaxt, 'bxt' : cfbxt,
    'axr' : cfaxr, 'bxr' : cfbxr}},
- 'modeldata' : dict with model info:
  {'pmodel': pmodel,
  'pcolorshift' : pcolorshift,
  'dab_model' : dab_model,
  'dab_res' : dab_res,
  'dab_std' : dab_std,
  'modeltype' : modeltype,
  'fmodel' : poly_model,
  'Jabtm' : Jabtm,
  'Jabrm' : Jabrm,
  'DEim' : DEim},
- 'vshifts' :dict with various vector shifts:
  * 'Jabshiftvector_r_to_t' : ndarray with difference vectors
    between jabt and jabr.
  * 'vshift_ab_s' : vshift_ab_s: ab-shift vectors of samples
  * 'vshift_ab_s_vf' : vshift_ab_s_vf: ab-shift vectors of
    VF model predictions of samples.
  * 'vshift_ab_vf' : vshift_ab_vf: ab-shift vectors of VF
    model predictions of vector field grid.

luxpy.color.cri.VFPX.**initialize_VF_hue_angles**(*hx=None*, *Cxr=40*, *cri_type='iesrf'*, *modeltype='M6'*, *determine_hue_angles=True*)

Initialize the hue angles that will be used to 'summarize' the VF model fitting parameters.

Args:

hx

None or ndarray, optional
None defaults to Munsell H5 hues.

Cxr

_VF_MAXR, optional

cri_type

_VF_CRI_DEFAULT or str or dict, optional,
Cri_type parameters for cri and VF model.

modeltype

_VF_MODEL_TYPE or 'M5' or 'M6', optional
Determines the type of polynomial model.

determine_hue_angles

_DETERMINE_HUE_ANGLES or True or False, optional
True: determines the 10 primary / secondary Munsell hues ('5..').
Note that for 'M6', an additional

Returns:

pcolorshift

{'href': href,
'Cref' : _VF_MAXR,
'sig' : _VF_SIG,
'labels' : list[str]}

luxpy.color.cri.VFPX.**generate_grid**(*jab_ranges=None*, *out='grid'*, *ax=array([-40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40])*, *bx=array([-40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40])*, *jx=None*, *limit_grid_radius=0*)

Generate a grid of color coordinates.

Args:

out

'grid' or 'vectors', optional
- 'grid': outputs a single 2d numpy.nd-vector with the grid coordinates
- 'vector': outputs each dimension seperately.

jab_ranges

None or ndarray, optional
Specifies the pixelization of color space.
(ndarray.shape = (3,3), with first axis: J,a,b, and second
axis: min, max, delta)

ax

default ndarray or user defined ndarray, optional
default = np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR)

bx

default ndarray or user defined ndarray, optional

default = np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR)

jx

None, optional

Note that not-None :jab_ranges: override :ax:, :bx: and :jx input.

limit_grid_radius

0, optional

A value of zeros keeps grid as specified by axr,bxr.

A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

Returns:

returns

single ndarray with ax,bx [,jx]

or

seperate ndarrays for each dimension specified.

luxpy.color.cri.VFPX.**calculate_shiftvectors**(*jabt*, *jabr*, *average=True*, *vtype='ab'*)

Calculate color shift vectors.

Args:

jabt

ndarray with jab coordinates under the test SPD

jabr

ndarray with jab coordinates under the reference SPD

average

True, optional

If True, take mean of difference vectors along axis = 0.

vtype

'ab' or 'jab', optional

Reduce output ndarray to only a,b coordinates of shift vector(s).

Returns:

returns

ndarray of (mean) shift vector(s).

luxpy.color.cri.VFPX.**plot_shift_data**(*data*, *fieldtype='vectorfield'*, *scalef=40*, *color='k'*, *axtype='polar'*, *ax=None*, *hbins=10*, *start_hue=0.0*, *bin_labels='#'*, *plot_center_lines=True*, *plot_axis_labels=False*, *plot_edge_lines=False*, *plot_bin_colors=True*, *force_CVG_layout=True*)

Plots vector or circle fields generated by VFcolorshiftmodel() or PXcolorshiftmodel().

Args:

data

dict generated by VFcolorshiftmodel() or PXcolorshiftmodel()

Must contain 'fielddata'- key, which is a dict with possible keys:

- key: 'vectorfield': ndarray with vector field data
- key: 'circlefield': ndarray with circle field data

color

'k', optional

Color for plotting the vector-fields.

axtype

'polar' or 'cart', optional

Make polar or Cartesian plot.

ax

None or 'new' or 'same', optional
- None or 'new' creates new plot
- 'same': continue plot on same axes.
- axes handle: plot on specified axes.

hbins

16 or ndarray with sorted hue bin centers (°), optional

start_hue

_VF_MAXR, optional
Scale factor for graphic.

plot_axis_labels

False, optional
Turns axis ticks on/off (True/False).

bin_labels

None or list[str] or '#', optional
Plots labels at the bin center hues.
- None: don't plot.
- list[str]: list with str for each bin.
  (len(:bin_labels:) = :nhbins:)
- '#': plots number.

plot_edge_lines

True or False, optional
Plot grey bin edge lines with '–'.

plot_center_lines

False or True, optional
Plot colored lines at 'center' of hue bin.

plot_bin_colors

True, optional
Colorize hue-bins.

force_CVG_layout

False or True, optional
True: Force plot of basis of CVG.

Returns:

returns

figCVG, hax, cmap

:figCVG: handle to CVG figure
:hax: handle to CVG axes
:cmap: list with rgb colors for hue bins
(for use in other plotting fcns)

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

Plot one or more concentric circles around (0,0).

Args:

radii

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

angles

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

color

'k', optional
Color for plotting.

linestyle

'–', optional
Linestyle of circles.

out

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

Returns:

x,y

ndarrays with circle coordinates (only returned if out is 'x,y')

luxpy.color.cri.VFPX.**get_pixel_coordinates**(*jab*, *jab_ranges=None*, *jab_deltas=None*, *limit_grid_radius=0*)

Get pixel coordinates corresponding to array of jab color coordinates.

Args:

jab

ndarray of color coordinates

jab_ranges

None or ndarray, optional
Specifies the pixelization of color space.
(ndarray.shape = (3,3), with first axis: J,a,b, and second axis: min, max, delta)

jab_deltas

float or ndarray, optional
Specifies the sampling range.
A float uses jab_deltas as the maximum Euclidean distance to select
samples around each pixel center. A ndarray of 3 deltas, uses
a city block sampling around each pixel center.

limit_grid_radius

0, optional
A value of zeros keeps grid as specified by axr,bxr.
A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

Returns:

returns

gridp, idxp, jabp, samplenrs, samplesIDs
- :gridp: ndarray with coordinates of all pixel centers.
- :idxp: list[int] with pixel index for each non-empty pixel
- :jabp: ndarray with center color coordinates of non-empty pixels
- :samplenrs: list[list[int]] with sample numbers belong to each

non-empty pixel
- :sampleIDs: summarizing list,
with column order: 'idxp, jabp, samplenrs'

luxpy.color.cri.VFPX.**PX_colorshift_model**(*Jabt*, *Jabr*, *jab_ranges=None*, *jab_deltas=None*, *limit_grid_radius=0*)

Pixelates the color space and calculates the color shifts in each pixel.

Args:

Jabt

ndarray with color coordinates under the (single) test SPD.

Jabr

ndarray with color coordinates under the (single) reference SPD.

jab_ranges

None or ndarray, optional
Specifies the pixelization of color space.
(ndarray.shape = (3,3), with first axis: J,a,b, and second
axis: min, max, delta)

jab_deltas

float or ndarray, optional
Specifies the sampling range.
A float uses jab_deltas as the maximum Euclidean distance to select
samples around each pixel center. A ndarray of 3 deltas, uses
a city block sampling around each pixel center.

limit_grid_radius

0, optional
A value of zeros keeps grid as specified by axr,bxr.
A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

Returns:

returns

dict with the following keys:
- 'Jab': dict with with ndarrays for:
Jabt, Jabr, DEi, DEi_ab (only ab-coordinates), DEa (mean)
and DEa_ab
- 'vshifts': dict with:
* 'vectorshift': ndarray with vector shifts between average
Jabt and Jabr for each pixel
* 'vectorshift_ab': ndarray with vector shifts averaged
over J for each pixel
* 'vectorshift_ab_J0': ndarray with vector shifts averaged
over J for each pixel of J=0 plane.
* 'vectorshift_len': length of 'vectorshift'
* 'vectorshift_ab_len': length of 'vectorshift_ab'
* 'vectorshift_ab_J0_len': length of 'vectorshift_ab_J0'
* 'vectorshift_len_DEnormed': length of 'vectorshift'
normalized to 'DEa'
* 'vectorshift_ab_len_DEnormed': length of 'vectorshift_ab'
normalized to 'DEa_ab'

* 'vectorshift_ab_J0_len_DEnormed': length of 'vectorshift_ab_J0'
normalized to 'DEa_ab'
- 'pixeldata': dict with pixel info:
* 'grid' ndarray with coordinates of all pixel centers.
* 'idx': list[int] with pixel index for each non-empty pixel
* 'Jab': ndarray with center coordinates of non-empty pixels
* 'samplenrs': list[list[int]] with sample numbers belong to each non-empty pixel
* 'IDs: summarizing list, with column order: 'idxp, jabp, samplenrs'
- 'fielddata' : dict with dicts containing data on the calculated vector-field and circle-fields
* 'vectorfield': dict with ndarrays for the ab-coordinates under the ref. (axr, bxr) and test (axt, bxt) illuminants, centered at the pixel centers corresponding to the ab-coordinates of the reference illuminant.

luxpy.color.cri.VFPX.**calculate_VF_PX_models**(*S*, *cri_type='iesrf'*, *sampleset=None*, *pool=False*, *pcolorshift={'Cref': 40*, *'href': array([3.1416e-01*, *9.4248e-01*, *1.5708e+00*, *2.1991e+00*, *2.8274e+00*, *3.4558e+00*, *4.0841e+00*, *4.7124e+00*, *5.3407e+00*, *5.9690e+00])*, *'labels': '#'*, *'sig': 0.3}*, *vfcolor='k'*, *verbosity=0*)

Calculate Vector Field and Pixel color shift models.

Args:

cri_type

_VF_CRI_DEFAULT or str or dict, optional
Specifies type of color fidelity model to use.
Controls choice of ref. ill., sample set, averaging, scaling, etc.
See luxpy.cri.spd_to_cri for more info.

sampleset

None or str or ndarray, optional
Sampleset to be used when calculating vector field model.

pool

False, optional
If :S: contains multiple spectra, True pools all jab data before modeling the vector field, while False models a different field for each spectrum.

pcolorshift

default dict (see below) or user defined dict, optional
Dict containing the specification input for apply_poly_model_at_hue_x().
Default dict = {'href': np.arange(np.pi/10,2*np.pi,2*np.pi/10),
'Cref' : _VF_MAXR,
'sig' : _VF_SIG,
'labels' : '#'}

The polynomial models of degree 5 and 6 can be fully specified or summarized by the model parameters themselved OR by calculating the dCoverC and dH at resp. 5 and 6 hues.

vfcolor

'k', optional
For plotting the vector fields.

verbosity

0, optional
Report warnings or not.

Returns:

returns

:dataVF:, :dataPX:
Dicts, for more info, see output description of resp.:
luxpy.cri.VF_colorshift_model() and luxpy.cri.PX_colorshift_model()

luxpy.color.cri.VFPX.**subsample_RFL_set**(*rfl*, *rflpath=''*, *samplefcn='rand'*, *S=array([[3.6000e+02, 3.6100e+02, 3.6200e+02, 3.6300e+02, 3.6400e+02, 3.6500e+02, 3.6600e+02, 3.6700e+02, 3.6800e+02, 3.6900e+02, 3.7000e+02, 3.7100e+02, 3.7200e+02, 3.7300e+02, 3.7400e+02, 3.7500e+02, 3.7600e+02, 3.7700e+02, 3.7800e+02, 3.7900e+02, 3.8000e+02, 3.8100e+02, 3.8200e+02, 3.8300e+02, 3.8400e+02, 3.8500e+02, 3.8600e+02, 3.8700e+02, 3.8800e+02, 3.8900e+02, 3.9000e+02, 3.9100e+02, 3.9200e+02, 3.9300e+02, 3.9400e+02, 3.9500e+02, 3.9600e+02, 3.9700e+02, 3.9800e+02, 3.9900e+02, 4.0000e+02, 4.0100e+02, 4.0200e+02, 4.0300e+02, 4.0400e+02, 4.0500e+02, 4.0600e+02, 4.0700e+02, 4.0800e+02, 4.0900e+02, 4.1000e+02, 4.1100e+02, 4.1200e+02, 4.1300e+02, 4.1400e+02, 4.1500e+02, 4.1600e+02, 4.1700e+02, 4.1800e+02, 4.1900e+02, 4.2000e+02, 4.2100e+02, 4.2200e+02, 4.2300e+02, 4.2400e+02, 4.2500e+02, 4.2600e+02, 4.2700e+02, 4.2800e+02, 4.2900e+02, 4.3000e+02, 4.3100e+02, 4.3200e+02, 4.3300e+02, 4.3400e+02, 4.3500e+02, 4.3600e+02, 4.3700e+02, 4.3800e+02, 4.3900e+02, 4.4000e+02, 4.4100e+02, 4.4200e+02, 4.4300e+02, 4.4400e+02, 4.4500e+02, 4.4600e+02, 4.4700e+02, 4.4800e+02, 4.4900e+02, 4.5000e+02, 4.5100e+02, 4.5200e+02, 4.5300e+02, 4.5400e+02, 4.5500e+02, 4.5600e+02, 4.5700e+02, 4.5800e+02, 4.5900e+02, 4.6000e+02, 4.6100e+02, 4.6200e+02, 4.6300e+02, 4.6400e+02, 4.6500e+02, 4.6600e+02, 4.6700e+02, 4.6800e+02, 4.6900e+02, 4.7000e+02, 4.7100e+02, 4.7200e+02, 4.7300e+02, 4.7400e+02, 4.7500e+02, 4.7600e+02, 4.7700e+02, 4.7800e+02, 4.7900e+02, 4.8000e+02, 4.8100e+02, 4.8200e+02, 4.8300e+02, 4.8400e+02, 4.8500e+02, 4.8600e+02, 4.8700e+02, 4.8800e+02, 4.8900e+02, 4.9000e+02, 4.9100e+02, 4.9200e+02, 4.9300e+02, 4.9400e+02, 4.9500e+02, 4.9600e+02, 4.9700e+02, 4.9800e+02, 4.9900e+02, 5.0000e+02, 5.0100e+02, 5.0200e+02, 5.0300e+02, 5.0400e+02, 5.0500e+02, 5.0600e+02, 5.0700e+02, 5.0800e+02, 5.0900e+02, 5.1000e+02, 5.1100e+02, 5.1200e+02, 5.1300e+02, 5.1400e+02, 5.1500e+02, 5.1600e+02, 5.1700e+02, 5.1800e+02, 5.1900e+02, 5.2000e+02, 5.2100e+02, 5.2200e+02, 5.2300e+02, 5.2400e+02, 5.2500e+02, 5.2600e+02, 5.2700e+02, 5.2800e+02, 5.2900e+02, 5.3000e+02, 5.3100e+02, 5.3200e+02, 5.3300e+02, 5.3400e+02, 5.3500e+02, 5.3600e+02,*

Sub-samples a spectral reflectance set by pixelization of color space.
Args:

rfl

ndarray or str
Array with of str referring to a set of spectral reflectance
functions to be subsampled.
If str to file: file must contain data as columns, with first
column the wavelengths.

rflpath

‘’ or str, optional
Path to folder with rfl-set specified in a str :rfl: filename.

samplefcn

‘rand’ or ‘mean’, optional
-‘rand’: selects a random sample from the samples within each pixel
-‘mean’: returns the mean spectral reflectance in each pixel.

S

_CIE_ILLUMINANTS[‘E’], optional
Illuminant used to calculate the color coordinates of the spectral
reflectance samples.

jab_ranges

None or ndarray, optional
Specifies the pixelization of color space.
(ndarray.shape = (3,3), with first axis: J,a,b, and second
axis: min, max, delta)

jab_deltas

float or ndarray, optional
Specifies the sampling range.
A float uses jab_deltas as the maximum Euclidean distance to select
samples around each pixel center. A ndarray of 3 deltas, uses
a city block sampling around each pixel center.

cspace

_VF_CSPACE or dict, optional
Specifies color space. See _VF_CSPACE_EXAMPLE for example structure.

cieobs

_VF_CIEOBS or str, optional
Specifies CMF set used to calculate color coordinates.

ax

default ndarray or user defined ndarray, optional
default = np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR)

bx

default ndarray or user defined ndarray, optional
default = np.arange(-_VF_MAXR,_VF_MAXR+_VF_DELTAR,_VF_DELTAR)

jx

None, optional
Note that not-None :jab_ranges: override :ax:, :bx: and :jx input.

limit_grid_radius

0, optional
A value of zeros keeps grid as specified by axr,bxr.
A value > 0 only keeps (a,b) coordinates within :limit_grid_radius:

Returns:

returns

rflsampled, jabp
ndarrays with resp. the subsampled set of spectral reflectance
functions and the pixel coordinate centers.

luxpy.color.cri.VFPX.**plot_VF_PX_models**(*dataVF=None, dataPX=None, plot_VF=True, plot_PX=True, axtype='polar', ax='new', plot_circle_field=True, plot_sample_shifts=False, plot_samples_shifts_at_pixel_center=False, jabp_sampled=None, plot_VF_colors=['g'], plot_PX_colors=['r'], hbin_cmap=None, bin_labels=None, plot_bin_colors=True, force_CVG_layout=False*)

Plot the VF and PX model color shift vectors.

Args:

dataVF

None or list[dict] with VF_colorshift_model() output, optional
None plots nothing related to VF model.
Each list element refers to a different test SPD.

dataPX

None or list[dict] with PX_colorshift_model() output, optional
None plots nothing related to PX model.
Each list element refers to a different test SPD.

plot_VF

True, optional
Plot VF model (if :dataVF: is not None).

plot_PX

True, optional
Plot PX model (if :dataPX: is not None).

axtype

'polar' or 'cart', optional
Make polar or Cartesian plot.

ax

None or 'new' or 'same', optional
- None or 'new' creates new plot
- 'same': continue plot on same axes.
- axes handle: plot on specified axes.

plot_circle_field

True or False, optional
Plot lines showing how a series of circles of color coordinates is
distorted by the test SPD.
The width (wider means more) and color (red means more) of the

lines specify the intensity of the hue part of the color shift.

plot_sample_shifts

False or True, optional
Plots the shifts of the individual samples of the rfl-set used to calculated the VF model.

plot_samples_shifts_at_pixel_center

False, optional
Offers the possibility of shifting the vector shifts of subsampled sets from the reference illuminant positions to the pixel centers.
Note that the pixel centers must be supplied in :jabp_sampled:.

jabp_sampled

None, ndarray, optional
Corresponding pixel center for each sample in a subsampled set.

plot_VF_colors

['g'] or list[str], optional
Specifies the plot color the color shift vectors of the VF model.
If len(:plot_VF_colors:) == 1: same color for each list element of :dataVF:.

plot_VF_colors

['g'] or list[str], optional
Specifies the plot color the color shift vectors of the VF model.
If len(:plot_VF_colors:) == 1: same color for each list element of :dataVF:.

hbin_cmap

None or colormap, optional
Color map with RGB entries for each of the hue bins specified by the hues in _VF_PCOLORSHIFT.
If None: cmap will be obtained on first run by luxpy.cri.plot_shift_data() and returned for use in other functions

plot_bin_colors

True, optional
Colorize hue-bins.

bin_labels

None or list[str] or '#', optional
Plots labels at the bin center hues.
- None: don't plot.
- list[str]: list with str for each bin.
  (len(:bin_labels:) = :nhbins:)
- '#': plots number.
- '_VF_PCOLORSHIFT': uses the labels in _VF_PCOLORSHIFT['labels']
- 'pcolorshift': uses the labels in dataVF['modeldata']['pcolorshift']['labels']

force_CVG_layout

False or True, optional
True: Force plot of basis of CVG.

Returns:

returns

ax (handle to current axes), cmap (hbin_cmap)

4.5 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)

Photoreceptor	Photopigment (label,)	Spectral efficiency s()	Quantity (-opic irradiance)	Q-symbol (Ee,)	Unit symbol
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, Ee,() (Wm2), for each wavelength, with the action spectrum, s(),
where s() is normalized to one at its peak:

Ee, = Ee,() s() d

where the corresponding units are Wm2 in each case.

The equivalent luminance is calculated as:

E, = Km Ee,() s() d V() d / s() d

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)

Module for calculation of cyanosis index (AS/NZS 1680.2.5:1997)

_COI_OBS Default CMF set for calculations

_COI_CSPACE Default color space (CIELAB)

_COI_RFL_BLOOD ndarray with reflectance spectra of 100% and 50% oxygenated blood

spd_to_COI_ASNZS1680 Calculate the Cyanosis Observartion Index (COI) [ASNZS 1680.2.5-1995]

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

`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

'Eeas, Eas' or str, optional
Determines values to return.

Returns:

returns

(Eeas, Eas) with Eeas 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'*, *extrapolate_rfl=False*)

Calculate the Cyanosis Observation Index (COI) [ASNZS 1680.2.5-1995].

Args:

S

ndarray with light source spectrum (first column are wavelengths).

tf

_COI_CSPACE, optional
Color space in which to calculate the COI.
Default is CIELAB.

cieobs

_COI_CIEOBS, optional
CMF set to use.
Default is '1931_2'.

out

'COI,cct' or str, optional
Determines output.

extrapolate_rfl

False, optional
If False:
limit the wavelength range of the source to that of the standard
reflectance spectra for the 50% and 100% oxygenated blood.

Returns:

COI

ndarray with cyanosis indices for input sources.

cct

ndarray with correlated color temperatures.

Note: Clause 7.2 of the ASNZS 1680.2.5-1995. standard mentions the properties demanded of the light source used in region where visual conditions suitable to the detection of cyanosis should be provided:

1. The correlated color temperature (CCT) of the source should be from 3300 to 5300 K.
2. The cyanosis observation index should not exceed 3.3

luxpy.toolboxes.photbiochem.**spd_to_CS_CLa_lrc**(*El=None*, *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 (E.q. 1) had set the peak wavelength of the melanopsin at 480 nm. Rea et al. later published a corrigendum with updated model parameters for k, a_{b-y} and a_rod. The comparison table between showing values calculated for a number of sources with the old and updated parameters were very close (~1 unit voor CLa).

2. In that corrrection paper they did not mention a change in the factor (1622) that multiplies the (sum of) the integral(s) in Eq. 1. HOWEVER, the excel calculator released in 2017 and the online calculator show that factor to have a value of 1547.9. The change in values due to the new factor is much larger than their the updated mentioned in note 1!

3. For reasons of consistency the calculator uses the latest model parameters, as could be read from the excel calculator. They values adopted are: multiplier 1547.9, k = 0.2616, a_{b-y} = 0.7 and a_rod = 3.3.

4. The parameter values to convert CLa to CS were also taken from the 2017 excel calculator.

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)

_INDVCMF_DATA_PATH path to data files

_INDVCMF_DATA Dict with required data

_INDVCMF_STD_DEV_ALL_PARAM Dict with std. dev. model parameters

_INDVCMF_CATOBSPFCTR Categorical observer parameters.

_INDVCMF_M_10d xyz to 10° lms conversion matrix.

_WL_CRIT critical wavelength above which interpolation of S-cone data fails.

_WL wavelengths of spectral data.

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.

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.

lmsb_to_xyzb() Convert from LMS cone fundamentals to XYZ CMF.

add_to_cmf_dict() Add set of cmfs to _CMF dict.

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

4. Asano's Individual Colorimetric Observer Model

Note

Port of Matlab code from: https://www.rit.edu/cos/colorscience/re_AsanoObserverFunctions.php (Accessed April 20, 2018)

`luxpy.toolboxes.indvcmf.`**`load_database`**(*wl=None*, *dsrc_std=None*, *dsrc_lms_odens=None*, *path=None*)

Load database required for Asano Individual Observer Model.

Args:

wl

None, optional

Wavelength range to interpolate data to.

None defaults to the wavelength range associated with data in :dsrc_lms_odens:

path

None, optional

Path where data files are stored (If None: look in ./data/ folder under toolbox path)

dsrc_std

None, optional

Data source ('matlab' code, or 'germany') for stdev data on physiological factors.

None defaults to string in _DSRC_STD_DEF

dsrc_lms_odens

None, optional

Data source ('asano', 'cietc197') for LMS absorbance and optical density data.

None defaults to string in _DSRC_LMS_ODENS_DEF

Returns:

data

dict with data for:

- 'LMSa': LMS absorbances

- 'rmd': relative macular pigment density

- 'docul': ocular media optical density

- 'USCensus2010population': data (age and numbers) on a 2010 US Census

- 'CatObsPfctr': dict with iteratively derived Categorical Observer physiological stdevs.

- 'M2d': Asano 2° lms to xyz conversion matrix

- 'M10d': Asano 10° lms to xyz conversion matrix

- standard deviations on physiological parameters: 'od_lens', 'od_macula', 'od_L', 'od_M', 'od_S', 'shft_L', 'shft_M', 'shft_S'

luxpy.toolboxes.indvcmf.**init**(*wl=None*, *dsrc_std=None*, *dsrc_lms_odens=None*, *lms_to_xyz_method=None*, *use_sign_figs=True*, *use_my_round=True*, *use_chop=True*, *path=None*, *out=None*, *verbosity=1*)

Initialize: load database required for Asano Individual Observer Model into the default _DATA dict and set some options for rounding, sign. figs and chopping small value to zero; for source data to use for spectral data for LMS absorp. and optical desnities, ...

Args:

wl

None, optional

Wavelength range to interpolate data to.

None defaults to the wavelength range associated with data in :dsrc_lms_odens:

dsrc_std

None, optional

Data source ('matlab' code, or 'germany') for stdev data on physiological factors.

None defaults to string in _DSRC_STD_DEF

dsrc_lms_odens

None, optional

Data source ('asano', 'cietc197') for LMS absorbance and optical density data.

None defaults to string in _DSRC_LMS_ODENS_DEF

lms_to_xyz_method

None, optional

Method to use to determine lms-to-xyz conversion matrix (options: 'asano', 'cietc197')

use_my_round

True, optional

If True: use my_rounding() conform CIE TC1-91 Python code 'ciefunctions'. (slows down code)

by setting _USE_MY_ROUND.

use_sign_figs

True, optional

If True: use sign_figs() conform CIE TC1-91 Python code 'ciefunctions'. (slows down code)

by setting _USE_SIGN_FIGS.

use_chop

True, optional

If True: use chop() conform CIE TC1-91 Python code 'ciefunctions'. (slows down code)

by setting _USE_CHOP.

path

None, optional

Path where data files are stored (If None: look in ./data/ folder under toolbox path)

out

None, optional

If None: only set global variables, do not output _DATA.copy()

verbosity

1, optional

Print new state of global settings.

Returns:

data

if out is not None: return a dict with dict with data for:
- 'LMSa': LMS absorbances
- 'rmd': relative macular pigment density
- 'docul': ocular media optical density
- 'USCensus2010population': data (age and numbers) on a 2010 US Census
- 'CatObsPfctr': dict with iteratively derived Categorical Observer physiological stdevs.
- 'M2d': Asano 2° lms to xyz conversion matrix
- 'M10d': Asano 10° lms to xyz conversion matrix
- standard deviations on physiological parameters: 'od_lens', 'od_macula', 'od_L', 'od_M', 'od_S', 'shft_L', 'shft_M', 'shft_S'

luxpy.toolboxes.indvcmf.**query_state**()

Print current settings for 'global variables'.

luxpy.toolboxes.indvcmf.**cie2006cmfsEx**(*age=32*, *fieldsize=10*, *wl=None*, *var_od_lens=0*, *var_od_macula=0*, *var_od_L=0*, *var_od_M=0*, *var_od_S=0*, *var_shft_L=0*, *var_shft_M=0*, *var_shft_S=0*, *norm_type=None*, *out='lms'*, *base=False*, *strategy_2=True*, *odata0=None*, *lms_to_xyz_method=None*, *allow_negative_values=False*, *normalize_lms_to_xyz_matrix=False*)

Generate Individual Observer CMFs (cone fundamentals) based on CIE2006 cone fundamentals and published literature on observer variability in color matching and in physiological parameters.

Args:

age

32 or float or int, optional

Observer age

fieldsize

10, optional

Field size of stimulus in degrees (between 2° and 10°).

wl

None, optional

Interpolation/extraplation of :LMS: output to specified wavelengths.

None: output original _WL

var_od_lens

0, optional

Std Dev. in peak optical density [%] of lens.

var_od_macula

0, optional

Std Dev. in peak optical density [%] of macula.

var_od_L

0, optional

Std Dev. in peak optical density [%] of L-cone.

var_od_M

0, optional
Std Dev. in peak optical density [%] of M-cone.

var_od_S

0, optional
Std Dev. in peak optical density [%] of S-cone.

var_shft_L

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

var_shft_L

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

var_shft_S

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

norm_type

None, optional
- 'max': normalize LMSq functions to max = 1
- 'area': normalize to area
- 'power': normalize to power

out

'lms' or 'xyz', optional
Determines output.

base

False, boolean, optional
The returned energy-based LMS cone fundamentals given to the
precision of 9 sign. figs. if 'True', and to the precision of
6 sign. figs. if 'False'.

strategy_2

True, bool, optional
Use strategy 2 in github.com/ifarup/ciefunctions issue #121 for
computing the weighting factor. If false, strategy 3 is applied.

odata0

None, optional
Dict with uncorrected ocular media and macula density functions and LMS absorptance functions
None defaults to the ones stored in _DATA

lms_to_xyz_method

None, optional
Method to use to determine lms-to-xyz conversion matrix (options: 'asano', 'cietc197')

allow_negative_values

False, optional
Cone fundamentals or color matching functions should not have negative values.
If False: X[X<0] = 0.

normalize_lms_to_xyz_matrix

False, optional
Normalize that EEW is always at [100,100,100] in XYZ and LMS system.

Returns:

returns

- 'LMS' [or 'XYZ']: ndarray with individual observer equal area-normalized cone fundamentals. Wavelength have been added.

[- 'M': lms to xyz conversion matrix
- 'trans_lens': ndarray with lens transmission (no interpolation)
- 'trans_macula': ndarray with macula transmission (no interpolation)
- 'sens_photopig' : ndarray with photopigment sens. (no interpolation)]

References: 1. Asano Y, Fairchild MD, and Blondé L (2016). Individual Colorimetric Observer Model. PLoS One 11, 1–19.

2. Asano Y, Fairchild MD, Blondé L, and Morvan P (2016). Color matching experiment for highlighting interobserver variability. Color Res. Appl. 41, 530–539.

3. CIE, and CIE (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, allow_negative_values=False*)

Monte-Carlo generation of individual observer cone fundamentals.

Args:

n_obs

1, optional
Number of observer CMFs to generate.

list_Age

list of observer ages or str, optional
Defaults to 32 (cfr. CIE2006 CMFs)
If 'us_census': use US population census of 2010
to generate list_Age.

fieldsize

fieldsize in degrees (between 2° and 10°), optional
Defaults to 10°.

wl

None, optional
Interpolation/extraplation of :LMS: output to specified wavelengths.
None: output original _WL

norm_type

None, optional
- 'max': normalize LMSq functions to max = 1
- 'area': normalize to area
- 'power': normalize to power

out

'lms' or 'xyz', optional
Determines output.

base

False, boolean, optional
The returned energy-based LMS cone fundamentals given to the
precision of 9 sign. figs. if 'True', and to the precision of
6 sign. figs. if 'False'.

strategy_2

True, bool, optional
Use strategy 2 in github.com/ifarup/ciefunctions issue #121 for
computing the weighting factor. If false, strategy 3 is applied.

odata0

None, optional
Dict with uncorrected ocular media and macula density functions and LMS absorptance functions
None defaults to the ones stored in _DATA

lms_to_xyz_method

None, optional
Method to use to determine lms-to-xyz conversion matrix (options: 'asano', 'cietc197')

allow_negative_values

False, optional
Cone fundamentals or color matching functions should not have negative values.

If False: X[X<0] = 0.

Returns:

returns

LMS [,var_age, vAll]

- LMS: ndarray with population LMS functions.
- var_age: ndarray with population observer ages.
- vAll: dict with population physiological factors (see .keys())

References: 1. Asano Y, Fairchild 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, and CIE (2006). Fundamental Chromaticity Diagram with Physiological Axes - Part I (Vienna: CIE).

4. Asano's Individual Colorimetric Observer Model

`luxpy.toolboxes.indvcmf.`**`getCatObs`**(*n_cat=10*, *fieldsize=2*, *wl=None*, *norm_type=None*, *out='lms'*, *base=False*, *strategy_2=True*, *odata0=None*, *lms_to_xyz_method=None*, *allow_negative_values=False*)

Generate cone fundamentals for categorical observers.

Args:

n_cat

10, optional

Number of observer CMFs to generate.

fieldsize

fieldsize in degrees (between 2° and 10°), optional

Defaults to 10°.

out

'LMS' or str, optional

Determines output.

wl

None, optional

Interpolation/extraplation of :LMS: output to specified wavelengths.

None: output original _WL

norm_type

None, optional

- 'max': normalize LMSq functions to max = 1
- 'area': normalize to area
- 'power': normalize to power

out

'lms' or 'xyz', optional

Determines output.

base

False, boolean, optional

The returned energy-based LMS cone fundamentals given to the precision of 9 sign. figs. if 'True', and to the precision of 6 sign. figs. if 'False'.

strategy_2

True, bool, optional

Use strategy 2 in github.com/ifarup/ciefunctions issue #121 for computing the weighting factor. If false, strategy 3 is applied.

odata0

None, optional

Dict with uncorrected ocular media and macula density functions and LMS absorptance functions

None defaults to the ones stored in _DATA

lms_to_xyz_method

None, optional

Method to use to determine lms-to-xyz conversion matrix (options: 'asano', 'cietc197')

allow_negative_values

False, optional

Cone fundamentals or color matching functions should not have negative values.

If False: X[X<0] = 0.

Returns:

returns

LMS [,var_age, vAll]

- LMS: ndarray with population LMS functions.
- var_age: ndarray with population observer ages.
- vAll: dict with population physiological factors (see .keys())

Notes: 1. Categorical observers are observer functions that would represent color-normal populations. They are finite and discrete as opposed to observer functions generated from the individual colorimetric observer model. Thus, they would offer more convenient and practical approaches for the personalized color imaging workflow and color matching analyses. Categorical observers were derived in two steps. At the first step, 10000 observer functions were generated from the individual colorimetric observer model using Monte Carlo simulation. At the second step, the cluster analysis, a modified k-medoids algorithm, was applied to the 10000 observers minimizing the squared Euclidean distance in cone fundamentals space, and categorical observers were derived iteratively. Since the proposed categorical observers are defined by their physiological parameters and ages, their CMFs can be derived for any target field size. 2. Categorical observers were ordered by the importance; the first categorical observer vas the average observer equivalent to CIEPO06 with 38 year-old for a given field size, followed by the second most important categorical observer, the third, and so on.

3. see: https://www.rit.edu/cos/colorscience/re_AsanoObserverFunctions.php

`luxpy.toolboxes.indvcmf.`**`compute_cmfs`**(*fieldsize=10, age=32, wl=None, var_od_lens=0, var_od_macula=0, var_shft_LMS=[0, 0, 0], var_od_LMS=[0, 0, 0], norm_type=None, out='lms', base=False, strategy_2=True, odata0=None, lms_to_xyz_method=None, allow_negative_values=False, normalize_lms_to_xyz_matrix=False*)

Generate Individual Observer CMFs (cone fundamentals) based on CIE2006 cone fundamentals and published literature on observer variability in color matching and in physiological parameters.

Args:

age

32 or float or int, optional

Observer age

fieldsize

10, optional
Field size of stimulus in degrees (between 2° and 10°).

wl

None, optional
Interpolation/extraplation of :LMS: output to specified wavelengths.
None: output original _WL

var_od_lens

0, optional
Variation of optical density of lens.

var_od_macula

0, optional
Variation of optical density of macula.

var_shft_LMS

[0, 0, 0] optional
Variation (shift) of LMS peak absorptance.

var_od_LMS

[0, 0, 0] optional
Variation of LMS optical densities.

norm_type

None, optional
- 'max': normalize LMSq functions to max = 1
- 'area': normalize to area
- 'power': normalize to power

out

'lms' or 'xyz', optional
Determines output.

base

False, boolean, optional
The returned energy-based LMS cone fundamentals given to the
precision of 9 sign. figs. if 'True', and to the precision of
6 sign. figs. if 'False'.

strategy_2

True, bool, optional
Use strategy 2 in github.com/ifarup/ciefunctions issue #121 for
computing the weighting factor. If false, strategy 3 is applied.

odata0

None, optional
Dict with uncorrected ocular media and macula density functions and LMS absorptance functions
None defaults to the ones stored in _DATA

lms_to_xyz_method

None, optional
Method to use to determine lms-to-xyz conversion matrix (options: 'asano', 'cietc197')

allow_negative_values

False, optional
Cone fundamentals or color matching functions should not have negative values.
If False: X[X<0] = 0.

normalize_lms_to_xyz_matrix

False, optional
Normalize that EEW is always at [100,100,100] in XYZ and LMS system.

Returns:

returns

- 'LMS' [or 'XYZ']: ndarray with individual observer equal area-normalized cone fundamentals. Wavelength have been added.

[- 'M': lms to xyz conversion matrix
- 'trans_lens': ndarray with lens transmission (no interpolation)
- 'trans_macula': ndarray with macula transmission (no interpolation)
- 'sens_photopig' : ndarray with photopigment sens. (no interpolation)]

References: 1. Asano Y, Fairchild MD, and Blondé L (2016). Individual Colorimetric Observer Model. PLoS One 11, 1–19.

2. Asano Y, Fairchild MD, Blondé L, and Morvan P (2016). Color matching experiment for highlighting interobserver variability. Color Res. Appl. 41, 530–539.

3. CIE, and CIE (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.**plot_cmfs**(*cmf*, *axh=None*, ***kwargs*)

Plot cmf set.

luxpy.toolboxes.indvcmf.**add_to_cmf_dict**(*bar=None, cieobs='indv', K=683, M=array([[1.0000e+00, 0.0000e+00, 0.0000e+00], [0.0000e+00, 1.0000e+00, 0.0000e+00], [0.0000e+00, 0.0000e+00, 1.0000e+00]])*)

Add set of cmfs to _CMF dict.

Args:

bar

None, optional
Set of CMFs. None: initializes to empty ndarray.

cieobs

'indv' or str, optional
Name of CMF set.

K

683 (lm/W), optional
Conversion factor from radiometric to photometric quantity.

M

np.eye, optional
Matrix for lms to xyz conversion.

luxpy.toolboxes.indvcmf.**plot_cmfs**(*cmf*, *axh=None*, ***kwargs*)
Plot cmf set.

4.5.3 spdbuild/

py

- __init__.py
- spdbuilder.py

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 phophor LED spectra.

color3mixer() Calculate fluxes required to obtain a target chromaticity when (additively) mixing 3 light sources.

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

spd_builder() Build spectrum based on Gaussians, monochromatic and/or phophor LED-type spectra.

get_w_summed_spd() Calculate weighted sum of spds.

fitnessfcn() Fitness function that calculates closeness of solution x to target values for specified objective functions.

spd_constructor_2() Construct spd from spectral model parameters using pairs of intermediate sources.

spd_constructor_3() Construct spd from spectral model parameters using trio's of intermediate sources.

spd_optimizer_2_3() Optimizes the weights (fluxes) of a set of component spectra by combining pairs (2) or trio's (3) of components to intermediate sources until only 3 remain. Color3mixer can then be called to calculate required fluxes to obtain target chromaticity and fluxes are then back-calculated.

get_optim_pars_dict() Setup dict with optimization parameters.

initialize_spd_model_pars() Initialize spd_model_pars (for spd_constructor) based on type of component_data.

initialize_spd_optim_pars() Initialize spd_optim_pars (x0, lb, ub for use with math.minimizebnd) based on type of component_data.

spd_optimizer() Generate a spectrum with specified white point and optimized for certain objective functions from a set of component spectra or component spectrum model parameters.

Module for building and optimizing SPDs (2)

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_wlr Setup the wavelength range for use in prim_constructor.

_extract_prim_optimization_parameters Extact 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)

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 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)
with n = 2*(2**0.5-1)**0.5

Butterworth :
bw = 1 / (1 + ((2*(wl - peakwl)/fwhm)**2))

Ohno's model:
ohno = (g + strength_shoulder*g**5)/(1+strength_shoulder)

mono_led_spd = ohno*((bw_order >= -1) & (bw_order <= 0)).T + bw*(bw_order > 0).T + lz*((bw_order >=-2) & (bw_order < -1)).T

Reference: 1. Ohno Y (2005). Spectral design considerations for white LED color rendering. Opt. Eng. 44, 111302.

luxpy.toolboxes.spdbuild.**phosphor_led_spd**(*peakwl=450, fwhm=20, wl=[360.0, 830.0, 1.0], bw_order=-1, with_wl=True, strength_shoulder=2, strength_ph=0, peakwl_ph1=530, fwhm_ph1=80, strength_ph1=1, peakwl_ph2=560, fwhm_ph2=80, strength_ph2=None, use_piecewise_fcn=False, verbosity=0, out='spd'*)

Generate phosphor LED spectrum with up to 2 phosphors based on Smet (Opt. Expr. 2011).

Model:
1) If strength_ph2 is not None:
phosphor_spd = (strength_ph1*mono_led_spd(peakwl_ph1, . . . , strength_shoulder = 1)
+ strength_ph2)*mono_led_spd(peakwl_ph2, . . . , strength_shoulder = 1))
/ (strength_ph1 + strength_ph2)
else:
phosphor_spd = (strength_ph1*mono_led_spd(peakwl_ph1, . . . , strength_shoulder = 1)
+ (1-strength_ph1)*mono_led_spd(peakwl_ph2, . . . , strength_shoulder = 1))

2) S = (mono_led_spd() + strength_ph*(phosphor_spd/phosphor_spd.max()))/(1 + strength_ph)

3) piecewise_fcn = S for wl < peakwl and 1 for wl >= peakwl

4) phosphor_led_spd = S*piecewise_fcn

Args:

peakw

int or float or list or ndarray, optional
Peak wavelengths of the monochromatic led.

fwhm

int or float or list or ndarray, optional
Full-Width-Half-Maximum of mono_led spectrum.

wl

_WL3, optional

Wavelength range.

bw_order

-1, optional

Order of Butterworth function.

If -1 or 0: spd profile is Ohno's gaussian based

(to obtain pure Gaussian: set strength_shoulder = 0).

If -2: spd profile is Lorentzian,

else (>0): Butterworth.

Note that this only applies to the monochromatic led spds and not the phosphors spds (these are always gaussian based).

with_wl

True, optional

True outputs a ndarray with first row wavelengths.

strength_shoulder

2, optiona l

Determines the strength of the spectrum shoulders of the mono led.

strength_ph

0, optional

Total contribution of phosphors in mixture.

peakwl_ph1

int or float or list or ndarray, optional

Peak wavelength of the first phosphor.

fwhm_ph1

int or float or list or ndarray, optional

Full-Width-Half-Maximum of gaussian used to simulate first phosphor.

strength_ph1

1, optional

Strength of first phosphor in phosphor mixture.

If :strength_ph2: is None: value should be in the [0,1] range.

peakwl_ph2

int or float or list or ndarray, optional

Peak wavelength of the second phosphor.

fwhm_ph2

int or float or list or ndarray, optional

Full-Width-Half-Maximum of gaussian used to simulate second phosphor.

strength_ph2

None, optional

Strength of second phosphor in phosphor mixture.

If None: strength is calculated as (1-:strength_ph1:)

:target: np2d([100,1/3,1/3]), optional

ndarray with Yxy chromaticity of target.

verbosity

0, optional

If > 0: plots spectrum components (mono_led, ph1, ph2, . . .)

out

'spd', optional
Specifies output.

use_piecewise_fcn

False, optional
True: uses piece-wise function as in Smet et al. 2011. Can give non_smooth spectra optimized from components to which it is applied.

Returns:

returns

spd, component_spds
ndarrays with spectra (and component spds used to build the final spectra)

References: 1. Ohno Y (2005). Spectral design considerations for white LED color rendering. Opt. Eng. 44, 111302.

2. Smet K, Ryckaert WR, Pointer MR, Deconinck G, and Hanselaer P (2011). Optimal colour quality of LED clusters based on memory colours. Opt. Express 19, 6903–6912.

luxpy.toolboxes.spdbuild.**spd_builder**(*flux=None, component_spds=None, peakwl=450, fwhm=20, bw_order=-1, pair_strengths=None, wl=[360.0, 830.0, 1.0], with_wl=True, strength_shoulder=2, strength_ph=0, peakwl_ph1=530, fwhm_ph1=80, strength_ph1=1, peakwl_ph2=560, fwhm_ph2=80, strength_ph2=None, target=None, tar_type='Yuv', cspace_bwtf={}, cieobs='1931_2', use_piecewise_fcn=False, verbosity=0, out='spd', **kwargs*)

Build spectrum based on Gaussian, monochromatic and/or phophor type spectra.

Args:

flux

None, optional
Fluxes of each of the component spectra.
None outputs the individual component spectra.

component_spds

None or ndarray, optional
If None: calculate component spds from input args.

peakw

int or float or list or ndarray, optional
Peak wavelengths of the monochromatic led.

fwhm

int or float or list or ndarray, optional (but must be same shape as peakw!)
Full-Width-Half-Maximum of gaussian.

wl

_WL3, optional
Wavelength range.

bw_order

-1, optional
Order of Butterworth function.
If -1 or 0: spd profile is Ohno's gaussian based
(to obtain pure Gaussian: set strength_shoulder = 0).

If -2: spd profile is Lorentzian,
else (>0): Butterworth.
Note that this only applies to the monochromatic led spds and not
the phosphors spds (these are always gaussian based).

pair_strengths

ndarray with pair_strengths of mono_led spds, optional
If None: will be randomly selected, possibly resulting in
unphysical (out-of-gamut) solution.

with_wl

True, optional
True outputs a ndarray with first row wavelengths.

strength_shoulder

2, optiona l
Determines the strength of the spectrum shoulders of the mono led.

strength_ph

0, optional
Total contribution of phosphors in mixture.

peakwl_ph1

int or float or list or ndarray, optional
Peak wavelength of the first phosphor.

fwhm_ph1

int or float or list or ndarray, optional
Full-Width-Half-Maximum of gaussian used to simulate first phosphor.

strength_ph1

1, optional
Strength of first phosphor in phosphor mixture.
If :strength_ph2: is None: value should be in the [0,1] range.

peakwl_ph2

int or float or list or ndarray, optional
Peak wavelength of the second phosphor.

fwhm_ph2

int or float or list or ndarray, optional
Full-Width-Half-Maximum of gaussian used to simulate second phosphor.

strength_ph2

None, optional
Strength of second phosphor in phosphor mixture.
If None: strength is calculated as (1-:strength_ph1:)
:target: np2d([100,1/3,1/3]), optional
ndarray with Yxy chromaticity of target.

verbosity

0, optional
If > 0: plots spectrum components (mono_led, ph1, ph2, . . .)

out

'spd', optional
Specifies output.

use_piecewise_fcn

False, optional
True: uses piece-wise function as in Smet et al. 2011. Can give non_smooth spectra optimized from components to which it is applied.

target

None, optional
ndarray with Yxy chromaticity of target.
If None: don't override phosphor strengths, else calculate strength to obtain :target: using color3mixer().
If not None AND strength_ph is None or 0: components are monochromatic and colormixer is used to optimize fluxes to obtain target chromaticity (N can be > 3 components)

tar_type

'Yxy' or str, optional
Specifies the input type in :target: (e.g. 'Yxy' or 'cct')

cieobs

_CIEOBS, optional
CIE CMF set used to calculate chromaticity values.

cspace_bwtf

{}, optional
Backward (. . . _to_xyz) transform parameters (see colortf()) to go from :tar_type: to 'Yxy')

Returns:

returns

ndarray with spectra.

Note: 1. Target-optimization is only for phophor_leds with three components (blue pump, ph1 and ph2) spanning a sufficiently large gamut.

References: 1. Ohno Y (2005). Spectral design considerations for white LED color rendering. Opt. Eng. 44, 111302.

2. Smet K, Ryckaert WR, Pointer MR, Deconinck G, and Hanselaer P (2011). Optimal colour quality of LED clusters based on memory colours. Opt. Express 19, 6903–6912.

luxpy.toolboxes.spdbuild.**get_w_summed_spd**(*w*, *spds*)

Calculate weighted sum of spds.

Args:

w

ndarray with weigths (e.g. fluxes)

spds

ndarray with component spds.

Returns:

returns

ndarray with weighted sum.

luxpy.toolboxes.spdbuild.**fitnessfcn**(*x, spd_constructor, spd_constructor_pars=None, F_rss=True, decimals=[3], obj_fcn=[None], obj_fcn_pars=[{}], obj_fcn_weights=[1], obj_tar_vals=[0], verbosity=0, out='F'*)

Fitness function that calculates closeness of solution x to target values for specified objective functions.

Args:

x

ndarray with parameter values

spd_constructor

function handle to a function that constructs the spd from parameter values in :x:.

spd_constructor_pars

None, optional,
Parameters required by :spd_constructor:

F_rss

True, optional
Take Root-Sum-of-Squares of 'closeness' values between target and
objective function values.

decimals

3, optional
List of rounding decimals of objective function values.

obj_fcn

[None] or list, optional
List of function handles to objective function.

obj_fcn_weights

[1] or list, optional.
List of weigths for each obj. fcn

obj_fcn_pars

[None] or list, optional
List of parameter dicts for each obj. fcn.

obj_tar_vals

[0] or list, optional
List of target values for each objective function.

verbosity

0, optional
If > 0: print intermediate results.

out

'F', optional
Determines output.

Returns:

F

float or ndarray with fitness value for current solution :x:.

`luxpy.toolboxes.spdbuild.`**`spd_constructor_2`**(*x*, *constructor_pars={}*, ***kwargs*)

Construct spd from model parameters using pairs of intermediate sources.

Pairs (odd,even) of components are selected and combined using
'pair_strength'. This process is continued until only 3 intermediate
(combined) sources remain. Color3mixer is then used to calculate the
fluxes for the remaining 3 sources, after which the fluxes of all

components are back-calculated.

Args:

x

vector of optimization parameters.

constructor_pars

dict with model parameters.
Key 'list' determines which parameters are in :x: and key 'len'
(Specifies the number of variables representing each parameter).

Returns:

returns

spd, M, spds
ndarrays with spectrum corresponding to x, M the fluxes of
the spectral components of spd and spds the spectral components
themselves.

luxpy.toolboxes.spdbuild.**color3mixer**(*Yxyt*, *Yxy1*, *Yxy2*, *Yxy3*)

Calculate fluxes required to obtain a target chromaticity when (additively) mixing 3 light sources.

Args:

Yxyt

ndarray with target Yxy chromaticities.

Yxy1

ndarray with Yxy chromaticities of light sources 1.

Yxy2

ndarray with Yxy chromaticities of light sources 2.

Yxy3

ndarray with Yxy chromaticities of light sources 3.

Returns:

M

ndarray with fluxes.

Note: Yxyt, Yxy1, ... can contain multiple rows, referring to single mixture.

luxpy.toolboxes.spdbuild.**colormixer**(*Yxyt=None*, *Yxyi=None*, *n=4*, *pair_strengths=None*, *source_order=None*)

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

Args:

Yxyt

ndarray with target Yxy chromaticities.
Defaults to equi-energy white.

Yxyi

ndarray with Yxy chromaticities of light sources i = 1 to n.

n

4 or int, optional
Number of source components to randomly generate when Yxyi is None.

pair_strengths

ndarray with light source pair strengths.

source_order

ndarray with order of source components.
If None: use np.arange(n)

Returns:

M

ndarray with fluxes.

Note:

Algorithm

1. Loop over all source components and create intermediate sources
   from all (even,odd)-pairs using the relative strengths
   of the pair (specified in pair_strengths).
2. Collect any remaining sources.
3. Combine with new intermediate source components
4. Repeat 1-3 until there are only 3 source components left.
5. Use color3mixer to calculate the required fluxes of the 3 final
   intermediate components to obtain the target chromaticity.
6. Backward calculate the fluxes of all original source components
   from the 3 final intermediate fluxes.

`luxpy.toolboxes.spdbuild.`**`spd_constructor_3`**(*x*, *constructor_pars={}*, ***kwargs*)

Construct spd from model parameters using trio's of intermediate sources.

The triangle/trio method creates for all possible combinations of 3 primary
component spectra a spectrum that results in the target chromaticity
using color3mixer() and then optimizes the weights of each of the latter
spectra such that adding them (additive mixing) results in obj_vals as
close as possible to the target values.

Args:

x

vector of optimization parameters.

constructor_pars

dict with model parameters.
Key 'list' determines which parameters are in :x: and key 'len'
(specifies the number of variables representing each parameter).

Returns:

returns

spd, M, spds
ndarrays with spectrum corresponding to x, M the fluxes of
the spectral components of spd and spds the spectral components
themselves.

luxpy.toolboxes.spdbuild.**spd_optimizer_2_3**(*optimizer_type='2mixer', spd_constructor=None, spd_model_pars=None, component_data=4, N_components=None, wl=[360.0, 830.0, 1.0], allow_nongaussianbased_mono_spds=False, Yxy_target=array([[1.0000e+02, 3.3333e-01, 3.3333e-01]]), cieobs='1931_2', obj_fcn=[None], obj_fcn_pars=[{}], obj_fcn_weights=[1], obj_tar_vals=[0], decimals=[5], minimize_method='nelder-mead', minimize_opts=None, F_rss=True, verbosity=0, **kwargs*)

Optimizes the weights (fluxes) of a set of component spectra by combining pairs (2) or trio's (3) of components to intermediate sources until only 3 remain. Color3mixer can then be called to calculate required fluxes to obtain target chromaticity and fluxes are then back-calculated.

Args:

optimizer_type

'2mixer' or '3mixer' or 'user', optional
Specifies whether to optimize spectral model parameters by combining pairs or trio's of comonponents.

spd_constructor

None, optional
Function handle to user defined spd_constructor function.
Input: fcn(x, constructor_pars = {}, kwargs)
Output: spd,M,spds
nd array with:
- spd: spectrum resulting from x
- M: fluxes of all component spds
- spds: component spds (in [N+1,wl] format)
(See e.g. spd_constructor_2 or spd_constructor_3)

spd_model_pars

dict with model parameters required by spd_constructor
and with optimization parameters required by minimize (x0, lb, ub). .
Only used when :optimizer_type: == 'user'.

component_data

4, optional
Component spectra data:
If int: specifies number of components used in optimization
(peakwl, fwhm and pair_strengths will be optimized).
If dict: generate components based on parameters (peakwl, fwhm, pair_strengths, etc.) in dict.
(keys with None values will be optimized)
If ndarray: optimize pair_strengths of component spectra.

N_components

None, optional
Specifies number of components used in optimization. (only used

when :component_data: is dict and user wants to override dict.
Note that shape of parameters arrays must match N_components).

allow_nongaussianbased_mono_spds

False, optional
False: use pure Gaussian based monochrom. spds.

wl

_WL3, optional
Wavelengths used in optimization when :component_data: is not
ndarray with spectral data.

Yxy_target

np2d([100,1/3,1/3]), optional
ndarray with Yxy chromaticity of target.

cieobs

_CIEOBS, optional
CIE CMF set used to calculate chromaticity values if not provided
in :Yxyi:.

F_rss

True, optional
Take Root-Sum-of-Squares of 'closeness' values between target and
objective function values.

decimals

5, optional
Rounding decimals of objective function values.

obj_fcn

[None] or list, optional
Function handles to objective function.

obj_fcn_weights

[1] or list, optional.
Weigths for each obj. fcn

obj_fcn_pars

[None] or list, optional
Parameter dicts for each obj. fcn.

obj_tar_vals

[0] or list, optional
Target values for each objective function.

minimize_method

'nelder-mead', optional
Optimization method used by minimize function.

minimize_opts

None, optional
Dict with minimization options.
None defaults to: {'xtol': 1e-5, 'disp': True, 'maxiter': 1000*Nc,
'maxfev' : 1000*Nc,'fatol': 0.01}

verbosity

0, optional

If > 0: print intermediate results.

Returns:

returns

M, spd_opt, obj_vals
- 'M': ndarray with fluxes for each component spectrum.
- 'spd_opt': optimized spectrum.
- 'obj_vals': values of the obj. fcns for the optimized spectrum.

luxpy.toolboxes.spdbuild.**get_optim_pars_dict**(*target=array([[1.0000e+02, 3.3333e-01, 3.3333e-01]]), tar_type='Yxy', cieobs='1931_2', optimizer_type='2mixer', spd_constructor=None, spd_model_pars=None, cspace='Yuv', cspace_bwtf={}, cspace_fwtf={}, component_spds=None, N_components=None, obj_fcn=[None], obj_fcn_pars=[{}], obj_fcn_weights=[1], obj_tar_vals=[0], decimals=[5], minimize_method='nelder-mead', minimize_opts=None, F_rss=True, peakwl=[450, 530, 610], fwhm=[20, 20, 20], allow_nongaussianbased_mono_spds=False, bw_order=[-1], wl=[360.0, 830.0, 1.0], with_wl=True, strength_shoulder=2, strength_ph=[0], use_piecewise_fcn=False, peakwl_ph1=[530], fwhm_ph1=[80], strength_ph1=[1], peakwl_ph2=[560], fwhm_ph2=[80], strength_ph2=None, verbosity=0, pair_strengths=None, triangle_strengths=None, peakwl_min=[400], peakwl_max=[700], fwhm_min=[5], fwhm_max=[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, allow_nongaussianbased_mono_spds=False, optimizer_type='2mixer', wl=[360.0, 830.0, 1.0]*)

Initialize spd_model_pars dict (for spd_constructor) based on type of component_data.

Args:

component_data

None, optional

Component spectra data:

If int: specifies number of components used in optimization
(peakwl, fwhm and pair_strengths will be optimized).

If dict: generate components based on parameters (peakwl, fwhm,

pair_strengths, etc.) in dict.
(keys with None values will be optimized)
If ndarray: optimize pair_strengths of component spectra.

N_components

None, optional
Specifies number of components used in optimization. (only used
when :component_data: is dict and user wants to override dict.
Note that shape of parameters arrays must match N_components).

allow_nongaussianbased_mono_spds

False, optional
- False: use Gaussian based monochrom. spds.
- True: also allow butterworth and lorentzian type monochrom. spds while optimizing.

optimizer_type

'2mixer', optional
Type of spectral optimization routine.
(other options: '3mixer', 'search')

wl

_WL3, optional
Wavelengths used in optimization when :component_data: is not an
ndarray with spectral data.

Returns:

spd_model_pars

dict with spectrum-model parameters

luxpy.toolboxes.spdbuild.**initialize_spd_optim_pars**(*component_data, N_components=None, allow_nongaussianbased_mono_spds=False, optimizer_type='2mixer', wl=[360.0, 830.0, 1.0], spd_model_pars=None*)

Initialize spd_optim_pars dict based on type of component_data.

Args:

component_data

None, optional
Component spectra data:
If int: specifies number of components used in optimization
(peakwl, fwhm and pair_strengths will be optimized).
If dict: generate components based on parameters (peakwl, fwhm,
pair_strengths, etc.) in dict.
(keys with None values will be optimized)
If ndarray: optimize pair_strengths of component spectra.

N_components

None, optional
Specifies number of components used in optimization. (only used
when :component_data: is dict and user wants to override dict.
Note that shape of parameters arrays must match N_components).

allow_nongaussianbased_mono_spds

False, optional
False: use Gaussian based monochrom. spds.

optimizer_type

'2mixer', optional
Type of spectral optimization routine.
(other options: '3mixer', 'search')

wl

_WL3, optional
Wavelengths used in optimization when :component_data: is not an
ndarray with spectral data.

spd_model_pars

None, optional
If None, initialize based on type of component_data.
else: initialize on pre-defined spd_model_pars dict.

Returns:

spd_optim_pars

dict with optimization parameters (x0, ub, lb)

luxpy.toolboxes.spdbuild.**get_primary_fluxratios**(*res*, *primaries*, *Ytarget=1*, *ptype='pu'*, *cieobs='1931_2'*, *out='M*, *Sopt'*)

Get flux ratios of primaries.

Args:

res

dict or ndarray with optimized fluxes for component spds normalized to max = 1.
(output of spd_optimizer)

primaries

ndarray with primary spectra.

Ytarget

1, optional
M will be scaled to result in a photo-/radio-metric power of Ytarget

ptype

'pu' or 'ru', optional
Type of power:
-'pu': photometric units
-'ru': radiometric units

cieobs

_CIEOBS, optional
CMF set/Vlambda to use in calculation of power.

Returns:

M

ndarray with flux ratios.

Sopt

ndarray with optimized scaled spectrum.

luxpy.toolboxes.spdbuild.**spd_optimizer**(*target=array([[1.0000e+02, 3.3333e-01, 3.3333e-01]]), tar_type='Yxy', cieobs='1931_2', optimizer_type='2mixer', spd_constructor=None, spd_model_pars=None, cspace='Yuv', cspace_bwtf={}, cspace_fwtf={}, component_spds=None, N_components=None, obj_fcn=[None], obj_fcn_pars=[{}], obj_fcn_weights=[1], obj_tar_vals=[0], decimals=[5], minimize_method='nelder-mead', minimize_opts=None, F_rss=True, peakwl=[450, 530, 610], fwhm=[20, 20, 20], allow_nongaussianbased_mono_spds=False, bw_order=[-1], wl=[360.0, 830.0, 1.0], with_wl=True, strength_shoulder=2, strength_ph=[0], use_piecewise_fcn=False, peakwl_ph1=[530], fwhm_ph1=[80], strength_ph1=[1], peakwl_ph2=[560], fwhm_ph2=[80], strength_ph2=None, verbosity=0, pair_strengths=None, 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.
Weigths for each obj. fcn

obj_fcn_pars

[None] or list, optional
Parameter dicts for each obj. fcn.

obj_tar_vals

[0] or list, optional
Target values for each objective function.

minimize_method

'nelder-mead', optional
Optimization method used by minimize function.

minimize_opts

None, optional
Dict with minimization options.
None defaults to: {'xtol': 1e-5, 'disp': True, 'maxiter': 1000*Nc,
'maxfev' : 1000*Nc,'fatol': 0.01}

verbosity

0, optional
If > 0: print intermediate results.

out

'spds,M', optional
Determines output of function.

Note: peakwl:, :fwhm:, ... : see ?spd_builder for more info.

Returns:

returns

spds, M
- 'spds': optimized spectrum.
- 'M': ndarray with fluxes for each component spectrum.

Notes:

Optimization algorithms

1. '2mixer': Pairs (odd,even) of components are selected and combined using 'pair_strength'. This process is continued until only 3 (combined) intermediate sources remain. Color3mixer is then used to calculate the fluxes for the remaining 3 sources, after which the fluxes of all components are back-calculated.

2. '3mixer': The triangle/trio method creates for all possible combinations of 3 primary component spectra a spectrum that results in the target chromaticity using color3mixer() and then optimizes the weights of each of the latter spectra such that adding them (additive mixing) results in obj_vals as close as possible to the target values.

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.
Weigths for each obj. fcn

obj_fcn_pars

[None] or list, optional
Parameter dicts for each obj. fcn.

obj_tar_vals

[0] or list, optional
Target values for each objective function.

minimize_method

'nelder-mead', optional
Optimization method used by minimize function.
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 Optimizatizer
  (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*)

Extact 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:

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

then it should contain an optimizer funtion 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 prims.

Args:

nprim

None, optional
If not None: generate nprim random prims (based fixed pars and bounds in pdefs)
else: values for all pars should be defined in pdefs!
(nprims is determined by number of elements in pdefs[ptypes[0]])

class luxpy.toolboxes.spdbuild.**Minimizer**(*method='nelder-mead'*, *opts={}*, *x0=None*, *pareto=False*, *display=True*)

_set_defopts_and_pareto(*pareto=None*, *x0=None*, *display=None*)
Set default options if not provided, as well as pareto (False: output Root-Sum-Squares of Fi in _fitnessfcn).

apply(*fitness_fcn*, *npars*, *fitness_args_dict*, *bounds*, *verbosity=1*)
Run minimizer on fitness function with specified fitness_args_dict input arguments and bounds.

class luxpy.toolboxes.spdbuild.**ObjFcns**(*f=None, fp=[{}], fw=[1], ft=[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', triangle_strengths_bnds=None, prim_constructor=<luxpy.toolboxes.spdbuild.spdoptimizer20 object>, prims=None, obj_fcn=<luxpy.toolboxes.spdbuild.spdoptimizer2020.ObjFcn object>, minimizer=<luxpy.toolboxes.spdbuild.spdoptimizer2020.Minimize object>, verbosity=1*)

_update_nprim_prims(*nprim=None*, *prims=None*)
Update prims (and nprim).

_update_target(*target=None*, *tar_type=None*, *cspace_bwtf=None*)
Update target chromaticity.

_update_prim_pars_bnds(*nprim=None*, ***kwargs*)
Get and set fixed and free parameters, as well as bnds on latter for an nprim primary mixture.

_update_triangle_strengths_bnds(*nprim=None*, *triangle_strengths_bnds=None*)
Update bounds of triangle_strengths for for an nprim primary mixture.

_update_bnds(*nprim=None*, *triangle_strengths_bnds=None*, ***prim_kwargs*)
Update all bounds (triangle_strengths and those of free parameters of primary constructor) for an nprim primary mixture..

update(*nprim=None*, *prims=None*, *cieobs=None*, *target=None*, *tar_type=None*, *cspace_bwtf=None*, *triangle_strengths_bnds=None*, ***prim_kwargs*)
Updates all that is needed when one of the input arguments is changed.

_spd_constructor_tri(*x*)
Construct a mixture spectrum composed of n primaries using the 3mixer algorithm.
Args:
x

optimization parameters, first n!/(n-3)!*3! are the strengths of
the triangles in the '3mixer' algorithm.

Returns:

spd, prims, M

- spd: spectrum resulting from x
- spds: primary spds
- M: fluxes of all primaries

Notes: 1. '3mixer' - optimization algorithm: The triangle/trio method creates for all possible combinations of 3 primary component spectra a spectrum that results in the target chromaticity using color3mixer() and then optimizes the weights of each of the latter spectra such that adding them (additive mixing) results in obj_vals as close as possible to the target values.

_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_rfl() approximate spectral reflectance of xyz based on k nearest neighbour interpolation of samples from a standard reflectance set.

render_image() Render image under specified light source spd.

luxpy.toolboxes.hypspcim.**render_image**(*img=None*, *spd=None*, *rfl=None*, *out='img_hyp'*, *refspd=None*, *D=None*, *cieobs='1931_2'*, *cspace='xyz'*, *cspace_tf={}*, *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 tristimlus values of white

luxpy.toolboxes.dispcal.**calibration_performance**(*rgb*, *xyztarget*, *M*, *N*, *tr*, *xyz_black*, *xyz_white*, *tr_type='lut'*, *cspace='lab'*, *avg=<function <lambda>>*, *rgb_is_xyz=False*, *is_verification_data=False*, *nbit=8*, *verbosity=1*, *sep=', '*, *header=None*)

Check calibration performance. Calculate DE for each stimulus.

Args:

rgb

ndarray [Nx3] or string with filename of RGB values
(or xyz values if argument rgb_to_xyz == True!)

xyztarget

ndarray [Nx3] or string with filename of target XYZ values corresponding
to the RGB settings (or the measured XYZ values, if argument rgb_to_xyz == True).

M

linear rgb to xyz conversion matrix

N

xyz to linear rgb conversion matrix

tr

Tone Response function parameters or lut

xyz_black

ndarray with XYZ tristimulus values of black

xyz_white

ndarray with tristimlus values of white

tr_type

'lut', optional
options:
- 'lut': Derive/specify Tone-Response as a look-up-table
- 'gog': Derive/specify Tone-Response as a gain-offset-gamma function

cspace

color space or chromaticity diagram to calculate color differences in.

avg

lambda x: ((x**2).mean()**0.5), optional
Function used to average the color differences of the individual RGB settings
in the optimization of the xyz_to_rgb and rgb_to_xyz conversion matrices.

rgb_is_xyz

False, optional
If True: the data in argument rgb are actually measured XYZ tristimulus values
and are directly compared to the target xyz.

is_verification_data

False, optional

If False: the data is assumed to be corresponding to RGB value settings used in the calibration (i.e. containing whites, blacks, grays, pure and binary mixtures)

If True: no assumptions on content of rgb, so use this settings when checking the performance for a set of measured and target xyz data different than the ones used in the actual calibration measurements.

nbit

8, optional

RGB values in nbit format (e.g. 8, 16, . . .)

verbosity

1, optional

> 0: print and plot optimization results

sep

‘,’, optional

separator in files with rgbcal and xyzcal data

header

None, optional

header specifier for files with rgbcal and xyzcal data

(see pandas.read_csv)

Returns:

M

linear rgb to xyz conversion matrix

N

xyz to linear rgb conversion matrix

tr

Tone Response function parameters or lut

xyz_black

ndarray with XYZ tristimulus values of black

xyz_white

ndarray with tristimlus values of white

`luxpy.toolboxes.dispcal.`**`rgb_to_xyz`**(*rgb*, *M*, *tr*, *xyz_black*, *tr_type='lut'*)

Convert input rgb to xyz.

Args:

rgb

ndarray [Nx3] with RGB values

M

linear rgb to xyz conversion matrix

tr

Tone Response function parameters or lut

xyz_black

ndarray with XYZ tristimulus values of black

tr_type

'lut', optional
Type of Tone Response in tr input argument
options:
- 'lut': Tone-Response as a look-up-table
- 'gog': Tone-Response as a gain-offset-gamma function

Returns:

xyz

ndarray [Nx3] of XYZ tristimulus values

luxpy.toolboxes.dispcal.**xyz_to_rgb**(*xyz*, *N*, *tr*, *xyz_black*, *tr_type='lut'*)

Convert xyz to input rgb.

Args:

xyz

ndarray [Nx3] with XYZ tristimulus values

N

xyz to linear rgb conversion matrix

tr

Tone Response function parameters or lut

xyz_black

ndarray with XYZ tristimulus values of black

tr_type

'lut', optional
Type of Tone Response in tr input argument
options:
- 'lut': Tone-Response as a look-up-table
- 'gog': Tone-Response as a gain-offset-gamma function

Returns:

rgb

ndarray [Nx3] of display RGB values

class luxpy.toolboxes.dispcal.**DisplayCalibration**(*rgbcal*, *xyzcal=None*, *L_type='lms'*, *cieobs='1931_2'*, *tr_type='lut'*, *nbit=8*, *cspace='lab'*, *avg=<function DisplayCalibration.<lambda>>*, *verbosity=1*, *sep=', '*, *header=None*)

Class for display_calibration.

Args:

rgbcal

ndarray [Nx3] or string with filename of RGB values
rgcal must contain at least the following type of settings:
- pure R,G,B: e.g. for pure R: (R != 0) & (G==0) & (B == 0)
- white(s): R = G = B = 2**nbit-1
- gray(s): R = G = B
- black(s): R = G = B = 0
- binary colors: cyan (G = B, R = 0), yellow (G = R, B = 0), magenta (R = B, G = 0)

xyzcal

None, optional
ndarray [Nx3] or string with filename of measured XYZ values for

the RGB settings in rgbcal.
if None: rgbcal is [Nx6] ndarray containing rgb (columns 0-2) and xyz data (columns 3-5)

L_type

'lms', optional
Type of response to use in the derivation of the Tone-Response curves.
options:
- 'lms': use cone fundamental responses: L vs R, M vs G and S vs B
  (reduces noise and generally leads to more accurate characterization)
- 'Y': use the luminance signal: Y vs R, Y vs G, Y vs B

tr_type

'lut', optional
options:
- 'lut': Derive/specify Tone-Response as a look-up-table
- 'gog': Derive/specify Tone-Response as a gain-offset-gamma function

cieobs

'1931_2', optional
CIE CMF set used to determine the XYZ tristimulus values
(needed when L_type == 'lms': determines the conversion matrix to convert xyz to lms values)

nbit

8, optional
RGB values in nbit format (e.g. 8, 16, . . .)

cspace

color space or chromaticity diagram to calculate color differences in when optimizing the xyz_to_rgb and rgb_to_xyz conversion matrices.

avg

lambda x: ((x**2).mean()**0.5), optional
Function used to average the color differences of the individual RGB settings in the optimization of the xyz_to_rgb and rgb_to_xyz conversion matrices.

verbosity

1, optional
> 0: print and plot optimization results

sep

',', optional
separator in files with rgbcal and xyzcal data

header

None, optional
header specifier for files with rgbcal and xyzcal data
(see pandas.read_csv)

Return:

calobject

attributes are:
- M: linear rgb to xyz conversion matrix
- N: xyz to linear rgb conversion matrix

- TR: Tone Response function parameters or lut
- xyz_black: ndarray with XYZ tristimulus values of black
- xyz_white: ndarray with tristimlus values of white

as well as:

- rgbcal, xyzcal, cieobs, avg, tr_type, nbit, cspace, verbosity
- performance: dictionary with various color differences set to np.nan
- (run calobject.performance() to fill it with actual values)

check_performance(*rgb=None*, *xyz=None*, *verbosity=None*, *sep=','*, *header=None*, *rgb_is_xyz=False*, *is_verification_data=True*)

Check calibration performance (if rgbcal is None: use calibration data).

Args:

rgb

None, optional
ndarray [Nx3] or string with filename of RGB values
(or xyz values if argument rgb_to_xyz == True!)
If None: use self.rgbcal

xyz

None, optional
ndarray [Nx3] or string with filename of target XYZ values corresponding to the RGB settings (or the measured XYZ values, if argument rgb_to_xyz == True).
If None: use self.xyzcal

verbosity

None, optional
if None: use self.verbosity
if > 0: print and plot optimization results

sep

',', optional
separator in files with rgb and xyz data

header

None, optional
header specifier for files with rgb and xyz data
(see pandas.read_csv)

rgb_is_xyz

False, optional
If True: the data in argument rgb are actually measured XYZ tristimulus values and are directly compared to the target xyz.

is_verification_data

False, optional
If False: the data is assumed to be corresponding to RGB value settings used in the calibration (i.e. containing whites, blacks, grays, pure and binary mixtures)
Performance results are stored in self.performance.
If True: no assumptions on content of rgb, so use this settings when checking the performance for a set of measured and target xyz data different than the ones used in the actual calibration measurements.

Return:

performance

dictionary with various color differences.

to_xyz(*rgb*)
Convert display rgb to xyz.

to_rgb(*xyz*)
Convert xyz to display rgb.

4.5.6 rgb2spec/

py

- __init__.py
- smits_mitsuba.py

namespace luxpy.rgb2spec

Module for RGB to spectrum conversions

_BASESPEC_SMITS Default dict with base spectra for white, cyan, magenta, yellow, blue, green and red for each intent ('rfl' or 'spd')

rgb_to_spec_smits() Convert an array of 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.
If None: use _BASESPEC_SMITS.

Returns:

spec

ndarray with spectrum or spectra (one for each rgb value, first row are the wavelengths)

luxpy.toolboxes.rgb2spec.**rgb_to_spec_smits**(*rgb, intent='rfl', 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.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

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, **??**