Satellite Imagery

Getting Started with Python

Outline

- Data Formats : text, binary, self-descriptive
- Reading Text Data
- Creating RGB and grey scale images

Basic Data Formats

Text - always ASCII, arranged in rows, columns, headers.
NEVER word processing format or excel. '.csv' is text.
PRO – easy for humans to read and understand.
CON – slow for computer to read, uses more storage than any other format. Easy to corrupt during viewing.
USE – small files. More convenient for humans.

Binary - single vector with each field of a known size.
Will need to rearrange into a grid after reading.
PRO – easy and fast for computer to read. Very compact.
CON – humans can only read through writing an interpreter.
Requires extra information into order to read.
USE – large files. Replaced by self-describing files for satellite.

Self Descriptive Formats

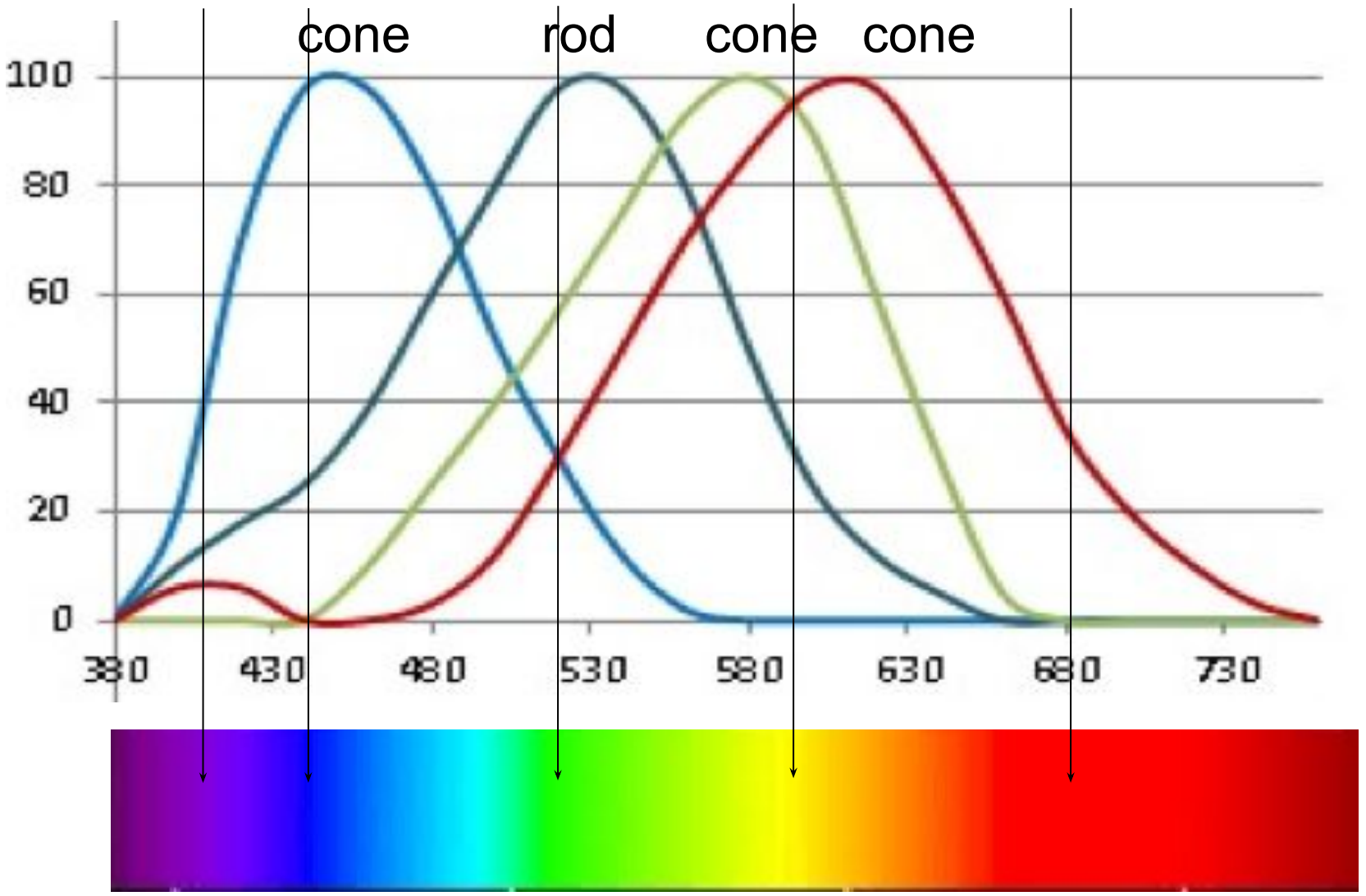
The data is encoded as binary for speed and compactness, but has descriptors embedded in the file so that special "readers" (packages of computer code) can pull out the "metadata" to format and display the data.

geoTIFF - a TIFF image with extra geographical and data set descriptors embedded. No reader needed to see image.
USE – primarily LandSat and other USGS products.

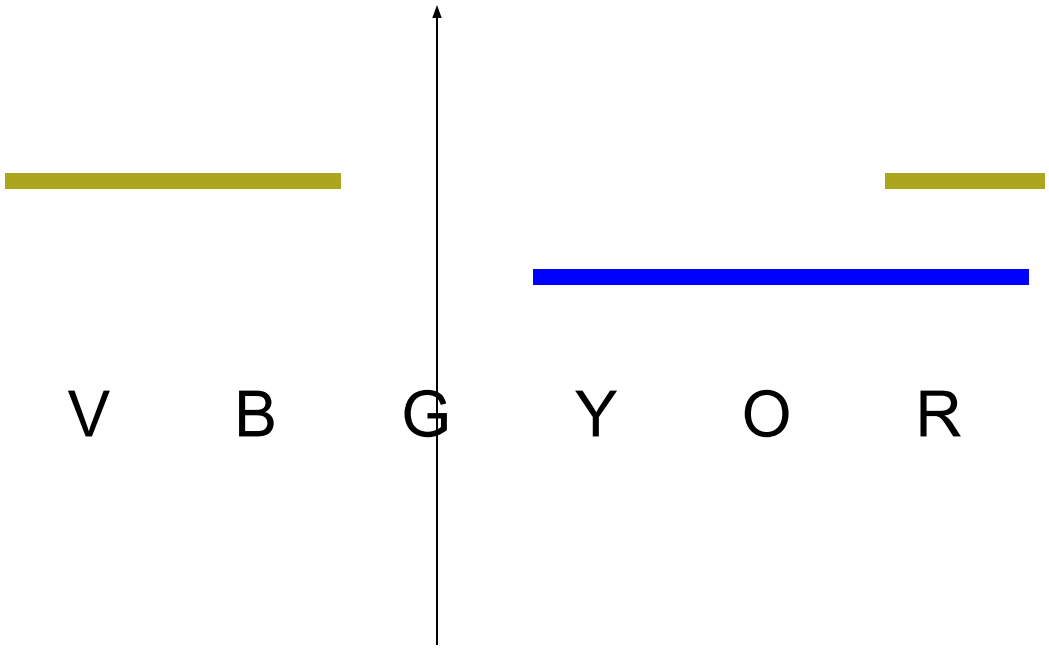
netCDF - "network Common Data Format" created at the request of NASA (which later switched mainly to HDF)
USE – the majority of NOAA satellite products and some by NASA, which is now providing multiple formats

HDF - "Hierarchical Data Format" more flexible but more complex than netCDF.
USE – mainly by NASA-EOS, but the new HDF5 format incorporates netCDF4, and is becoming more standard.

Rods, Cones, (and purple)



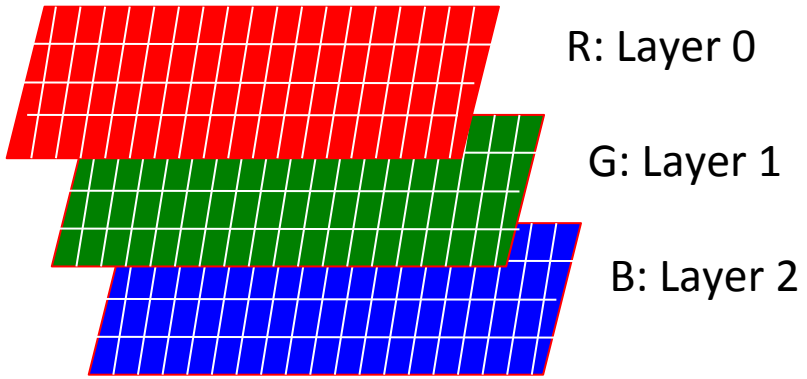
Subtractive Colors (paint)



Making Color Images in Python

Image_array = (rows, cols, 3 layers)

Image_array =



The values in each layer must range from 0 to 1, or they will be “railed off”. You may need to adjust your data accordingly.

Color Programming Exercises

```
>>>  import numpy as np   # well duh     >>> plt.on()
```

Make a zero-filled array called RGB with 300 rows, 300 columns, 3 layers.

```
>>> RGB =  np.zeros([300,300,3],dtype=float)
```

Display this image: what color should it be?

```
>>> plt.imshow(RGB)     >>> plt.show()
```

Turn the image Red.

```
>>> RGB[ : , : , 0] = 1          >>> plt.imshow(RGB)
```

Turn the image White.

```
>>> RGB( : , : , : ) = 1          >>> plt.imshow(RGB)
```

Turn the image Grey.

```
>>> RGB( : , : , : ) = 0.5          >>> plt.imshow(RGB)
```

Color Exercises Continued

Turn the center third of the image Green.

```
>>> RGB[ 100:200 , 100:200 , 1 ] = 1          >>> plt.imshow(RGB)
```

Turn the center third of the image Yellow.

```
>>> RGB[ 100:200 , 100:200 , 0 ] = 1          >>> plt.imshow(RGB)
```

Make the upper left corner Blue and lower right corner Red.

```
>>> RGB[ 0:100 , 0:100 , 2 ] = 1
>>> RGB[200:300,200:300,0 ] = 1
>>> plt.imshow(RGB)
>>> plt.savefig( 'myfigure.png')   # saves figure
```

Reading our first Satellite Image

- To make it easier to see what is going on, we are using a prepared text file called landsat_RGBN.txt
- The layers are red, green, blue, near-infrared
- Each row is a row of the image, then layers repeat
- We can also read the data as a single vector then reshape into an array, but with multiple layers it's easy to mix up dimensions, so I prefer to place each row of data into the proper position as we go.
- I also read row by row to make it easier to catch corrupted data.
- I have supplied read_landsat_loop in the module called landsat_module.py to do this.

Displaying our first Satellite Images

Download landsat_RGBN.txt and landsat_sample_functions.py into the same directory that python is set to. And then…

```
>>> import landsat_sample_functions as lm
>>> RGBN = lm.landsat_read('landsat_RGBN.txt')
>>> RGB = RGBN( :, :, 0:2) ;    # trim layers to visible
>>> plt.imshow(RGB)        >>> plt.show()
>>> plt.imshow(2*RGB)      # increase brightness, contrast
>>> plt.savefig('nycRGB.png')
```

How can we display vegetation amount in shades of green?

```
>>> NDVI = ( RGBN[ :, :, 3] – RGBN[ :, :, 0] )/( RGBN[ :, :, 3] + RGBN[ :, :, 0] ) ;
>>>RGB = 0*RGB;        RGB[ :, :, 1] = NDVI
>>> plt.imshow(RGB)    plt.colorbar()    >>> plt.show()
```

Exercise 1

Write a function that will display a map of NDVI in shades of green. It must have comments that explain the input and output as well as any internal calculations or variables. It must work when tested.

INPUT VARIABLES: Red intensities, Near Infrared Intensities.
Both in array format.
OUTPUT VARIABLES: NDVI in array format
OUTPUT PRODUCTS: map of NDVI, saved as .png file
~~map of RGB, saved as .png file~~

Exercise 2

- Use landsat_read to read the thermal radiances found in landsat_thermrad.txt.
- Use your BT function to convert these into temperatures
- Display this data with a colorbar to show the temperatures