python-colormath Documentation

Release 2.0.2

Greg Taylor

August 05, 2014
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Assorted Info</td>
<td>3</td>
</tr>
<tr>
<td>2 Topics</td>
<td></td>
</tr>
<tr>
<td>2.1 Installation</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Color Objects</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Observers and Illuminants</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Color Conversions</td>
<td>7</td>
</tr>
<tr>
<td>2.5 Delta E Equations</td>
<td>8</td>
</tr>
<tr>
<td>2.6 ANSI and ISO Density</td>
<td>9</td>
</tr>
<tr>
<td>2.7 Color Appearance Models</td>
<td>10</td>
</tr>
<tr>
<td>2.8 Release Notes</td>
<td>11</td>
</tr>
<tr>
<td>3 Useful color math resources</td>
<td>15</td>
</tr>
</tbody>
</table>
python-colormath is a simple Python module that spares the user from directly dealing with color math. Some features include:

- Support for a wide range of color spaces. A good chunk of the CIE spaces, RGB, HSL/HSV, CMY/CMYK, and many more.
- Conversions between the various color spaces. For example, XYZ to sRGB, Spectral to XYZ, CIE Lab to Adobe RGB.
- Calculation of color difference. All CIE Delta E functions, plus CMC.
- Chromatic adaptations (changing illuminants).
- RGB to hex and vice-versa.
- 16-bit RGB support.
- Runs on Python 2.7 and Python 3.3+.

License: python-colormath is licensed under the BSD License.
Assorted Info

- Issue tracker - Report bugs, ask questions, and share ideas here.
- GitHub project - Source code and issue tracking.
- @getaylor Twitter - Tweets from the maintainer.
- Greg Taylor's blog - Occasional posts about color math and software development.
2.1 Installation

python-colormath currently requires Python 2.7 or Python 3.3+. There are no plans to add support for earlier versions of Python 2 or 3. The only other requirement is NumPy.

For those on Linux/Unix Mac OS, the easiest route will be pip or easy_install:

```bash
pip install colormath
```

If you are on Windows, you’ll need to visit NumPy, download their binary distribution, then install colormath.

2.2 Color Objects

python-colormath has support for many of the commonly used color spaces. These are represented by Color objects.
2.2.1 SpectralColor

2.2.2 LabColor

2.2.3 LCHabColor

2.2.4 LCHuvColor

2.2.5 LuvColor

2.2.6 XYZColor

2.2.7 xyYColor

2.2.8 sRGBColor

2.2.9 AdobeRGBColor

2.2.10 HSLColor

2.2.11 HSVColor

2.2.12 CMYColor

2.2.13 CMYKColor

2.2.14 IPTColor

2.3 Observers and Illuminants

Illuminants and observer angles are used in all color spaces that use reflective (instead of transmissive) light. Here are a few brief overviews of what these are and what they do:

- Understanding Standard Illuminants in Color Measurement - Konica Minolta
- What is Meant by the Term “Observer Angle”? - XRite

To adjust the illuminants and/or observer angles on a color:

```python
color = LabColor(0.1, 0.2, 0.3, observer='10', illuminant='d65')
```

2.3.1 Two-degree observer angle

These illuminants can be used with observer='2', for the color spaces that require illuminant/observer:

- 'a'
- 'b'
- 'c'
- 'd50'
- 'd55'
2.3.2 Ten-degree observer angle

These illuminants can be used with \texttt{observer='10'}, for the color spaces that require illuminant/observer:

- 'd50'
- 'd55'
- 'd65'
- 'd75'

2.4 Color Conversions

Converting between color spaces is very simple with python-colormath. To see a full list of supported color spaces, see \textit{Color Objects}.

All conversions happen through the \texttt{convert\_color} function shown below. The original Color instance is passed in as the first argument, and the desired Color class (not an instance) is passed in as the second argument. If the conversion can be made, a new Color instance will be returned.

2.4.1 Example

This is a simple example of a CIE Lab to CIE XYZ conversion. Refer to \textit{Color Objects} for a full list of different color spaces that can be instantiated and converted between.

```python
from colormath.color_objects import LabColor, XYZColor
from colormath.color_conversions import convert_color

lab = LabColor(0.903, 16.296, -2.22)
xyz = convert_color(lab, XYZColor)
```

Some color spaces require a trip through RGB during conversion. For example, to get from XYZ to HSL, we have to convert XYZ->RGB->HSL. The same could be said for XYZ to CMYK (XYZ->RGB->CMY->CMYK). Different RGB color spaces have different gamut sizes and capabilities, which can affect your converted color values.

sRGB is the default RGB color space due to its ubiquity. If you would like to use a different RGB space for a conversion, you can do something like this:

```python
from colormath.color_objects import XYZColor, HSLColor, AdobeRGBColor
from colormath.color_conversions import convert_color

xyz = XYZColor(0.1, 0.2, 0.3)
hsl = convert_color(xyz, HSLColor, through_rgb_type=AdobeRGBColor)
# If you are going to convert back to XYZ, make sure you use the same
```
# RGB color space on the way back.

```python
xyz2 = convert_color(hsl, XYZColor, through_rgb_type=AdobeRGBColor)
```

## 2.4.2 RGB conversions and native illuminants

When converting RGB colors to any of the CIE spaces, we have to pass through the XYZ color space. This serves as a crossroads for conversions to basically all of the reflective color spaces (CIE Lab, LCH, Luv, etc). The RGB spaces are reflective, where the illumination is provided. In the case of a reflective space like XYZ, the illuminant must be supplied by a light source.

Each RGB space has its own native illuminant, which can vary from space to space. To see some of these for yourself, check out Bruce Lindbloom’s XYZ to RGB matrices.

To cite the most commonly used RGB color space as an example, sRGB has a native illuminant of D65. When we convert RGB to XYZ, that native illuminant carries over unless explicitly overridden. If you aren’t expecting this behavior, you’ll end up with variations in your converted color’s numbers.

To explicitly request a specific illuminant, provide the `target_illuminant` keyword when using `colormath.color_conversions.convert_color()`.

```python
from colormath.color_objects import XYZColor, sRGBColor
from colormath.color_conversions import convert_color

rgb = RGBColor(0.1, 0.2, 0.3)
xyz = convert_color(rgb, XYZColor, target_illuminant='d50')
```

## 2.4.3 RGB conversions and out-of-gamut coordinates

RGB spaces tend to have a smaller gamut than some of the CIE color spaces. When converting to RGB, this can cause some of the coordinates to end up being out of the acceptable range (0.0-1.0 or 1-255, depending on whether your RGB color is upscaled).

Rather than clamp these for you, we leave them as-is. This allows for more accurate conversions back to the CIE color spaces. If you require the clamped (0.0-1.0 or 1-255) values, use the following properties on any RGB color:

- `clamped_rgb_r`
- `clamped_rgb_g`
- `clamped_rgb_b`

## 2.5 Delta E Equations

Delta E equations are used to put a number on the visual difference between two `LabColor` instances. While different lighting conditions, substrates, and physical condition can all introduce unexpected variables, these equations are a good rough starting point for comparing colors.

Each of the following Delta E functions has different characteristics. Some may be more suitable for certain applications than others. While it’s outside the scope of this module’s documentation to go into much detail, we link to relevant material when possible.
2.5.1 Example

```python
from colormath.color_objects import LabColor
from colormath.color_diff import delta_e_cie1976

# Reference color.
color1 = LabColor(lab_l=0.9, lab_a=16.3, lab_b=-2.22)
# Color to be compared to the reference.
color2 = LabColor(lab_l=0.7, lab_a=14.2, lab_b=-1.80)
# This is your delta E value as a float.
delta_e = delta_e_cie1976(color1, color2)
```

2.5.2 Delta E CIE 1976

2.5.3 Delta E CIE 1994

2.5.4 Delta E CIE 2000

2.5.5 Delta E CMC

2.6 ANSI and ISO Density

Density may be calculated from LabColor instances.

2.6.1 Example

```python
from colormath.color_objects import SpectralColor
from colormath.density import auto_density, ansi_density
from colormath.density_standards import ANSI_STATUS_T_RED

# Omitted the full spectral kwargs for brevity.
color = SpectralColor(spec_340nm=0.08, ...)
# ANSI T Density for the spectral color.
density = auto_density(color)

# Or maybe we want to specify which filter to use.
red_density = ansi_density(color, ANSI_STATUS_T_RED)
```

2.6.2 Valid Density Constants

The following density constants within colormath.density_standards can be passed to colormath.density.anqi_density():

- ANSI_STATUS_A_RED
- ANSI_STATUS_A_GREEN
- ANSI_STATUS_A_BLUE
- ANSI_STATUS_E_RED
- ANSI_STATUS_E_GREEN

2.6. ANSI and ISO Density
2.7 Color Appearance Models

Color appearance models allow the prediction of perceptual correlates (e.g., lightness, chroma or hue) of a given surface color under certain viewing conditions (e.g., a certain illuminant, surround or background). The complexity of color appearance models can range from very low, e.g., CIELAB can technically be considered a color appearance model, to very complex models that take into account a large number color appearance phenomena.

Each of the classes in this module represents a specific model and its computation, yielding the predicted perceptual correlates as instance attributes. Discussing the details of each model go beyond this documentation, but we provide references to the relevant literature for each model and would advice familiarising yourself with it, before using a given models.

2.7.1 Example

```python
# Color stimulus
color = XYZColor(19.01, 20, 21.78)

# The two illuminants that will be compared.
illuminant_d65 = XYZColor(95.05, 100, 108.88)
illuminant_a = XYZColor(109.85, 100, 35.58)

# Background relative luminance
y_b = 20

# Adapting luminance
l_a = 328.31

# Surround condition assumed to be average (see CIECAM02 documentation for values)
c = 0.69
n_c = 1
f = 1

model = CIECAM02(color.xyz_x, color.xyz_y, color.xyz_z,
                  illuminant_d65.xyz_x, illuminant_d65.xyz_y, illuminant_d65.xyz_z,
                  y_b, l_a, c, n_c, f)
```
2.7.2 Nayatani95 et al. Model

2.7.3 Hunt Model

2.7.4 RLAB Model

2.7.5 ATD95 Model

2.7.6 LLAB Model

2.7.7 CIECAM02 Model

2.7.8 CIECAM02-m1 Model

2.8 Release Notes

2.8.1 2.1.0 (Unreleased)

Features

- Added the IPT color space. (MichaelMauderer)
- Added Color Appearance Models. Natayani95, Hunt, RLAB, ATD95, LLAB, CIECAM02, CIECAM02-m1. (MichaelMauderer)

Bug Fixes

- xyY conversions now correctly avoid division by zero. (dwbullok)
- Un-transposed adaptation matrices. Has no effect on conversions, but if you use these directly you may see different numbers. (JasonTam)

2.8.2 2.0.2

Bug Fixes

- Apparently I didn’t add the function body for the clamped RGB properties. Yikes.

2.8.3 2.0.1

Features

- Lots of documentation improvements.
- `convert_color()` now has an explicitly defined/documented target_illuminant kwarg, instead of letting this fall through to its **kwargs. This should make IDE auto-complete better and provide more clarity.
- Added clamped_rgb_r, clamped_rgb_g, and clamped_rgb_b to RGB color spaces. Use these if you have to have in-gamut, potentially compressed coordinates.
Bug Fixes

- Direct conversions to non-sRGB colorspaces returned sRGBColor objects. Reported by Cezary Wagner.

2.8.4 2.0.0

Backwards Incompatible changes

- Minimum Python version is now 2.7.
- ColorBase.convert_to() is no more. Use colormath.color_conversions.convert_color() instead. API isn’t as spiffy looking, but it’s a lot less magical now.
- Completely re-worked RGB handling. Each RGB space now has its own class, inheriting from BaseRGBColor. Consequently, RGBColor is no more. In most cases, you can substitute RGBColor with sRGBColor during your upgrade.
- RGB channels are now [0..1] instead of [1..255]. Can use BaseRGBColor.get_upscaled_value_tuple() to get the upscaled values.
- BaseRGBColor.set_from_rgb_hex() was replaced with a BaseRGBColor.new_from_rgb_hex(), which returns a properly formed sRGBColor object.
- BaseRGBColor no longer accepts observer or illuminant kwargs.
- HSL no longer accepts observer, illuminant or rgb_type kwargs.
- HSV no longer accepts observer, illuminant or rgb_type kwargs.
- CMY no longer accepts observer, illuminant or rgb_type kwargs.
- CMYK no longer accepts observer, illuminant or rgb_type kwargs.
- Removed ‘debug’ kwargs in favor of Python’s logging.
- Completely re-worked exception list. Eliminated some redundant exceptions, re-named basically everything else.

Features

- Python 3.3 support added.
- Added tox.ini with supported environments.
- Removed the old custom test runner in favor of nose.
- Replacing simplified RGB->XYZ conversions with Bruce Lindbloom’s.
- A round of PEP8 work.
- Added a BaseColorConversionTest test class with some greatly improved color comparison. Much more useful in tracking down breakages.
- Eliminated non-matrix delta E computations in favor of the matrix equivalents. No need to maintain duplicate code, and the matrix stuff is faster for bulk operations.
Bug Fixes

- Corrected delta_e CMC example error. Should now run correctly.
- color_diff_matrix.delta_e_cie2000 had an edge case where certain angles would result in an incorrect delta E.
- Un-hardcoded XYZColor.apply_adaptation()’s adaptation and observer angles.

2.8.5 1.0.9

Features

- Added an optional vectorized deltaE function. This uses NumPy array/matrix math for a very large speed boost. (Eddie Bell)
- Added this changelog.

Bug Fixes

- Un-hardcode the observer angle in adaptation matrix. (Bastien Dejean)

2.8.6 1.0.8

- Initial GitHub release.
Useful color math resources

- Bruce Lindbloom - Lots of formulas, calculators, and standards.
- John the Math Guy - Useful tutorials and explanations of color theory.