colormap

Release 1.0.4

Thomas Cokelaer

Oct 16, 2021

Contents

build passing

coverage 97%

version Python 3.6, 3.7, 3.8, 3.9

contributions Please join https://github.com/cokelaer/colormap

issues Please use https://github.com/cokelaer/colormap/issues

notebook Please see https://github.com/cokelaer/colormap/tree/master/notebooks

CHAPTER 1

What is it ?

colormap package provides simple utilities to convert colors between RGB, HEX, HLS, HUV and a class to easily build colormaps for matplotlib. All matplotlib colormaps and some R colormaps are available altogether. The plot_colormap method (see below) is handy to quickly pick up a colormaps and the test_colormap function is useful to see/test a new colormap.

CHAPTER 2

Installation

2.1 Prerequisites

You will need to install Python (linux and mac users should have it installed already). We recommend also to install ipython, which provides a more flexible shell alternative to the python shell itself. **colormap** requires **matplotlib** and **easydev**, which are available on pypi and installed automatically with this package.

2.2 Installation

Since **colormap** is available on PyPi, the following command should install the package and its dependencies automatically:

```
pip install colormap
```

colormap is also available on Conda (conda-forge):

```
conda install colormap
```

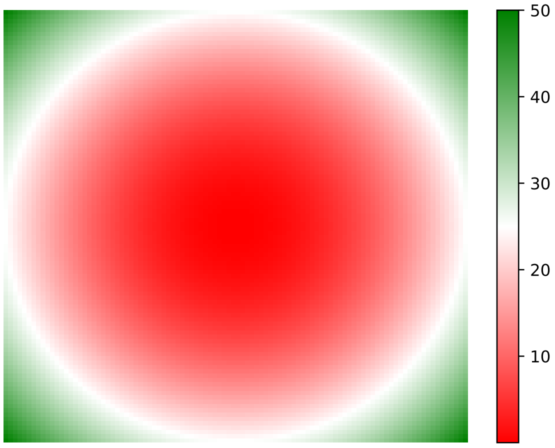
CHAPTER 3

Examples

3.1 Using the `Colormap` class

Create your own colormap from red to green colors with intermediate color as whitish (diverging map from red to green):

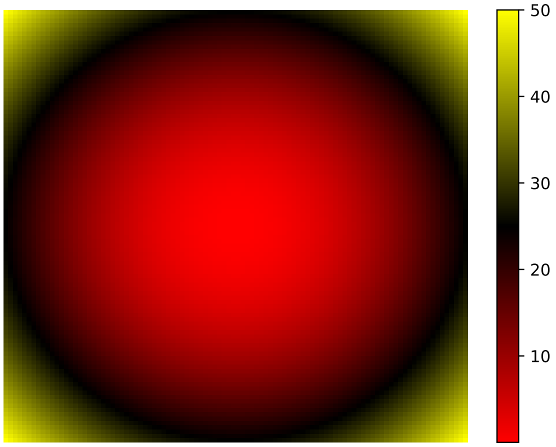
```python
from colormap import Colormap
c = Colormap()
mycmap = c.cmap_linear('red', 'white', 'green(w3c)')
c.test_colormap(mycmap)
```



3.2 Using the aliases

Without creating an instance of **Colormap**, you can use these functions:

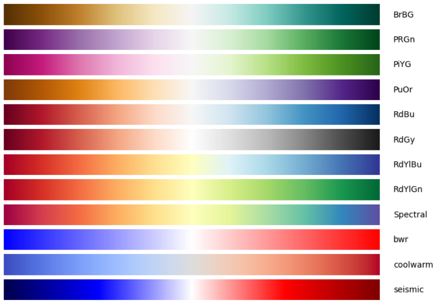
```
from colormap import cmap_builder, test_cmap
mycm = cmap_builder('red', 'black', 'yellow')
test_cmap(mycm)
```



3.3 Visualise set of colormap

Another convenient feature is to look at a set of colormaps altogether:

```
c = Colormap()
c.plot_colormap('diverging')
```



Other set names are:

- sequentials2,
- misc
- diverging
- diverging_black

- qualitative

See user guide for details.

CHAPTER 4

User Guide

4.1 Conventions

4.1.1 hexadecimal

hexadecimal can be encoded as explained in `colormap.colors.hex2rgb()`:

- #FFF
- #0000FF
- 0x0000FF
- 0xFA1

4.1.2 normalisation

By default, input should be normalised (e.g., RGB values between 0 and 1) and outputs are normalised. If you provide unnormalised values (e.g., RGB in 0-255) then set the noramlised parameter to True (see example in codecs).

4.2 Codecs

4.2.1 list

There is a bunch of codecs available in `colormap.colors` such as hex2rgb:

```
>>> from colormap.colors import hex2rgb
>>> hex2rgb("#FFF", normalise=False)
(255, 255, 255)
>>> hex2rgb("#FFFFFF", normalise=True)
(1.0, 1.0, 1.0)
```

codecs	
hex2web	*colormap.colors.hex2web()*
web2hex	*colormap.colors.web2hex()*
hex2rgb	*colormap.colors.hex2rgb()*
rgb2hex	*colormap.colors.rgb2hex()*
rgb2hls	*colormap.colors.rgb2hls()*
rgb2hsv	*colormap.colors.rgb2hsv()*
hsv2rgb	*colormap.colors.hsv2rgb()*
hls2rgb	*colormap.colors.hls2rgb()*
hex2dec	*colormap.colors.hex2dec()*
yuv2rgb	*colormap.colors.yuv2rgb()*
rgb2yuv_int	*colormap.colors.rgb2yuv_int()*
yuv2rgb_int	*colormap.colors.yuv2rgb_int()*

4.2.2 format

- RGB (red/green/blue): a triple of values between 0 and 255
- HLS (): H in 0-360 and L,S in 0-100
- HSV (): H in 0-360, S,V in
- YUV: all in 0-1

4.3 Color class

On task, which is quite common is to know the hexadecimal code of a color known by name (e.g. red). The *colormap.colors.Color* would be useful:

```
>>> c = Color('red')
>>> c.rgb
(1.0, 0.0, 0.0)
>>> c.hls
(0.0, 0.5, 1.0)
>>> c.hex
'#FF0000'

>>> print(c)
Color Red
hexa code: #FF0000
RGB code: (1.0, 0.0, 0.0)
RGB code (un-normalised): [255.0, 0.0, 0.0]

HSV code: (0.0, 1.0, 1.0)
HSV code: (un-normalised) 0.0 100.0 100.0

HLS code: (0.0, 0.5, 1.0)
HLS code: (un-normalised) 0.0 50.0 100.0
```

Input when instanciating can be anything in RGB, HEX, HLS, common name from *colormap.xfree86*:

```
>>> sorted(colormap.xfree86.XFree86_colors.keys())
```

4.4 colormap

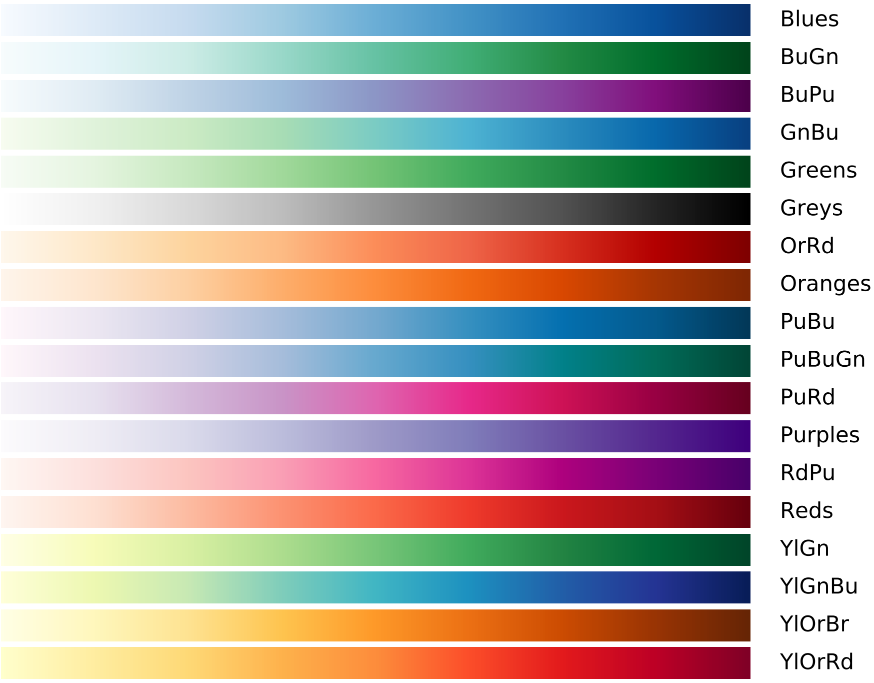
There are lots of colormap in matplotlib. This is great but some may be missing or it is not obvious to know what the colormap will look like.

The `colormap.colors.Colormap` class allows you:

- To build easily new colormaps and visualise them
- Visualise existing colormaps

4.4.1 visualise colormaps

```
>>> from colormap import Colormap
>>> c = Colormap()
>>> c.plot_colormap('sequentials')
```



Try with other sets:

- sequentials2,
- misc
- diverging
- qualitative

4.4.2 Create a linear colormap

The simplest colormap are linear with 3 colors. In such case, we provide a method that is easy to use. Imagine you want a colormap from red to green with white color in between:

```
c = Colormap()
cmap = cmap_linear('red', 'white', 'green')
c.test_colormap(cmap)
```

Here, we use color names, which are the xfree86 names. However, you could have used any format accepted by `Colors`:

```
red = Color('red')
cmap = cmap_linear(red, 'white', '#0000FF')
```

4.4.3 Create a general colormap

In the previous example, we used 3 colors assuming a linear scale. However, you may want a different scale, in which case, you need to provide more colors. In such case, you can use *cmap()* method.

Here we again use the same example a above but it can be generalised easily. First, we need to know the RGB components of the colors:

```
>>> from colormap import Color, Colormap
>>> green = Color('Dark Green').rgb
>>> red = Color('red').rgb
>>> white = Color('white').rgb
>>> white
(1.0, 1.0, 1.0)
```

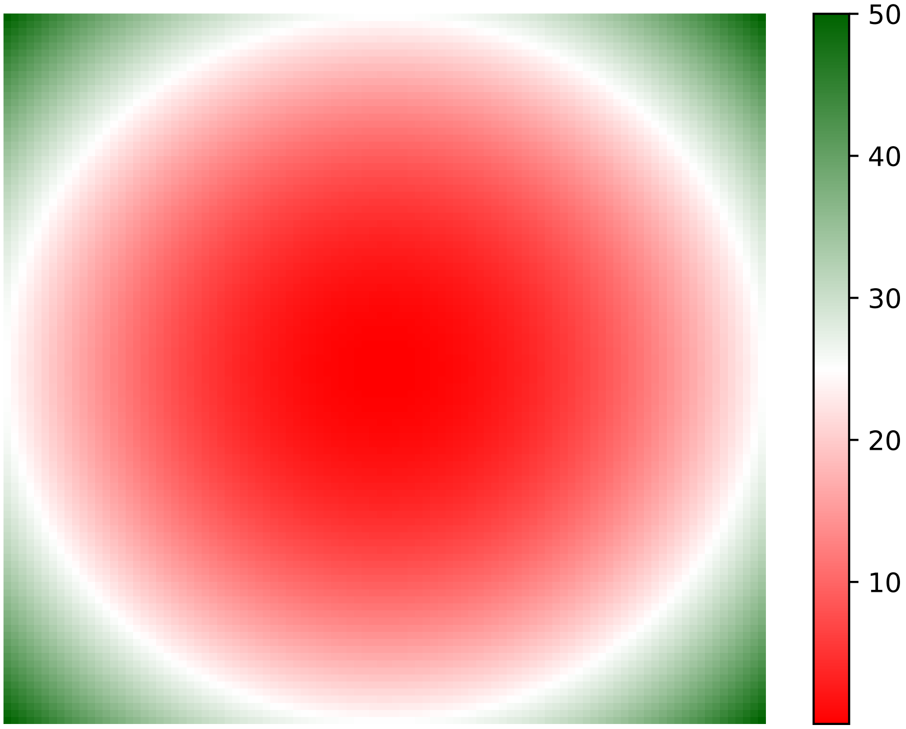
For instance RGB values of white are 1,1,1

Second, built a dictionary with the three RGB name (red/green/blue) as keys and with the values being the evolution of the red/green/blue when a value goes from 0 to 1. Here, we use a linear scaling so we just need 3 values at 0, 0.5, and 1. Therefore we have list of 3 values. You could provide list of arbitrary lengths if required

```
>>> c = Colormap()
>>> mycmap = c.cmap( {'red':[1,1,0], 'green':[0,1,.39],  'blue':[0,1,0]})
```

Finally, test it:

```
c.test_colormap(mycmap)
```


50
40
30
20
10

CHAPTER 5

Reference Guide

5.1 Reference guide

Contents

5.1.1 colors

Utilities provided in this module can be found either in the standard Python module called `colorsys` or in matplotlib.colors (e.g rgb2hex) or are original to this module (e.g., rgb2huv)

class HEX

Class to check the validity of an hexadecimal string and get standard string

By standard, we mean #FFFFFF (6 digits)

```
>>> h = HEX()
>>> h.is_valid_hex_color("#FFFF00")
True
```

get_standard_hex_color(*value*)

Return standard hexadecimal color

By standard, we mean a string that starts with # sign followed by 6 character, e.g. #AABBFF

is_valid_hex_color(*value*, *verbose=True*)

Return True is the string can be interpreted as hexadecimal color

Valid formats are

- #FFF
- #0000FF
- 0x0000FF
- 0xFA1

class Color(*name=None*, *rgb=None*, *hls=None*, *hsv=None*)

Class to ease manipulation and conversion between color codes

You can create an instance in many differen ways. You can either use a human-readable name as long as it is part of the XFree86 list You can also provide a hexadecimal string (either 3 or 6 digits). You can use triplets of values corresponding to the RGB, HSV or HLS conventions.

Here are some examples:

```
from colormap import Color

Color("red")            # human XFree86 compatible representation
Color("#f00")           # standard 3 hex digits
Color("#ff0000")        # standard 6 hex digits
Color(hsv=(0,1,0.5))
Color(hls=(0, 1, 0.5)) # HLS triplet
Color(rgb=(1, 0, 0))   # RGB triplet
Color(Color("red"))    # using an instance of :class:`Color`
```

Note that the RGB, HLS and HSV triplets use normalised values. If you need to normalise the triplet, you can use `colormap.colors._normalise` that provides a function to normalise RGB, HLS and HSV triplets:

```
colors._normalise(*(255, 255, 0), mode="rgb")
colors._normalise(*(360, 50, 100), mode="hls")
```

If you provide a string, it has to be a valid string from XFree86. In addition to the official names, the lower case names are valid. Besides, there are names with spaces. The equivalent names without space are also valid. Therefore the name “Spring Green”, which is an official name can be provided as “Spring Green”, “spring green”, “springgreen” or “SpringGreen”.

blue

getter/setter for the blue color in RGB triplet

get_standard_hex_color(*value*)

Return standard hexadecimal color

By standard, we mean a string that starts with # sign followed by 6 character, e.g. #AABBFF

green

getter/setter for the green color in RGB triplet

hex

getter/setter the hexadecimal value.

hls

getter/setter the HLS values (3-length tuple)

hsv

getter/setter the HSV values (3-length tuple)

hue

getter/setter the saturation in the HLS triplet

is_valid_hex_color (*value*, *verbose=True*)

Return True is the string can be interpreted as hexadecimal color

Valid formats are

- #FFF
- #0000FF
- 0x0000FF
- 0xFA1

lightness

getter/setter the lightness in the HLS triplet

red

getter/setter for the red color in RGB triplet

rgb

getter/setter the RGB values (3-length tuple)

saturation_hls

getter/setter the saturation in the HLS triplet

value

getter/setter the value in the HSV triplet

yiq

Getter for the YIQ triplet

hex2web (*hexa*)

Convert hexadecimal string (6 digits) into *web* version (3 digits)

```
>>> from colormap.colors import hex2web
>>> hex2web("#FFAA11")
'#FA1'
```

See also:

web2hex(), *hex2rgb() rgb2hex()*, *rgb2hsv()*, *hsv2rgb()*, *rgb2hls()*, *hls2rgb()*

web2hex (*web*)

Convert *web* hexadecimal string (3 digits) into 6 digits version

```
>>> from colormap.colors import web2hex
>>> web2hex("#FA1")
'#FFAA11'
```

See also:

hex2web(), *hex2rgb() rgb2hex()*, *rgb2hsv()*, *hsv2rgb()*, *rgb2hls()*, *hls2rgb()*

hex2rgb (*hexcolor*, *normalise=False*)

This function converts a hex color triplet into RGB

Valid hex code are:

- #FFF
- #0000FF

- 0x0000FF
- 0xFA1

```
>>> from colormap.colors import hex2rgb
>>> hex2rgb("#FFF", normalise=False)
(255, 255, 255)
>>> hex2rgb("#FFFFFF", normalise=True)
(1.0, 1.0, 1.0)
```

See also:

hex2web(), *web2hex()*, *rgb2hex()*, *rgb2hsv()*, *hsv2rgb()*, *rgb2hls()*, *hls2rgb()*

hex2dec(*data*)

convert hexadecimal string (data) into a float in the [0-65536] inclusive range

rgb2hex(*r*, *g*, *b*, *normalised=False*)

Convert RGB to hexadecimal color

Param can be a tuple/list/set of 3 values (R,G,B)

Returns a hex vesion ofthe RGB 3-tuple

```
>>> from colormap.colors import rgb2hex
>>> rgb2hex(0,0,255, normalised=False)
'#0000FF'
>>> rgb2hex(0,0,1, normalised=True)
'#0000FF'
```

See also:

hex2web(), *web2hex()*, *hex2rgb()* , *rgb2hsv()*, *hsv2rgb()*, *rgb2hls()*, *hls2rgb()*

rgb2hsv(*r*, *g*, *b*, *normalised=True*)

Convert an RGB value to an HSV value.

Parameters **normalised** (*bool*) – if *normalised* is True, the input RGB triplet should be in the range 0-1 (0-255 otherwise)

Returns the HSV triplet. If *normalised* parameter is True, the output triplet is in the range 0-1; otherwise, H in the range 0-360 and LS in the range 0-100.

```
>>> from colormap.colors import rgb2hsv
>>> rgb2hsv(0.5,0,1)
(0.75, 1, 1)
```

See also:

hex2web(), *web2hex()*, *hex2rgb()* *rgb2hex()*, *hsv2rgb()*, *rgb2hls()*, *hls2rgb()*

hsv2rgb(*h*, *s*, *v*, *normalised=True*)

Convert a hue-saturation-value (HSV) value to a red-green-blue (RGB).

Parameters **normalised** (*bool*) – If *normalised* is True, the input HSV triplet should be in the range 0-1; otherwise, H in the range 0-360 and LS in the range 0-100.

Returns the RGB triplet. The output triplet is in the range 0-1 whether the input is normalised or not.

```
>>> from colormap.colors import hsv2rgb
>>> hsv2rgb(0.5,1,1, normalised=True)
(0, 1, 1)
```

See also:

hex2web(), *web2hex()*, *hex2rgb()* *rgb2hex()*, *rgb2hsv()*, *rgb2hls()*, *hls2rgb()*

See also:

rgb2hex()

rgb2hls(*r*, *g*, *b*, *normalised=True*)

Convert an RGB value to an HLS value.

Parameters **normalised** (*bool*) – if *normalised* is True, the input RGB triplet should be in the range 0-1 (0-255 otherwise)

Returns the HLS triplet. If *normalised* parameter is True, the output triplet is in the range 0-1; otherwise, H in the range 0-360 and LS in the range 0-100.

```
>>> from colormap.colors import rgb2hls
>>> rgb2hls(255,255,255, normalised=False)
(0.0, 1.0, 0.0)
```

See also:

hex2web(), *web2hex()*, *hex2rgb()* *rgb2hex()*, *hsv2rgb()*, *hls2rgb()*

hls2rgb(*h*, *l*, *s*, *normalised=True*)

Convert an HLS value to a RGB value.

Parameters **normalised** (*bool*) – If *normalised* is True, the input HLS triplet should be in the range 0-1; otherwise, H in the range 0-360 and LS in the range 0-100.

Returns the RGB triplet. The output triplet is in the range 0-1 whether the input is normalised or not.

```
>>> from colormap.colors import hls2rgb
>>> hls2rgb(360, 50, 60, normalised=False)
(0.8, 0.2, 0.2)
```

See also:

hex2web(), *web2hex()*, *hex2rgb()* *rgb2hex()*, *rgb2hsv()*, *hsv2rgb()*, *rgb2hls()*,

yuv2rgb(*y*, *u*, *v*)

Convert YUV triplet into RGB

YUV

> **Warning:** expected input must be between 0 and 255 (not normalised)

rgb2yuv(*r*, *g*, *b*)

Convert RGB triplet into YUV

Returns YUV triplet with values between 0 and 1

YUV wikipedia

> **Warning:** expected input must be between 0 and 1

Note: the constants referenc used is Rec. 601

to_intensity(*n*)

Return intensity

Parameters **n** – value between 0 and 1

Returns value between 0 and 255; round(n*127.5+127.5)

yuv2rgb_int(*y*, *u*, *v*)

Convert YUV triplet into RGB

YUV

Warning: expected input must be between 0 and 255 (not normalised)

rgb2yuv_int(*r*, *g*, *b*)

Convert RGB triplet into YUV

YUV wikipedia

Warning: expected input must be between 0 and 255 (not normalised)

class Colormap

Class to create matplotlib colormap

This example show how to get the pre-defined colormap called *heat*

```
from pylab import *
from colormap.colors import Colormap

c = Colormap()
cmap = c.get_cmap_heat()
c.test_colormap(cmap)
```

You may be more interested in building your own colormap:

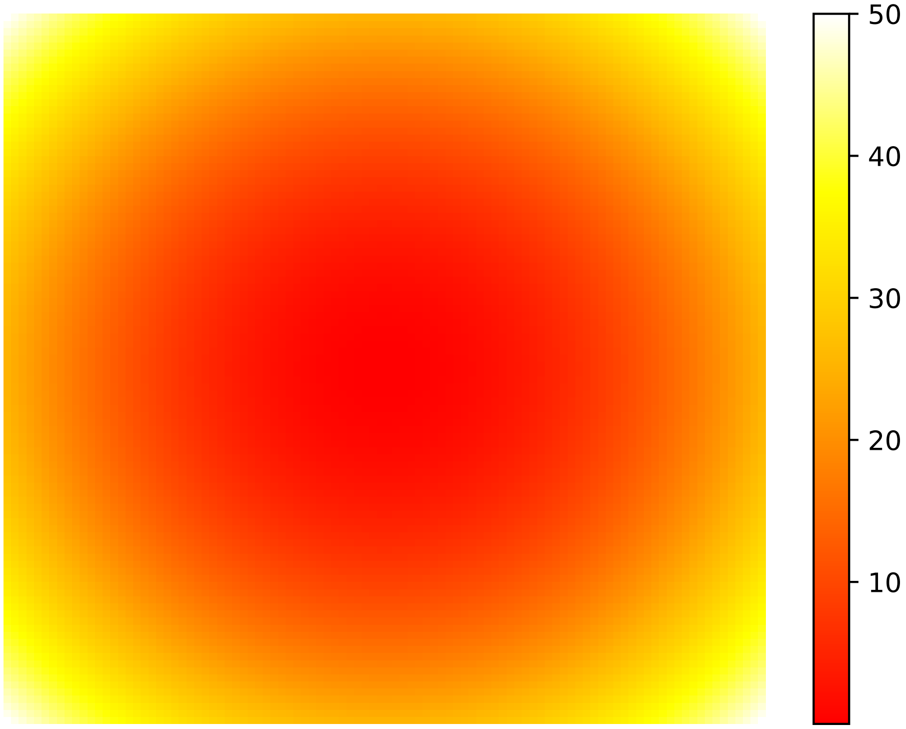
```
# design your own colormap
d = {'blue': [0,0,0,1,1,1,0],
        'green':[0,1,1,1,0,0,0],
        'red':   [1,1,0,0,0,1,1]}
cmap = c.cmap(d, reverse=False)

# see the results
c.test_colormap(cmap)
```

If you want a simple linear colormap, you can use the example above, or use the `cmap_linear()`. For instance for a diverging colormap from red to green (with with color in between):

```
cmap = c.cmap_linear("red", "white", "green")
c.test_colormap(cmap)
```

Even simpler, you can use a bicolor colormap `cmap_bicolor()`. For instance for a red to green colormap:


50
40
30
20
10

```
cmap = c.cmap_bicolor("red", "green")
c.test_colormap(cmap)
```

From matplotlib documentation, colormaps falls into 4 categories:

1. Sequential schemes for unipolar data that progresses from low to high
2. Diverging schemes for bipolar data that emphasizes positive or negative deviations from acentral value
3. Cyclic schemes meant for plotting values that wrap around at the endpoints, such as phase angle, wind direction, or time of day
4. Qualitative schemes for nominal data that has no inherent ordering, where color is used only to distinguish categories

References matplotlib documentation and examples http://matplotlib.org/examples/color/colormaps_reference.html

cmap(*colors=None*, *reverse=False*, *N=256*)
Return a colormap object to be used within matplotlib

Parameters

- **colors** (*dict*) – a dictionary that defines the RGB colors to be used in the colormap. See `get_cmap_heat()` for an example.
- **reverse** (*bool*) – reverse the colormap is set to True (defaults to False)
- **N** (*int*) – Defaults to 50

cmap_bicolor(*color1*, *color2*, *reverse=False*, *N=256*)
Provide 3 colors in format accepted by `Color`

```
>>> red = Color('red')
>>> white = Color('white')
>>> cmap = cmap_bicolor(red, white)
```

cmap_linear(*color1*, *color2*, *color3*, *reverse=False*, *N=256*)
Provide 3 colors in format accepted by `Color`

```
red = Color('red')
cmap = cmap_linear(red, 'white', '#0000FF')
```

get_cmap_heat()
Return a heat colormap matplotlib-compatible colormap

This heat colormap should be equivalent to heat.colors() in R.

```
>>> from colormap.colors import Colormap
>>> cmap = Colormap.get_cmap_heat()
```

You can generate the colormap based solely on this information for the RGB functions along:

```
d=  {   'blue':[0,0,0,0,1],
        'green':[0,.35,.7,1,1],
        'red':[1,1,1,1,1]}
cmap = Colormap.get_cmap(d)
```

get_cmap_heat_r()
Return a heat colormap matplotlib-compatible colormap

Same as *get_cmap_heat()* but reversed

get_cmap_rainbow()

colormap similar to rainbow colormap from R

Note: The red is actually appearing on both sides... Yet this looks like what is coded in R 3.0.1

plot_colormap(*cmap_list=None*)

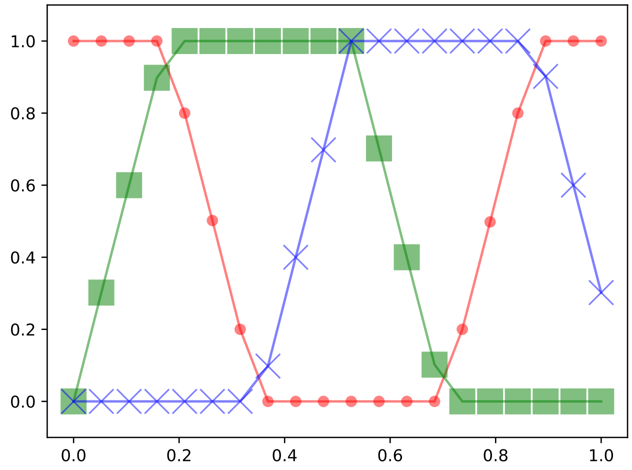
cmap_list list of valid cmap or name of a set (sequential, diverging,)

if none, plot all known colors

plot_rgb_from_hex_list(*cols*)

This functions takes a list of hexadecimal values and plots the RGB curves. This can be handy to figure out the RGB functions to be used in the `get_cmap()`.

```
from colormap.colors import Colormap
c = Colormap()
t = ['#FF0000FF', '#FF4D00FF', '#FF9900FF', '#FFE500FF',
     '#CCFF00FF', '#80FF00FF', '#33FF00FF', '#00FF19FF',
     '#00FF66FF', '#00FFB2FF', '#00FFFFFF', '#00B3FFFF',
     '#0066FFFF', '#001AFFFF', '#3300FFFF', '#7F00FFFF',
     '#CC00FFFF','#FF00E6FF','#FF0099FF', '#FF004DFF']
c.plot_rgb_from_hex_list(t)
```



test_colormap(*cmap=None*)

plot one colormap for testing

By default, test the *get_cmap_heat()*

5.1.2 cmap module

5.1.3 xfree86 module

Python Module Index

Index