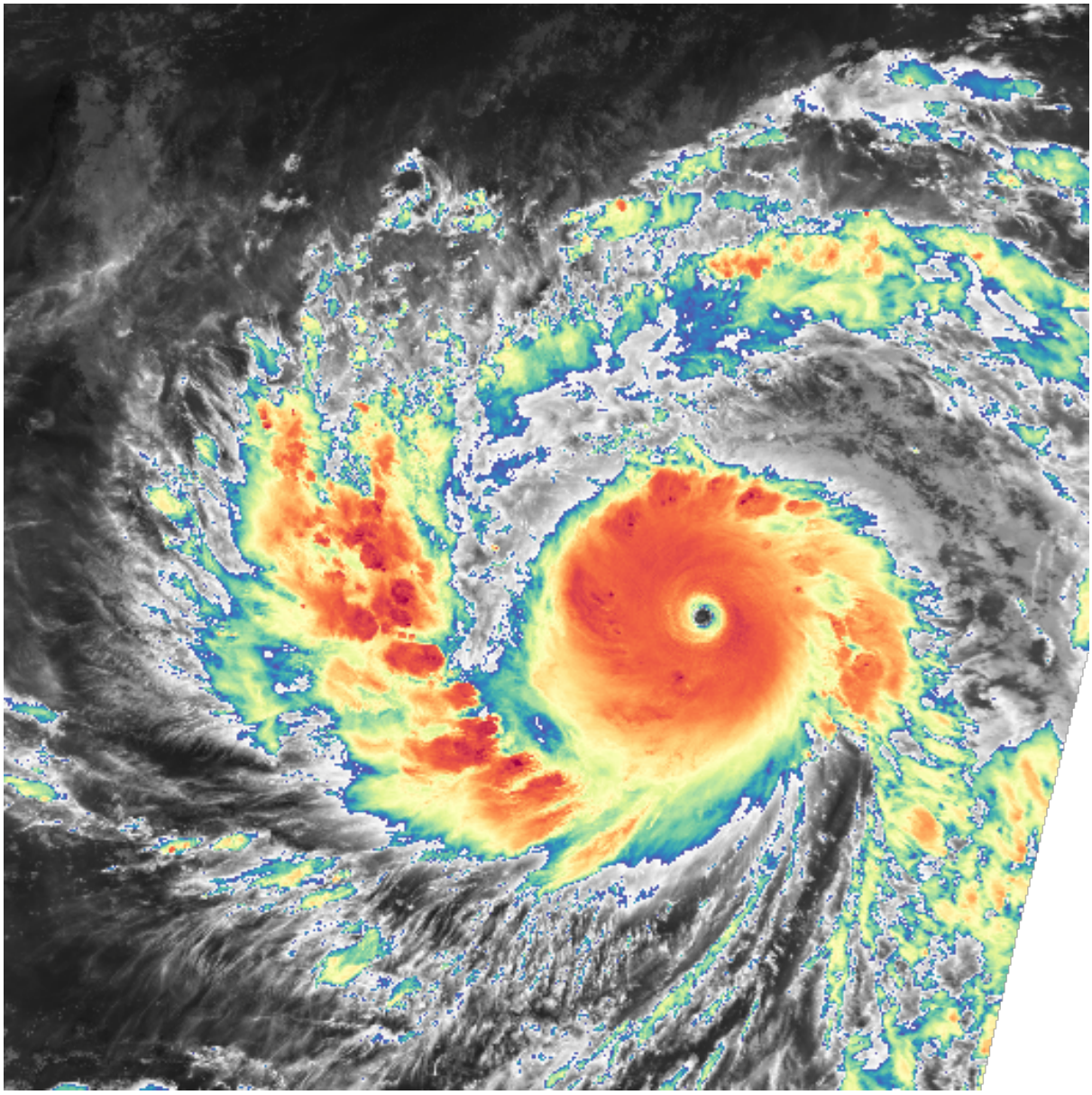
TrollImage Documentation

Release v0.4.0

The Pytroll Team

November 25, 2014

Contents



Get the source code here !

Contents:

CHAPTER 1

Simple images

This module defines the image class. It overlaps largely the PIL library, but has the advandage of using masked arrays as pixel arrays, so that data arrays containing invalid values may be properly handled.

class `trollimage.image.`**Image**(*channels=None*, *mode='L'*, *color_range=None*, *fill_value=None*, *palette=None*)

This class defines images. As such, it contains data of the different *channels* of the image (red, green, and blue for example). The *mode* tells if the channels define a black and white image ("L"), an rgb image ("RGB"), an YCbCr image ("YCbCr"), or an indexed image ("P"), in which case a *palette* is needed. Each mode has also a corresponding alpha mode, which is the mode with an "A" in the end: for example "RGBA" is rgb with an alpha channel. *fill_value* sets how the image is filled where data is missing, since channels are numpy masked arrays. Setting it to (0,0,0) in RGB mode for example will produce black where data is missing."None" (default) will produce transparency (thus adding an alpha channel) if the file format allows it, black otherwise.

The channels are considered to contain floating point values in the range [0.0,1.0]. In order to normalize the input data, the *color_range* parameter defines the original range of the data. The conversion to the classical [0,255] range and byte type is done automagically when saving the image to file.

blend(*other*)
Alpha blend *other* on top of the current image.

clip(*channels=True*)
Limit the values of the array to the default [0,1] range. *channels* says which channels should be clipped.

colorize(*colormap*)
Colorize the current image using *colormap*. Works only on"L" or "LA" images.

convert(*mode*)
Convert the current image to the given *mode*. See `Image` for a list of available modes.

crude_stretch(*ch_nb*, *min_stretch=None*, *max_stretch=None*)
Perform simple linear stretching (without any cutoff) on the channel *ch_nb* of the current image and normalize to the [0,1] range.

enhance(*inverse=False*, *gamma=1.0*, *stretch='no'*)
Image enhancement function. It applies **in this order** inversion, gamma correction, and stretching to the current image, with parameters *inverse* (see `Image.invert()`), *gamma* (see `Image.gamma()`), and *stretch* (see `Image.stretch()`).

gamma(*gamma=1.0*)
Apply gamma correction to the channels of the image. If *gamma* is a tuple, then it should have as many elements as the channels of the image, and the gamma correction is applied elementwise. If *gamma* is a number, the same gamma correction is applied on every channel, if there are several channels in the image. The behaviour of `gamma()` is undefined outside the normal [0,1] range of the channels.

invert(*invert=True*)
Inverts all the channels of a image according to *invert*. If invert is a tuple or a list, elementwise invertion is performed, otherwise all channels are inverted if *invert* is true (default).

is_empty()
Checks for an empty image.

merge(*img*)
Use the provided image as background for the current *img* image, that is if the current image has missing data.

modes = ['L', 'LA', 'RGB', 'RGBA', 'YCbCr', 'YCbCrA', 'P', 'PA']

palettize(*colormap*)
Palettize the current image using *colormap*. Works only on"L" or "LA" images.

pil_image()
Return a PIL image from the current image.

pil_save(*filename*, *compression=6*, *fformat=None*)
Save the image to the given *filename* using PIL. For now, the compression level [0-9] is ignored, due to PIL's lack of support. See also `save()`.

putalpha(*alpha*)
Adds an *alpha* channel to the current image, or replaces it with *alpha* if it already exists.

replace_luminance(*luminance*)
Replace the Y channel of the image by the array *luminance*. If the image is not in YCbCr mode, it is converted automatically to and from that mode.

resize(*shape*)
Resize the image to the given *shape* tuple, in place. For zooming, nearest neighbour method is used, while for shrinking, decimation is used. Therefore, *shape* must be a multiple or a divisor of the image shape.

save(*filename*, *compression=6*, *fformat=None*)
Save the image to the given *filename*. For some formats like jpg and png, the work is delegated to `pil_save()`, which doesn't support the *compression* option.

show()
Display the image on screen.

stretch(*stretch='crude'*, ***kwarg*)
Apply stretching to the current image. The value of *stretch* sets the type of stretching applied. The values "histogram", "linear", "crude" (or "crude-stretch") perform respectively histogram equalization, contrast stretching (with 5% cutoff on both sides), and contrast stretching without cutoff. The value "logarithmic" or "log" will do a logarithmic enhancement towards white. If a tuple or a list of two values is given as input, then a contrast stretching is performed with the values as cutoff. These values should be normalized in the range [0.0,1.0].

stretch_hist_equalize(*ch_nb*)
Stretch the current image's colors by performing histogram equalization on channel *ch_nb*.

stretch_linear(*ch_nb*, *cutoffs=(0.005*, *0.005)*)
Stretch linearly the contrast of the current image on channel *ch_nb*, using *cutoffs* for left and right trimming.

stretch_logarithmic(*ch_nb*, *factor=100.0*)
Move data into range [1:factor] and do a normalized logarithmic enhancement.

exception trollimage.image.**UnknownImageFormat**
Exception to be raised when image format is unknown to pytroll-image

trollimage.image.**check_image_format**(*fformat*)
Check that *fformat* is valid

trollimage.image.**ensure_dir**(*filename*)
Checks if the dir of f exists, otherwise create it.

trollimage.image.**rgb2ycbcr**(*r__*, *g__*, *b__*)
Convert the three RGB channels to YCbCr.

trollimage.image.**ycbcr2rgb**(*y__*, *cb_*, *cr_*)
Convert the three YCbCr channels to RGB channels.

CHAPTER 2

Colorspaces

Color spaces using numpy to run on arrays.

trollimage.colorspaces.**hcl2lab**(*h__*, *c__*, *l__*)
HCL to L*ab

trollimage.colorspaces.**hcl2rgb**(*h__*, *c__*, *l__*)
HCL to RGB

trollimage.colorspaces.**lab2hcl**(*l__*, *a__*, *b__*)
L*a*b* to HCL

trollimage.colorspaces.**lab2rgb**(*h__*, *c__*, *l__*)
L*ab to RGB

trollimage.colorspaces.**lab2xyz**(*l__*, *a__*, *b__*)
Convert L*a*b* to XYZ, L*a*b* expressed within [0, 1].

trollimage.colorspaces.**rgb2hcl**(*r__*, *g__*, *b__*)
RGB to HCL

trollimage.colorspaces.**rgb2lab**(*r__*, *g__*, *b__*)
RGB to L*ab

trollimage.colorspaces.**rgb2xyz**(*r__*, *g__*, *b__*)
RGB to XYZ

trollimage.colorspaces.**xyz2lab**(*x__*, *y__*, *z__*)
Convert XYZ to L*a*b*.

trollimage.colorspaces.**xyz2rgb**(*x__*, *y__*, *z__*)
XYZ colorspace to RGB

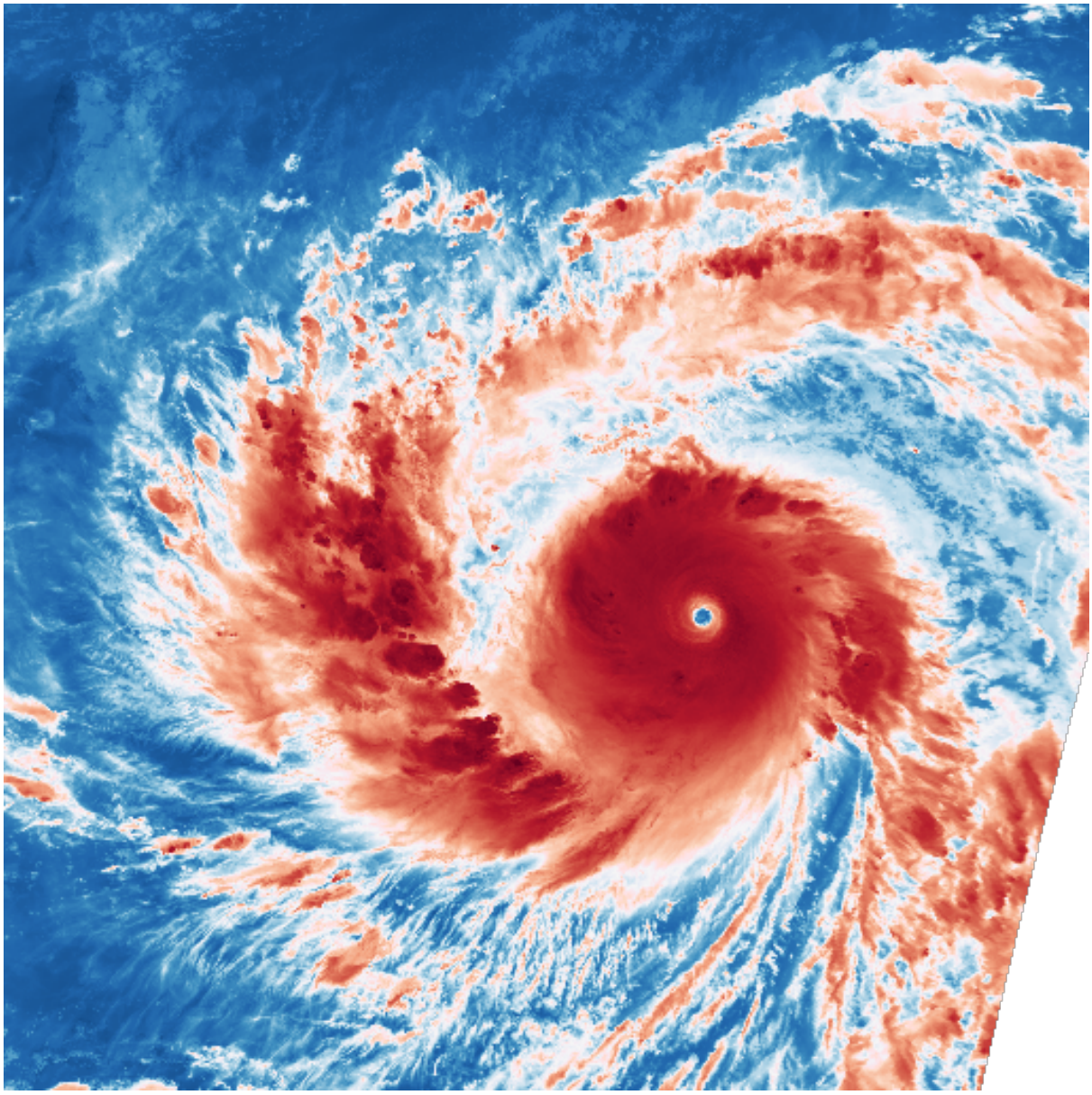
CHAPTER 3

Colormap

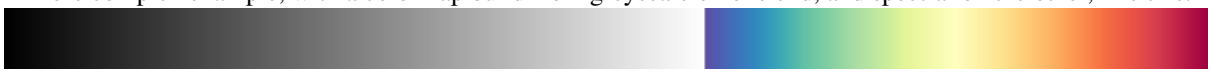
3.1 Example usage

A simple example of applying a colormap on data:

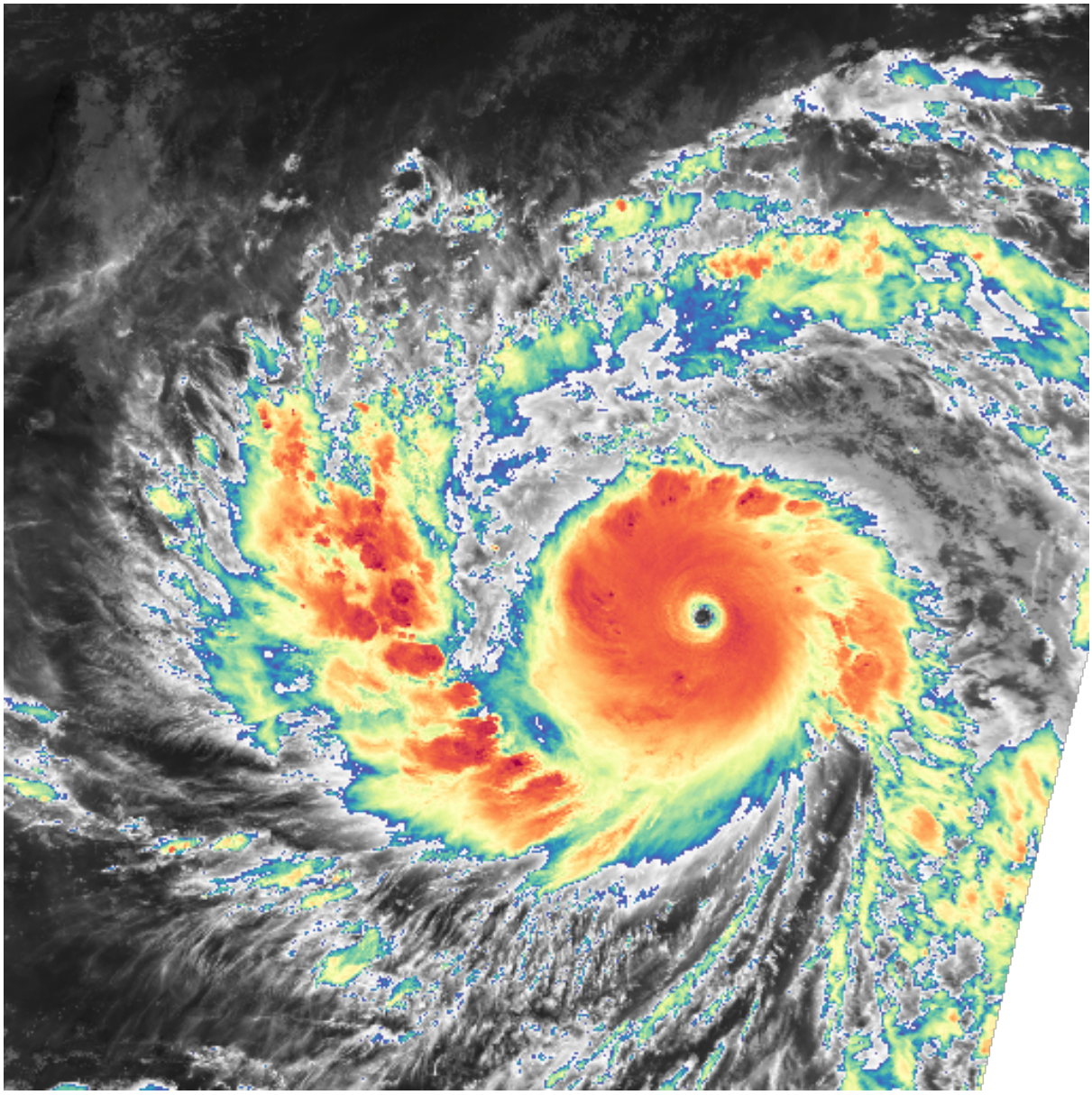
```
from trollimage.colormap import rdbu
from trollimage.image import Image

img = Image(data, mode="L")

rdbu.set_range(-90 + 273.15, 30 + 273.15)
img.colorize(rdbu)

img.show()
```



A more complex example, with a colormap build from greyscale on one end, and spectral on the other, like this:

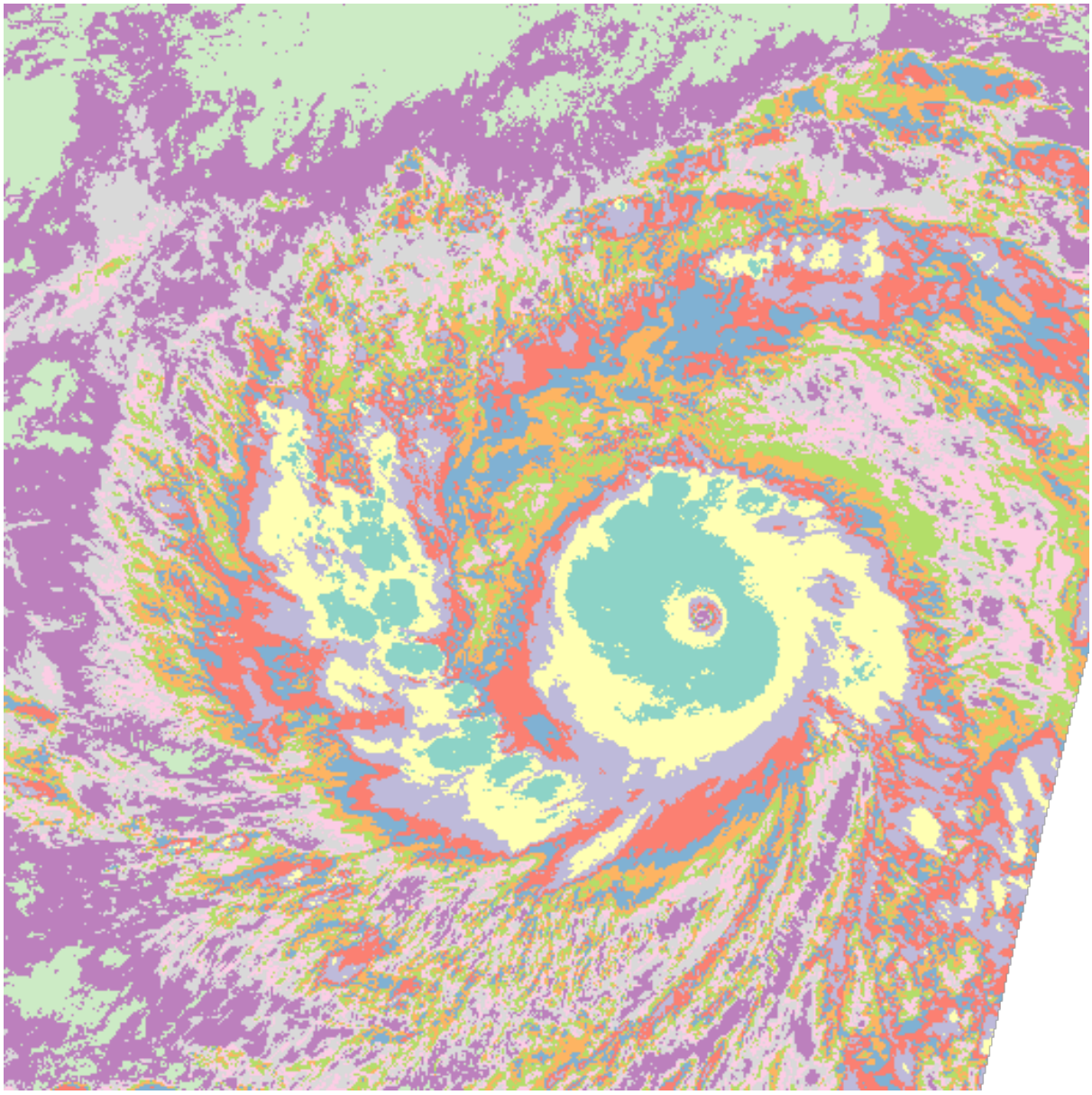


```
from trollimage.colormap import spectral, greys
from trollimage.image import Image

img = Image(data, mode="L")

greys.set_range(-40 + 273.15, 30 + 273.15)
spectral.set_range(-90 + 273.15, -40.00001 + 273.15)
my_cm = spectral + greys
img.colorize(my_cm)

img.show()
```



Now applying a palette to the data, with sharp edges:

```
from trollimage.colormap import set3
from trollimage.image import Image

img = Image(data, mode="L")

set3.set_range(-90 + 273.15, 30 + 273.15)
img.palettize(set3)

img.show()
```



3.2 API

A simple colormap module.

class `trollimage.colormap.`**`Colormap`**(**tuples*)

The colormap object.

Initialize with tuples of (value, (colors)), like this:

```
Colormap((-75.0, (1.0, 1.0, 0.0)),
         (-40.0001, (0.0, 1.0, 1.0)),
         (-40.0, (1, 1, 1)),
         (30.0, (0, 0, 0)))
```

You can also concatenate colormaps together, try:

```
cm = cm1 + cm2
```

`colorize`(*data*)

Colorize a monochromatic array *data*, based on the current colormap.

palettize(*data*)
Palettize a monochromatic array *data* based on the current colormap.

reverse()
Reverse the current colormap in place.

set_range(*min_val*, *max_val*)
Set the range of the colormap to [*min_val*, *max_val*]

trollimage.colormap.**colorbar**(*height*, *length*, *colormap*)
Return the channels of a colorbar.

trollimage.colormap.**colorize**(*arr*, *colors*, *values*)
Colorize a monochromatic array *arr*, based *colors* given for *values*. Interpolation is used. *values* must be in ascending order.

trollimage.colormap.**palettebar**(*height*, *length*, *colormap*)
Return the channels of a palettebar.

trollimage.colormap.**palettize**(*arr*, *colors*, *values*)
From start *values* apply *colors* to *data*.

3.3 Default Colormaps

Colors from www.ColorBrewer.org by Cynthia A. Brewer, Geography, Pennsylvania State University.

3.3.1 Sequential Colormaps

blues

greens

greys

oranges

purples

reds

bugn

bupu

gnbu

orrd

pubu

pubugn

purd

rdpu

ylgn

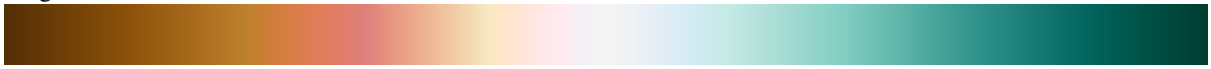
ylgnbu

ylorbr

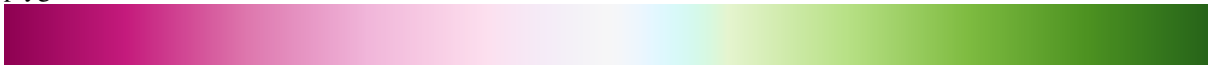
ylorrd

3.3.2 Diverging Colormaps

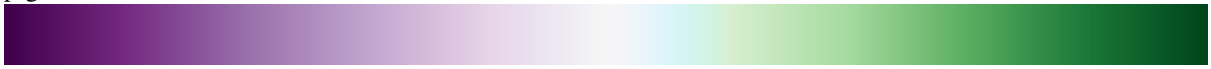
brbg



piyg

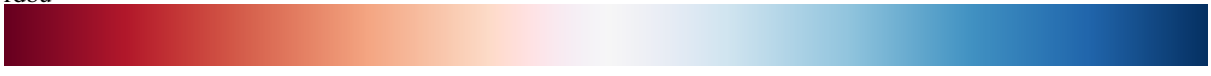


prgn



puor

rdbu



rdgy

rdylbu

rdylgn

spectral

3.3.3 Qualitative Colormaps

set1

set2

set3

paired

accent

dark2

pastel1

pastel2

3.3.4 Rainbow Colormap

Don't use this one ! See here why

rainbow

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

t