Satellite Imagery

Enough to get started with Octave

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

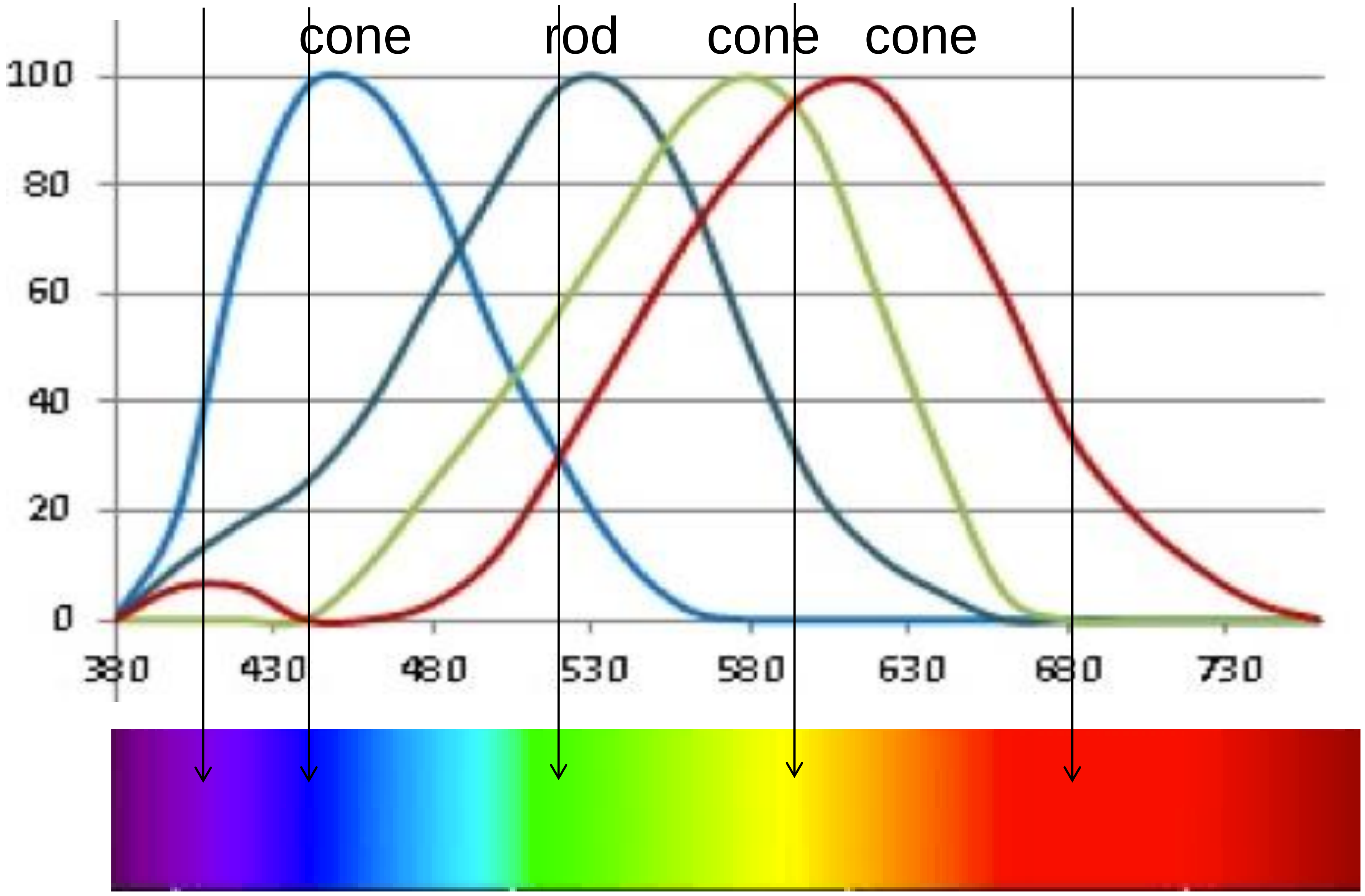
The data is encoded as binary for speed and compactness, but has descriptors embedded in the file so that special "readers" (packages of computer code) can pull out the "metadata" to format and display the data.

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.

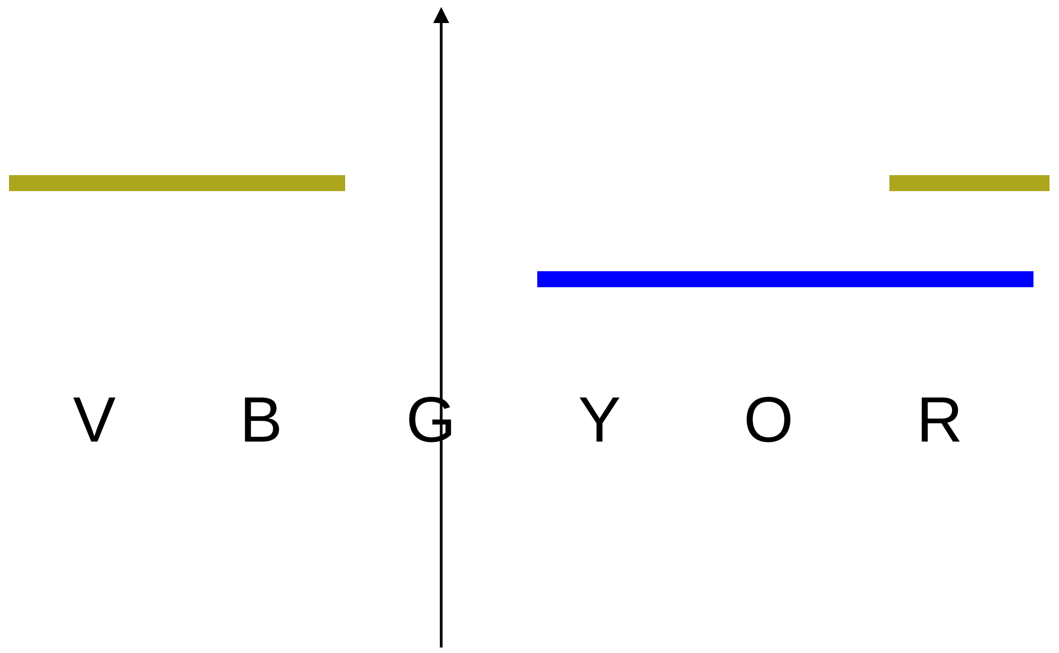
netCDF - "network Common Data Format" created at the request of NASA (which later switched mainly to HDF)
USE – the majority of NOAA satellite products and some by NASA, which is now providing multiple formats

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)



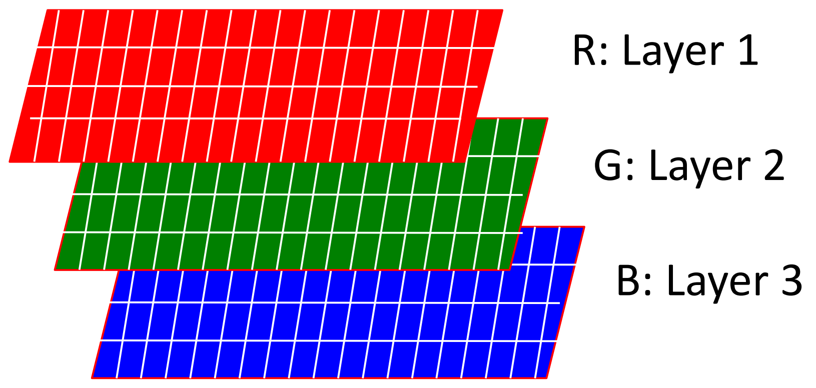
Subtractive Colors (paint)



Making Color Images in MatLab

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

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

```
>>> RGB = zeros(300,300,3);
```

Display this image: what color should it be?

```
>>> image(RGB);          black
```

Turn the image Red.

```
>>> RGB( : , : , 1 ) = 1;          >>> image(RGB);
```

Turn the image White.

```
>>> RGB( : , : , : ) = 1;          >>> image(RGB);
```

Turn the image Grey.

```
>>> RGB( : , : , : ) = 0.5;          >>> image(RGB);
```

Color Exercises Continued

Turn the center third of the image Green.

```
>>> RGB( 100:200 , 100:200 , 2 ) = 1;          >>> image(RGB);
```

Turn the center third of the image Yellow.

```
>>> RGB( 100:200 , 100:200 , 1 ) = 1;          >>> image(RGB);
```

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

```
>>> RGB( 1:100 , 1:100 , 3 ) = 1;
>>> RGB(200:300,200:300,1) = 1
>>> image(RGB);
```

Reading Text Files

There are several ways to read text files; my recommended method is to read them one line at a time, using fgets to read each line as a character string, or fscanf to read each line as a vector of numbers.

Our first step is to open the file and assign it to a simple variable.

```
>> fileID = read('landsat_RGBN.txt', 'r')
```

Now let's read the information in the header lines. Repeat 5 x

```
>> info = fgets(fileID)
```

The dimension data is worth keeping, let's read it as integers:

```
>> [dims, count] = fscanf(fileID, '%4i', 3)
```

4 digits long　　read 3 numbers

Now read the first row of data as floating point numbers:

```
>> [dims, count] = fscanf(fileID, '%f', dims(2));
```

Close the file so you can use the fileID variable again:

```
>> fclose(fileID)
```

Reading our first Satellite Image

- To see what is going on easier, 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
- If this was to be read as one vector then reformed, the layout would need to be rotated 90 degrees because matlab reads row by row but the “reform” command does column by column, which I hate.
- I also read row by row to make it easier to catch corrupted data.
- I have supplied read_landsat_loop.m to do this.

Displaying our first Satellite Images

Download landsat_RGBN.txt and read_landsat_loop.m into the same directory that Octave is set to. And then…

```
>>> RGBN = landsat_read_loop('landsat_RGBN.txt',5) ;   ← Don't forget!
>>> RGB = RGBN( :, :, 1:3) ;    # trim layers to visible
>>> image(RGB)
>>> image(2*RGB)      # increase brightness, contrast
```

How can we display vegetation amount in shades of green?

```
>>> NDVI = ( RGBN( :, :, 4) – RGBN( :, :, 1) )./( RGBN( :, :, 4) + RGBN( :, :, 1) ) ;
>>>RGB = 0*RGB;       RGB( :, :, 2) = NDVI ;
>>>  imagesc(NDVI)  vs  image(RGB)
```

The dot matters!

Exercise 1 (CWK 2)

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 (CWK 3)

- Use landsat_read_loop to read the thermal radiances found in landsat_thermrad.txt.
- Use your BT function to convert these into temperatures
- Use imagesc to make a scaled image of these temperatures
- Rescale the temperatures so that the data goes from a minimum of 0 to a maximum of 1
- Display this rescaled data as tones of red using image

Using Self-Describing Files

netCDF4 and HDF5 files can be read by Octave by using the 'load' function operating on the filename.

```
>>> dataset_name = load('filename')
```

The output dataset_name will be structure. To find out what's in the structure use the fieldnames command:

```
>> fieldnames(dataset_name)
 field1
field2
....
```

Dot is typical notation for structures

Get the information by >>> dataset_name.field1

Sample Data Files

http://hdfeos.org/zoo/index.php

Many of the .nc files will work, but only if they are netcdf4. the only way to tell is to try to load them.

The hdf files are not hdf5, so will not work.

You can install packages for Octave to read these other formats, but there can be a lot of dependencies requiring multiple installations to get them to work.

Our First Sample HDF5 file:

- In the table the first data center is GES DISC. To the right, click on "GPM".
- Scroll down to 'GPM More Examples'.
- Click on 1B.GPM.GMI.TB2016.20160105-S230545-E003816.010538.V05A.HDF5

Sample Data Session

The global precipitation mission has a microwave instrument with data stored as brightness temperature. The commands below will explain themselves *only when done.* Data structures are involved.

```
>>> gpm = load('1B.GPM.GMI.TB2016.20160105-S230545-
E003816.010538.V05A.HDF5')
>>> fieldnames(gpm)
>>> fieldnames(gpm.S1)
>>> size(gpm.S1.Tb)
>>> Tb1 = gpm.S1.Tb(1, :, : ) ;
>>> Tb1 = squeeze(Tb1);
>>> image(Tb1( :, 1:221))
>>> minTb1 = min(min(Tb1( :, 1:221)))
>>> maxTb1 = max(max(Tb1( :, 1:221)))
>>> imagesc(TB1(1, 1:221))
```

Georegistration

- Matching data with lat/lon coordinates is called "georegistration".
- Most files will come with latitude and longitude grids. Those that don't have a constant grid, and will provide some mathematical scheme to calculate latitude and longitude (LandSat, GOES).
- In many cases just being able to match grid points to lat/lon is good enough; but if you need to make nice maps a good deal of effort is required to learn the relevant software.
- If you can afford the loss of resolution, creating a square lat-lon grid coarse enough to include several of the original points in each one will allow very fast regridding by averaging values into each bin. Sort into bins by a simple rounding of the co-ordinates: i = round((lon – lonmin)/increment)

Making image maps the Easy Way

Start with learning how to set the axes of images:

```
>>> x = [-74, -73, -72]
>>> y = [40, 41, 42]
>>> z = [5, 8, 6; 6, 9, 8; 5, 7, 4]
>>> imagesc(x, y, z)
>>> xlabel = ‘x’
>>> ylabel = ‘y’
```

So you can set the axes to anything you want: maybe lat and lon?

Now make a blue version

```
>>> RGB = zeros(3,3,3)
>>> RGB( :, :, 3) = z/10;
>>> image(x, y, RGB)
```

So you can make a colored map with any axes you want, now add lat and lon?

Data Storage

- Integers take up less storage and are easier for computers to work with than floating point, so the raw form of most data is integers.
- To get to the physically meaningful floating point numbers, there is often a scale factor and an offset. For example, terrestrial temperatures never go below 100 K, so you can subtract that. If the resolution is 0.1 K, you can multiply by 10 and round. So
- 295.3 K -> round((295.3 – 100)*10) = 1953 and that's the number that's stored. You will need to reverse that process.
- Most readers of self described formats will take care of the scale and offset automatically, but be aware this may not always happen and you'll have to divide by the scale and add the offset yourself.

Final Thoughts on Satellite Data

- You should probably spend a full day researching your satellite products before even deciding which ones are worth downloading for your project. Then spend more time reading once you've gotten to play with the data.
- You'll often find the following terms:

- Level 1: raw radiances, minimally processed
- Level 2: products based on the radiances that are close to the original pixel layout
- Level 3: averaged by day, month, degree, etc for use in climate studies.

• Except in rare cases (such as Landsat) Level 2 products will need remapping for display or comparison to other data sets. You may be happier remapping onto a rectangular lat-lon grid by rebinning instead of using sophisticated mapping routines.