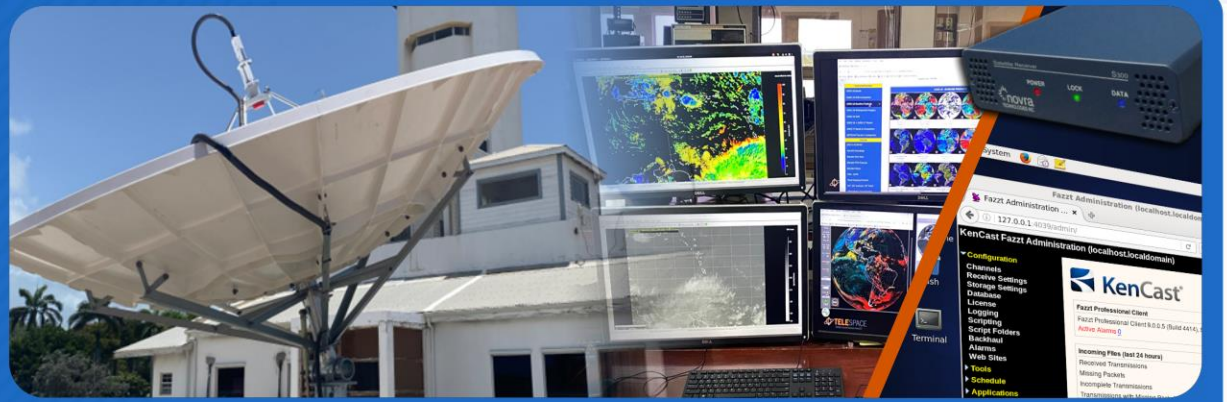
GEONETCast-Americas Training for the Eastern Caribbean States

Day 1 - May 3rd

Session 5:

Hands-on
GOES-R Data Processing



Diego Souza

diego.souza@inpe.br

DISSM - Meteorological Satellites and Sensors' Division
CGCT - General Coordination of Earth Sciences
INPE - National Institute for Space Research

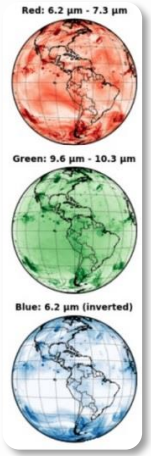
SDCR PROJECT
Strengthening Disaster and Climate Resilience in the Eastern and Southern Caribbean

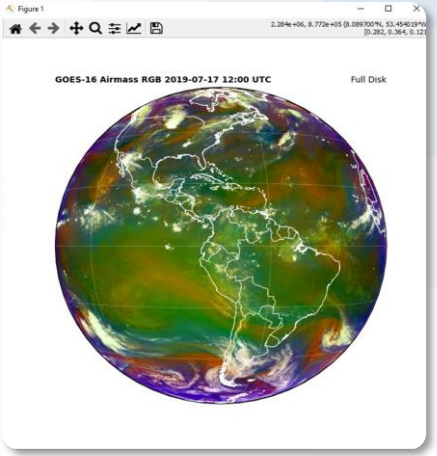
USAID FROM THE AMERICAN PEOPLE

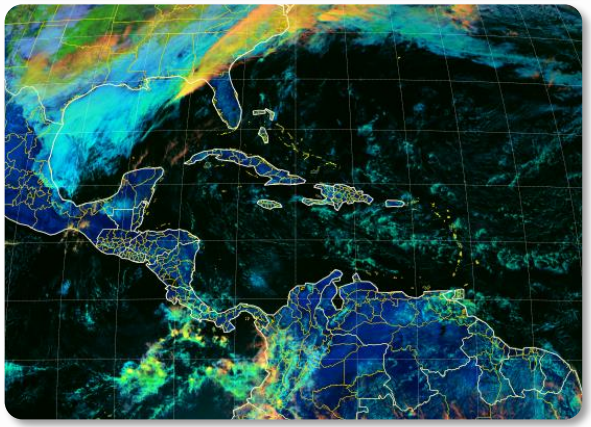
Caribbean Institute for Meteorology and Hydrology

Presentation Outline

- **Basic Concepts**
  - Basic Plot / Reading Pixel Values
  - Basic Operation / Colorbar / Title / Date
  - Overlaying Maps with Cartopy
  - Custom Colormaps
- **More Advanced Concepts**
  - Downloading Data from the Cloud
  - Cropping a Full Disk Image
  - Reprojection with GDAL
  - Derived Products







Part 1: Basic Concepts

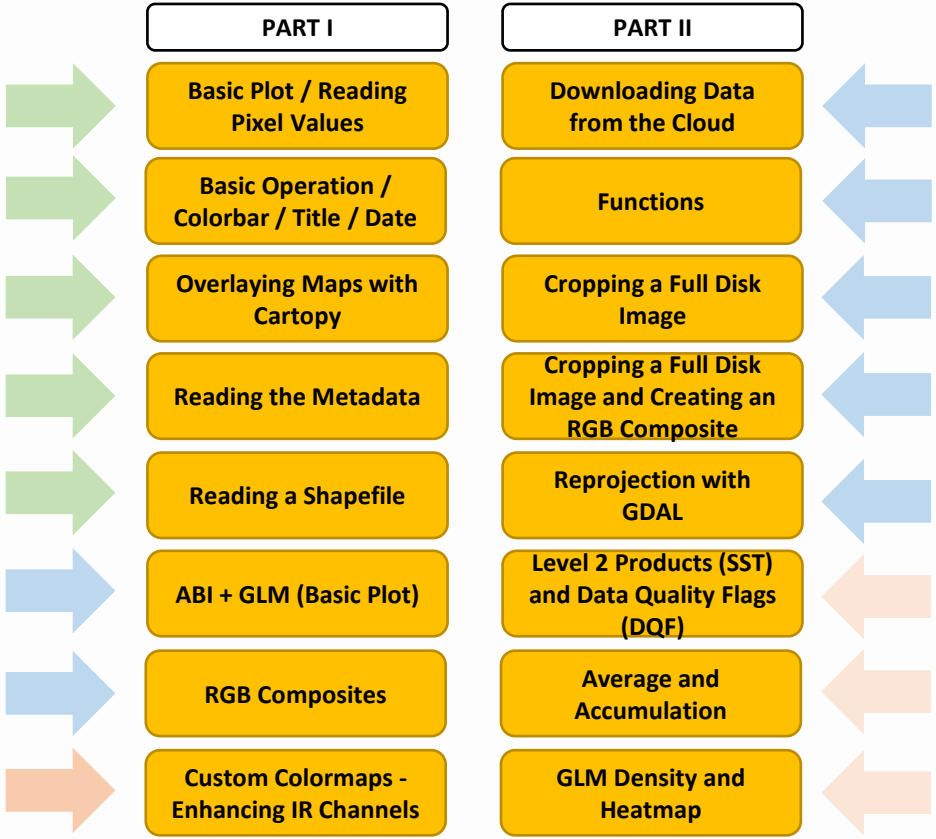
PART I	PART II
Basic Plot / Reading Pixel Values	Downloading Data from the Cloud
Basic Operation / Colorbar / Title / Date	Functions
Overlaying Maps with Cartopy	Cropping a Full Disk Image
Reading the Metadata	Cropping a Full Disk Image and Creating an RGB Composite
Reading a Shapefile	Reprojection with GDAL
ABI + GLM (Basic Plot)	Level 2 Products (SST) and Data Quality Flags (DQF)
RGB Composites	Average and Accumulation
Custom Colormaps - Enhancing IR Channels	GLM Density and Heatmap

These Will be Our Priorities

Higher priority

Medium priority

Low priority



Script 1: Basic Plot / Pixel Values

```python
# Training: Python and GOES-R Imagery: Script 1 - Basic Plot / Extracting Pixel Values
#
# Required modules
from netCDF4 import Dataset      # Read / Write NetCDF4 files
import matplotlib.pyplot as plt  # Plotting library
#
# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values
data = file.variables['CMI'][:]
#
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Plot the image
plt.imshow(data, vmin=193, vmax=313, cmap='Greys')
#
# Save the image
plt.savefig('Image_01.png')

# Show the image
plt.show()
```

LIBRARIES

DATA READING AND MANIPULATION

PLOT CONFIGURATION

IMAGE GENERATION

With each script, our “blocks of code” will become more complex.

GOES-16 Channels (ABI Sensor)

ABI Band	Central Wavelength (µm)	Wavelength Range (µm)	Resolution (km)	Spectral range	Descriptive Name
1	0.47	0.45 - 0.49	1	Visible	Blue
2	0.64	0.68 - 0.68	0.5	Visible	Red
3	0.86	0.847 - 0.882	1	Near Infrared	Vegetation
4	1.37	1.366 - 1.380	2	Near Infrared	Cirrus
5	1.6	1.59 - 1.63	1	Near Infrared	Snow/Ice
6	2.2	2.22 - 2.27	2	Near Infrared	Cloud Particle Size
7	3.9	3.80 - 3.99	2	Shortwave Infrared	Shortwave window
8	6.2	5.79 - 6.59	2	Midwave Infrared	Upper-level water vapor
9	6.9	6.72 - 7.14	2	Midwave Infrared	Midlevel water vapor
10	7.3	7.24 - 7.43	2	Midwave Infrared	Lower / midlevel watervapor
11	8.4	8.23 - 8.66	2	Longwave Infrared	Cloud-top phase
12	9.6	9.42 - 9.80	2	Longwave Infrared	Ozone
13	10.3	10.18 - 10.48	2	Longwave Infrared	Clean longwave window
14	11.2	10.82 - 11.60	2	Longwave Infrared	Longwave window
15	12.3	11.83 - 12.75	2	Longwave Infrared	Dirty longwave window
16	13.3	12.99-13.56	2	Longwave Infrared	CO2

GOES-16 Imagery Naming Convention (NetCDF)

OR_ABI-L2-CMIPF
M6C13_G16_s20191981200396 e20191981210116_c20181981210189.nc

<Operational System Real-Time Data>_<sensor>-<level>-<short product name>-M<operation mode><channel>-G<GOES>-s<start time of scan:year-julianday-hh:mm:ssUTC>/_e<end time of scan:year-julianday-hh:mm:ssUTC>/_c< data creation time ><year-julianday-hh:mm:ssUTC>/.nc

```
# Open the GOES-R image
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")
```

Script 1: Basic Plot / Pixel Values

```python
# Training: Python and GOES-R Imagery: Script 1 - Basic Plot / Extracting Pixel Values
#
# Required modules
from netCDF4 import Dataset      # Read / Write NetCDF4 files
import matplotlib.pyplot as plt  # Plotting library
#
# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values
data = file.variables['CMI'][:]
#
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Plot the image
plt.imshow(data, vmin=193, vmax=313, cmap='Greys')
#
# Save the image
plt.savefig('Image_01.png')

# Show the image
plt.show()
```

LIBRARIES

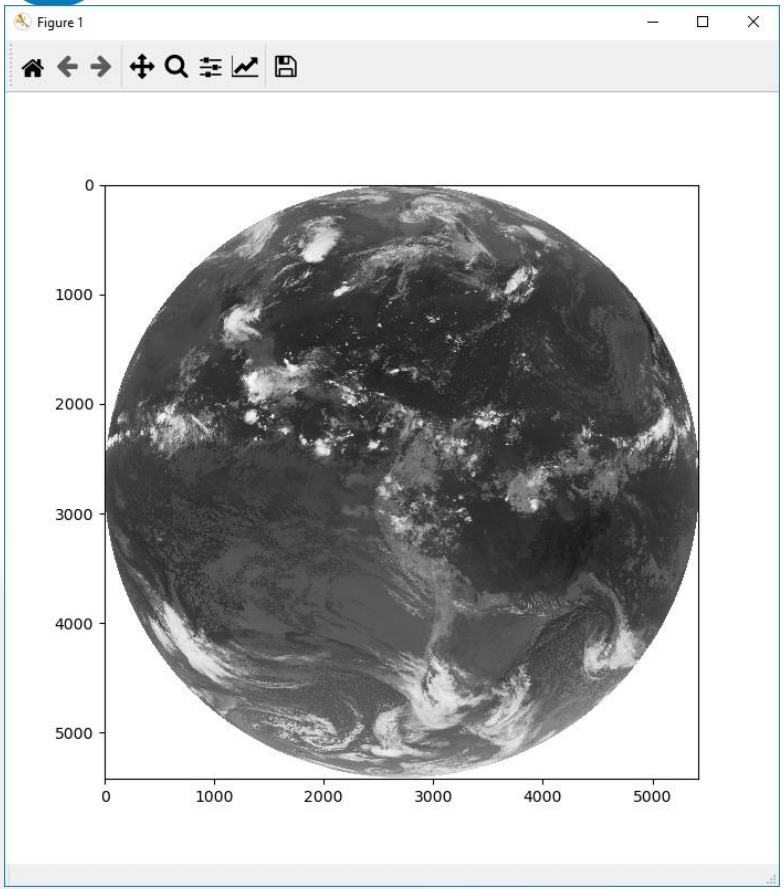
DATA READING AND MANIPULATION

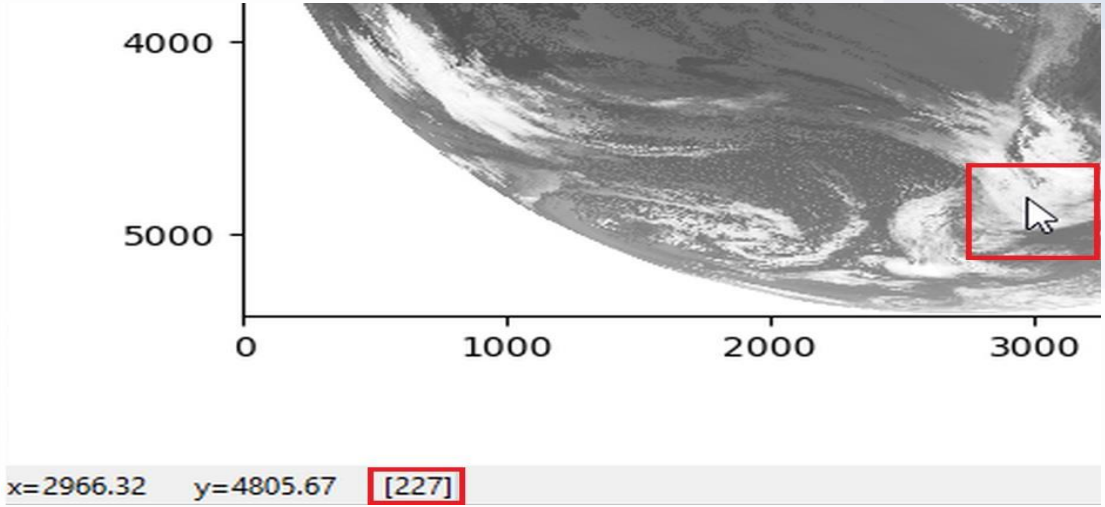
PLOT CONFIGURATION

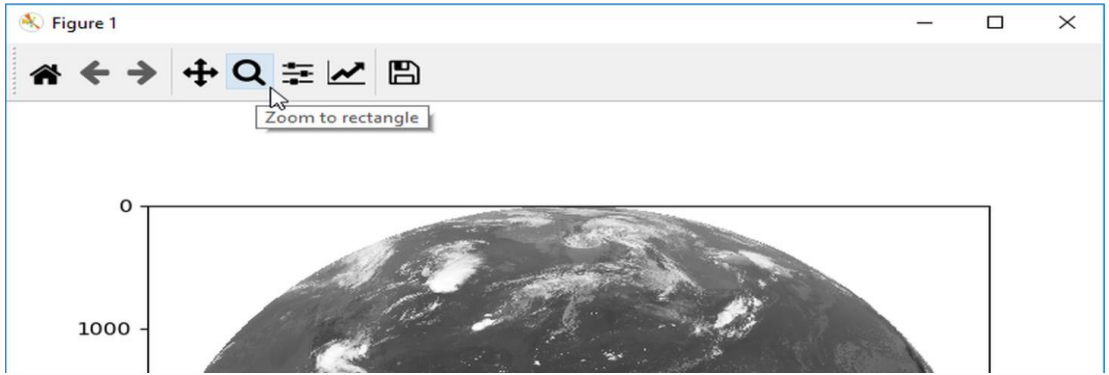
IMAGE GENERATION

With each script, our “blocks of code” will become more complex.

Script 1: Basic Plot / Pixel Values







Script 1: Basic Plot / Pixel Values

To visualize the datasets available in a NetCDF file, we may use the following command:

print(file.variables.keys())

```
#---------------------------------------------------------
# Open the GOES-R image
# Download files at this link: http://home.chpc.uta
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s2019198

# Show the available variables
print(file.variables.keys())

# Get the pixel values
data = file.variables['CMI'][:]
#---------------------------------------------------------
```

```
(workshop) D:\VLAB\Python>python Script_01.py
dict_keys(['CMI', 'DQF', 't', 'y', 'x', 'time_bounds', 'goes_imager_projection', 'y_image', 'y_image_bounds', 'x_image',
 'x_image_bounds', 'nominal_satellite_subpoint_lat', 'nominal_satellite_subpoint_lon', 'nominal_satellite_height', 'geos
patial_lat_lon_extent', 'band_wavelength', 'band_id', 'total_number_of_points', 'valid_pixel_count', 'outlier_pixel_coun
t', 'min_brightness_temperature', 'max_brightness_temperature', 'mean_brightness_temperature', 'std_dev_brightness_tempe
rature', 'planck_fk1', 'planck_fk2', 'planck_bc1', 'planck_bc2', 'algorithm_dynamic_input_data_container', 'percent_uncc
rrectable_GRB_errors', 'percent_uncorrectable_L0_errors', 'earth_sun_distance_anomaly_in_AU', 'processing_parm_version_c
ontainer', 'algorithm_product_version_container', 'esun', 'kappa0', 'focal_plane_temperature_threshold_exceeded_count',
'maximum_focal_plane_temperature', 'focal_plane_temperature_threshold_increasing', 'focal_plane_temperature_threshold_de
creasing'])
```

Some Quick Modifications

- The file to be read
- The dimensions of the final image
- The min. and max. values

Script 2: Basic Operation / Colorbar / Title / Date

```python
# Training: Python and GOES-R Imagery: Script 2 - Basic Operation / Colorbar / Title / Date

# Required modules
from netCDF4 import Dataset      # Read / Write NetCDF4 files
import matplotlib.pyplot as plt  # Plotting library
from datetime import datetime    # Basic Dates and time types

# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values
data = file.variables['CMI'][:] - 273.15

# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Plot the image
plt.imshow(data, vmin=-80, vmax=40, cmap='jet')

# Add a colorbar
plt.colorbar(label='Brightness Temperature (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(file.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 Band 13 ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Full Disk', fontsize=10, loc='right')

# Save the image
plt.savefig('Image_02.png')

# Show the image
plt.show()
```

LIBRARIES

DATA READING AND MANIPULATION

PLOT CONFIGURATION

Size relative to plot

Distance relative to plot

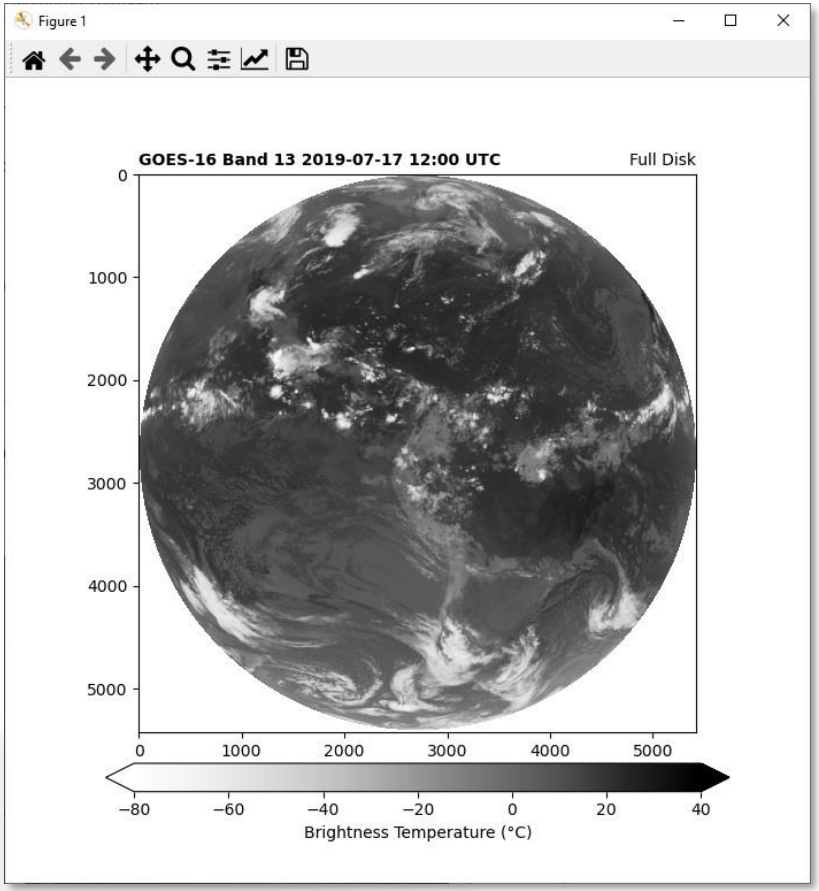
shrink

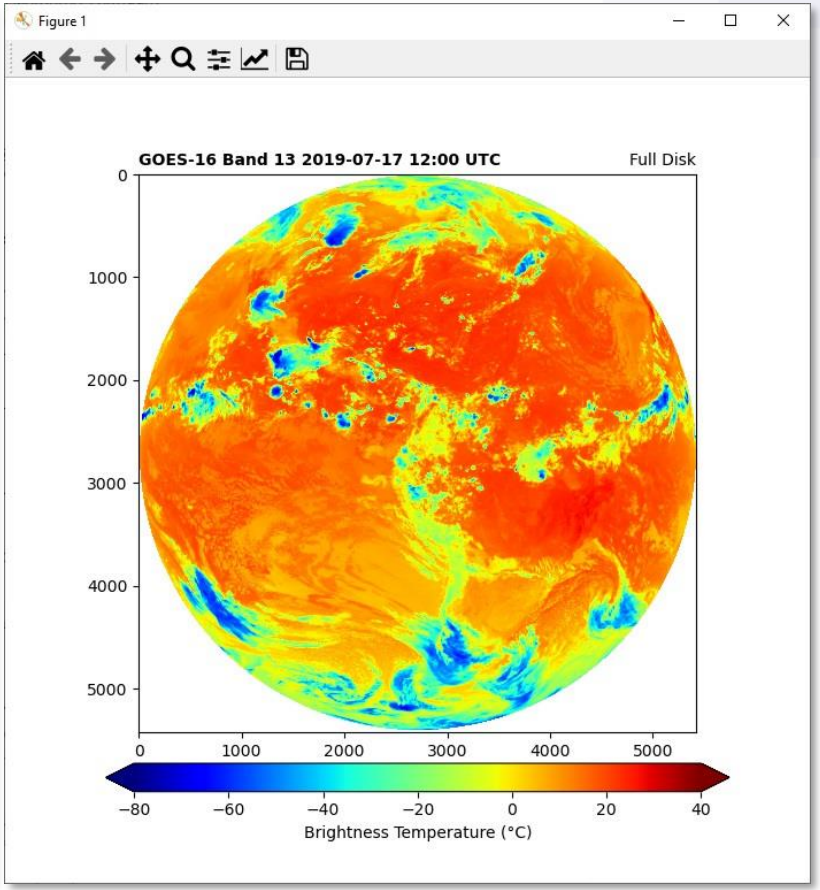
size multiplier

https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior

IMAGE GENERATION

Script 2: Basic Operation / Colorbar / Title / Date

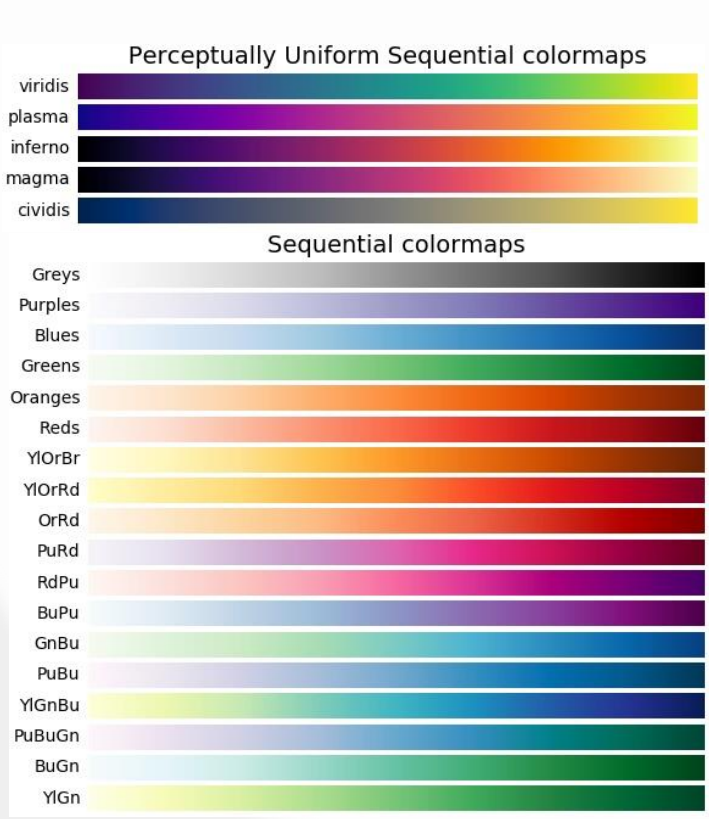


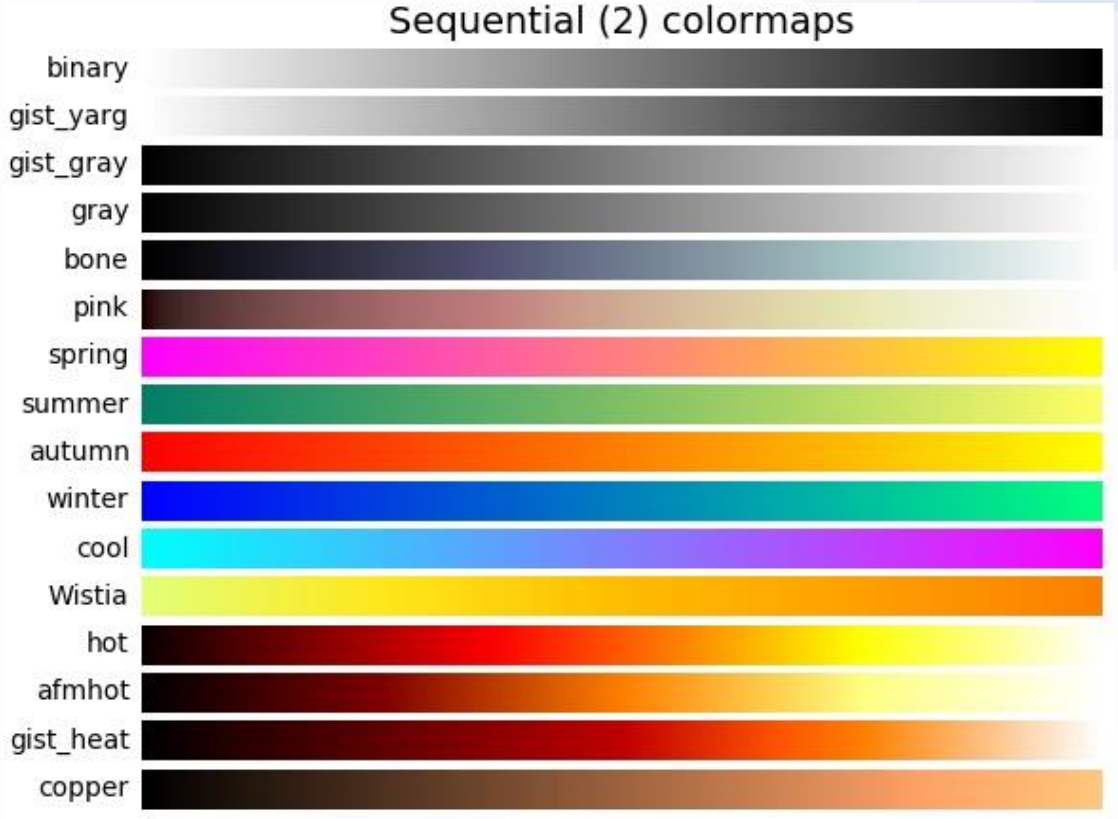


Some Quick Modifications

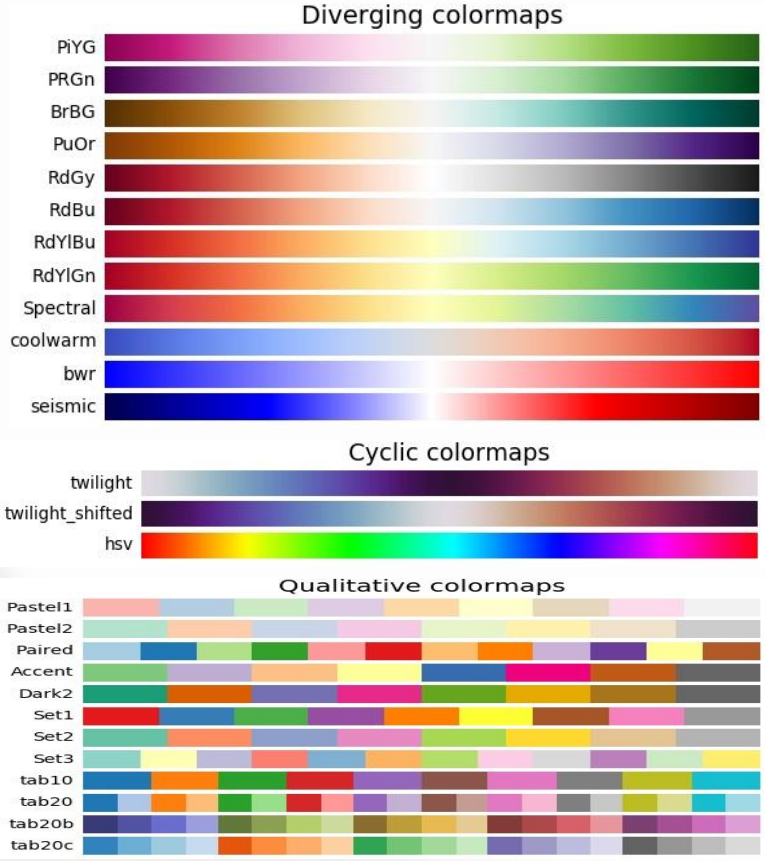
- The colorbar
- The title of the image
- The appearance of the colorbar

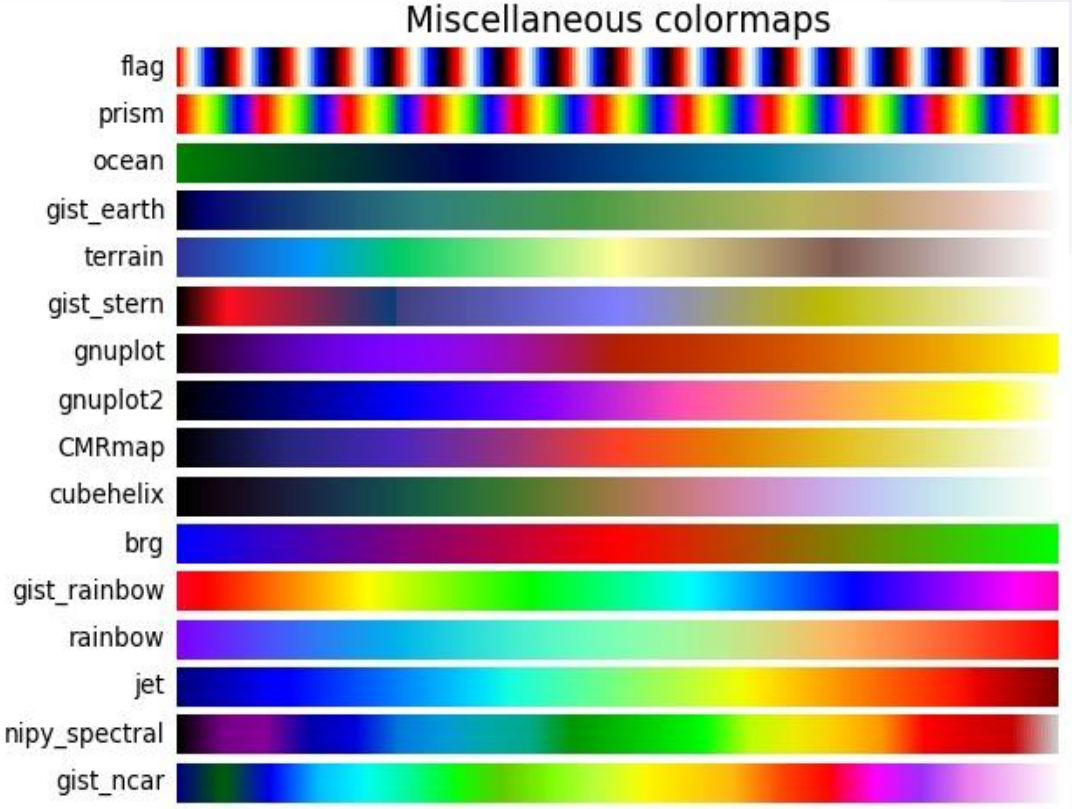
Some Standard Color Palettes





Some Standard Color Palettes

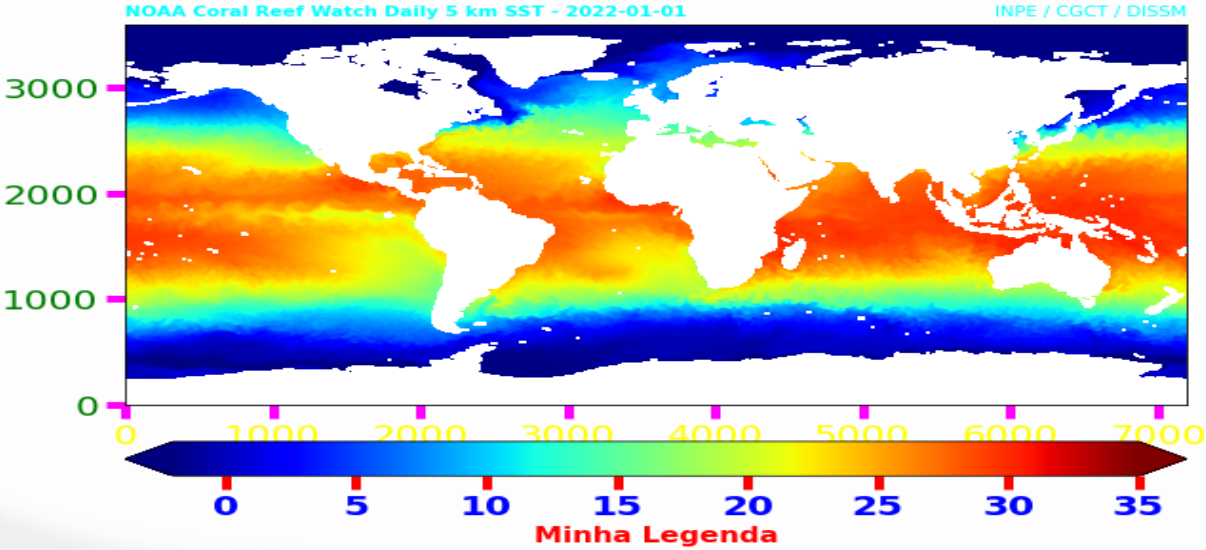




Anything Can Be Changed Via Code

The appearance of any component of the plot may be changed

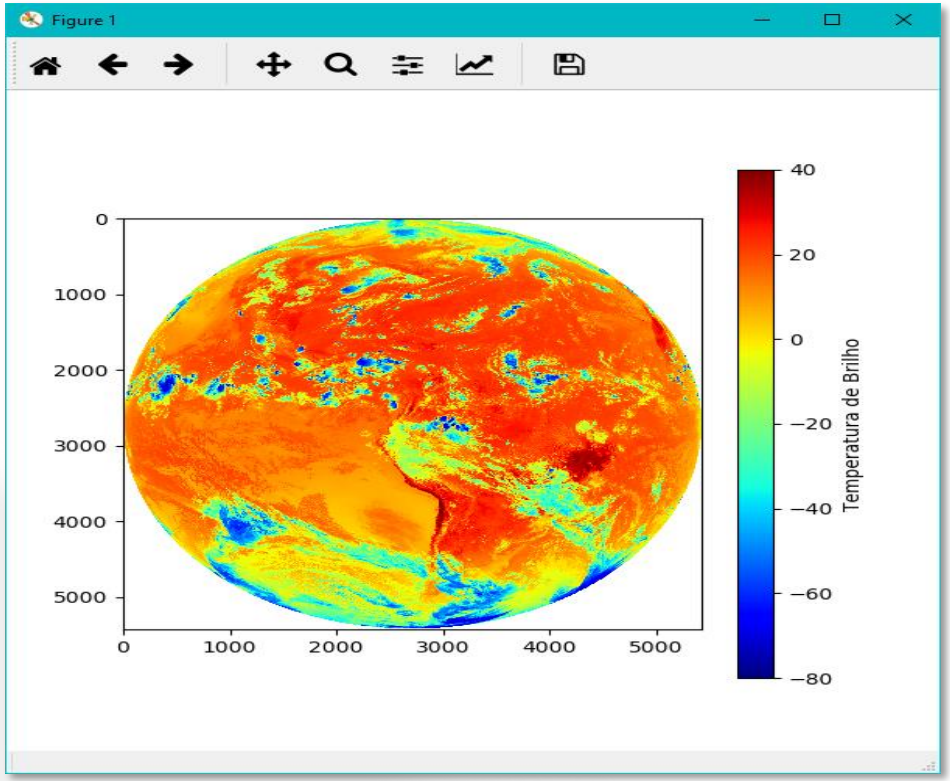
```
# Add a colorbar
cb = plt.colorbar(extend='both', orientation='horizontal', pad=0.05, fraction=0.05)
cb.set_label(label='Minha Legenda', color='red', size=15, weight='bold')
cb.ax.tick_params(color="red", width=5, length=10)
plt.tick_params(color='magenta', width=5, length=10)
plt.setp(plt.getp(cb.ax.axes, 'xticklabels'), color='blue', weight='bold')
plt.rcParams.update({'font.size':20, 'axes.labelcolor':'cyan', 'xtick.color':'yellow', 'ytick.color':'green'})
```

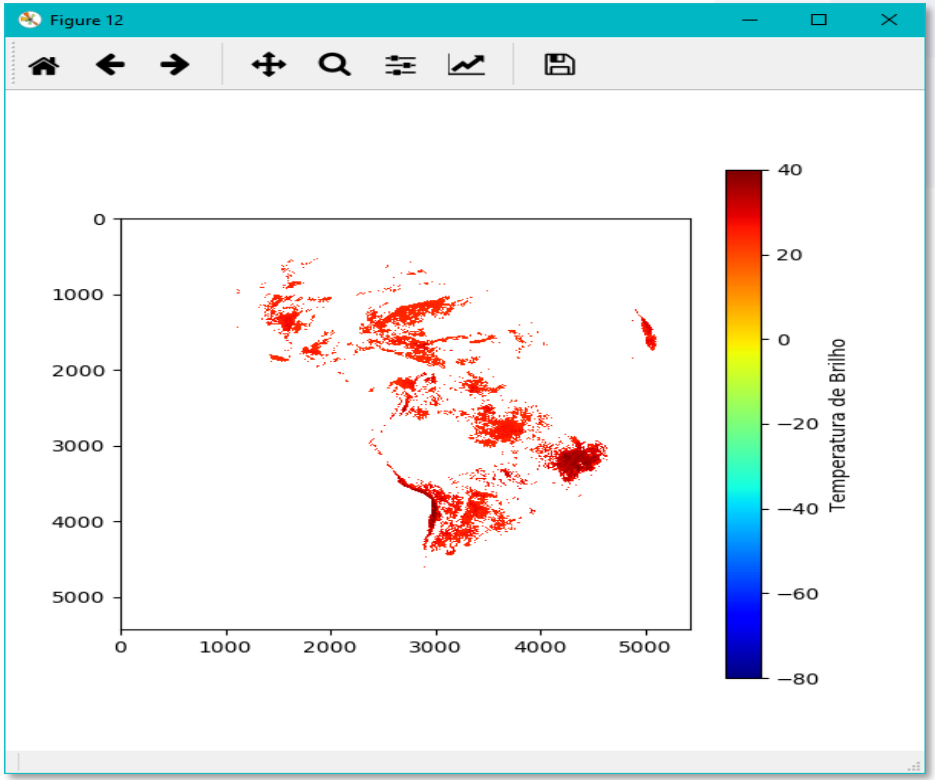


Script 2 (Extra): Visualizing a Specific Range

To not consider a range of values in the plot, use the following command:

```
import numpy as np
data[data < 23] = np.nan
```





Script 2 (Extra): Visualizing a Specific Range

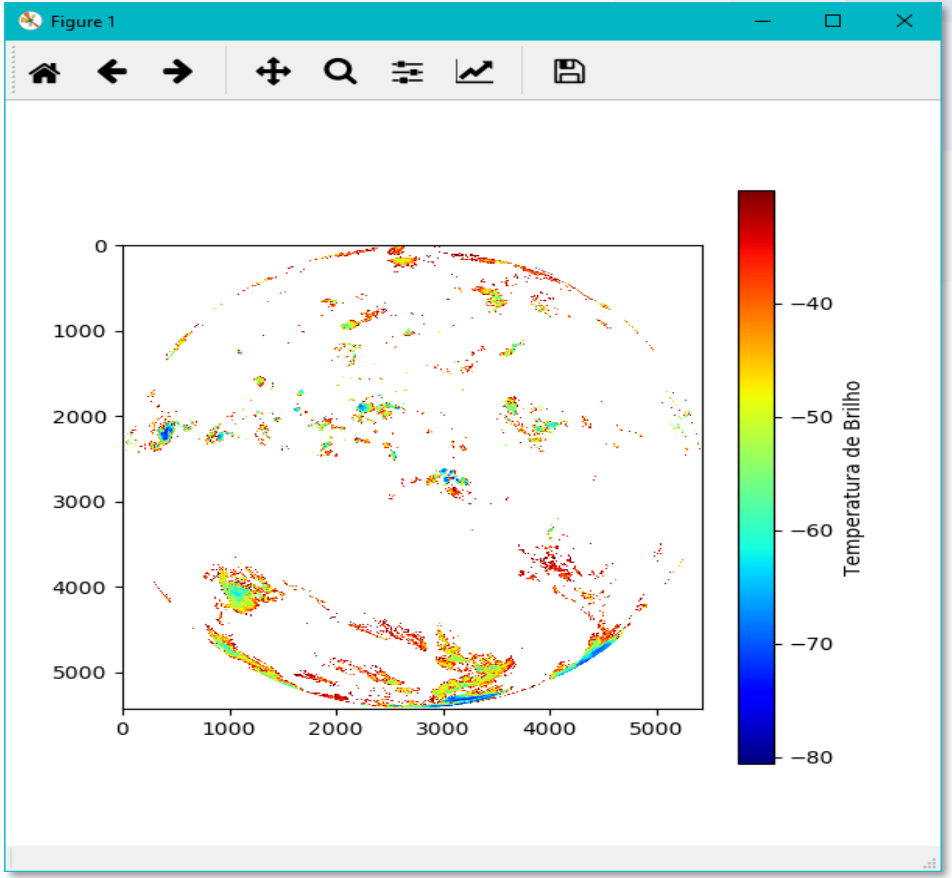
Create a plot of the 10.3 µm channel, considering only the coldest cloud tops.

Above what Brightness Temperature would you visualize the data?

```
data[data > ??????] = np.nan
```

Script 2 (Extra): Visualizing a Specific Range

```
import numpy as np
data[data > -30] = np.nan
```



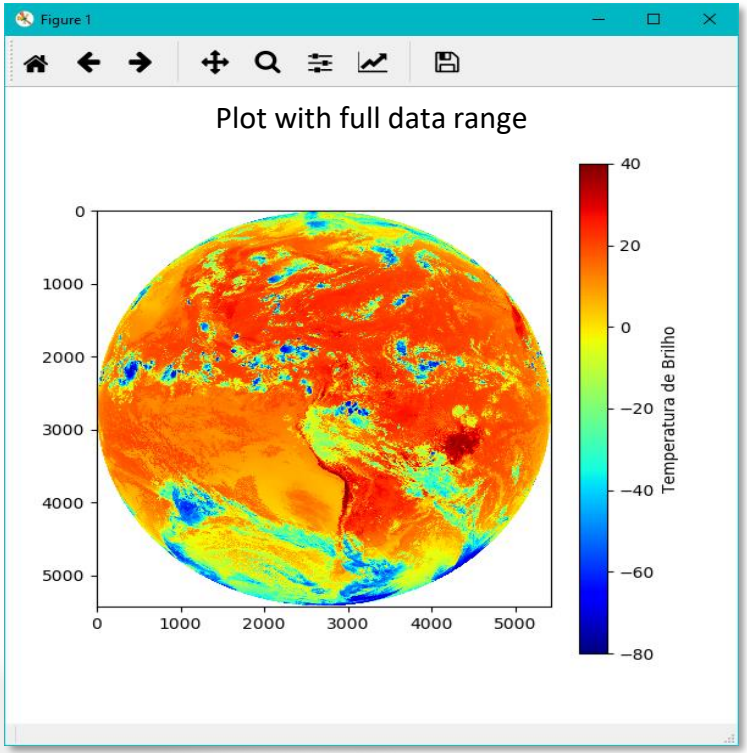
Script 2 (Extra): Visualizing a Specific Range

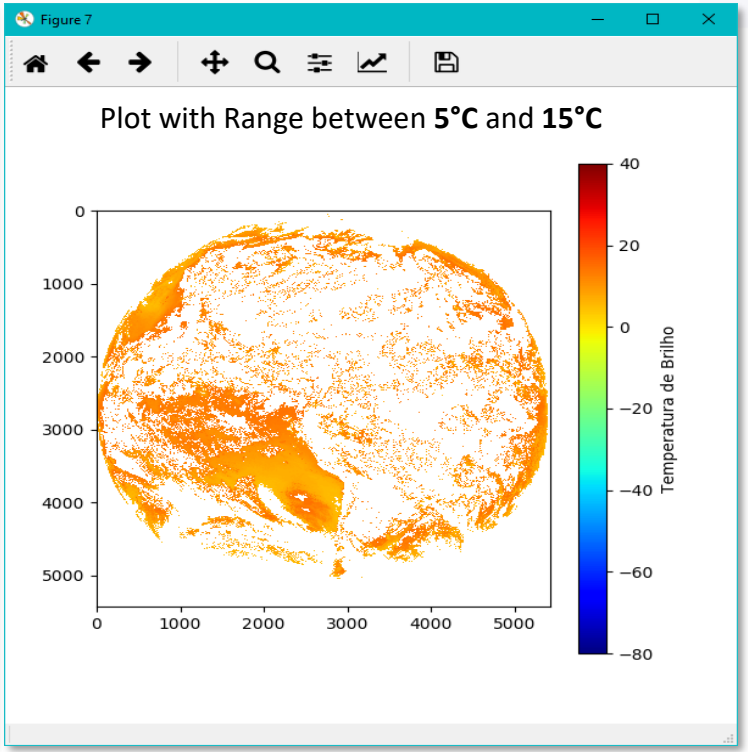
Let's create a plot considering only the pixels between two values : 5°C and 15°C

```
import numpy as np
data[np.logical_or(data < 5, data > 15)] = np.nan
```

Script 2 (Extra): Visualizing a Specific Range

```
import numpy as np
data[np.logical_or(data < 5, data > 15)] = np.nan
```



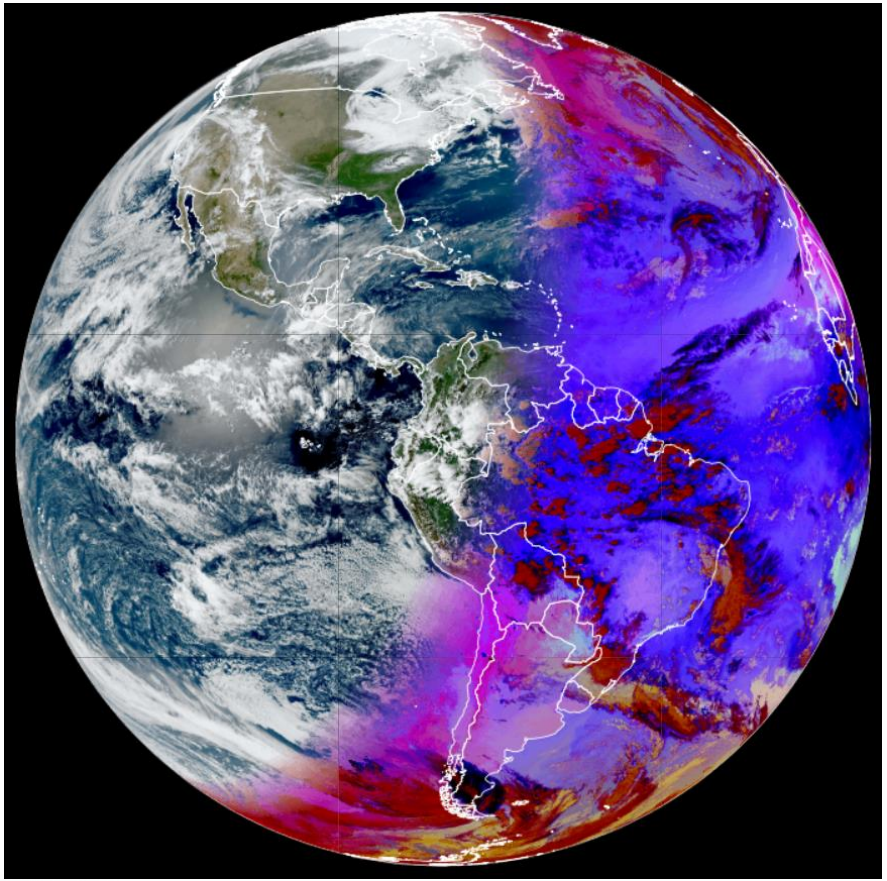


Script 2 (Extra): Why this is Useful - An Example

Let's suppose I want to plot an RGB during the day and another RGB during the night and overlay both

When I calculate the RGB for the day, I need to set all pixels in the night regions as NaN

When I calculate the RGB for the night, I need to set all pixels in the day regions as NaN



Then, when I overlay both RGBs, I have a nice composite as this one from DSAT.

The day and night regions are detected by calculating the solar zenith angle with the PyOrbital library

Script 3: Overlaying Maps with Cartopy

```python
# Training: Python and GOES-R Imagery: Script 3 - Adding a Map with Cartopy
#
# Required modules
from netCDF4 import Dataset          # Read / Write NetCDF4 files
import matplotlib.pyplot as plt      # Plotting library
from datetime import datetime        # Basic Dates and time types
import cartopy, cartopy.crs as ccrs  # Plot maps
#-------------------------------------------------------------------------------
# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values
data = file.variables['CMI'][:] - 273.15

# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.Geostationary(central_longitude=-75.0, satellite_height=35786023.0))
img_extent = (-5434894.67527,5434894.67527,-5434894.67527,5434894.67527)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='white', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.5)
ax.gridlines(color='white', alpha=0.5, linestyle='--', linewidth=0.5)

# Plot the image
img = ax.imshow(data, vmin=-80, vmax=40, origin='upper', extent=img_extent, cmap='Greys')

# Add a colorbar
plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(file.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 Band 13 ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Full Disk', fontsize=10, loc='right')

# Save the image
plt.savefig('Image_03.png')

# Show the image
plt.show()
```

LIBRARIES

DATA READING AND MANIPULATION

Central Longitude

Satellite Altitude

Data extension

Coastlines

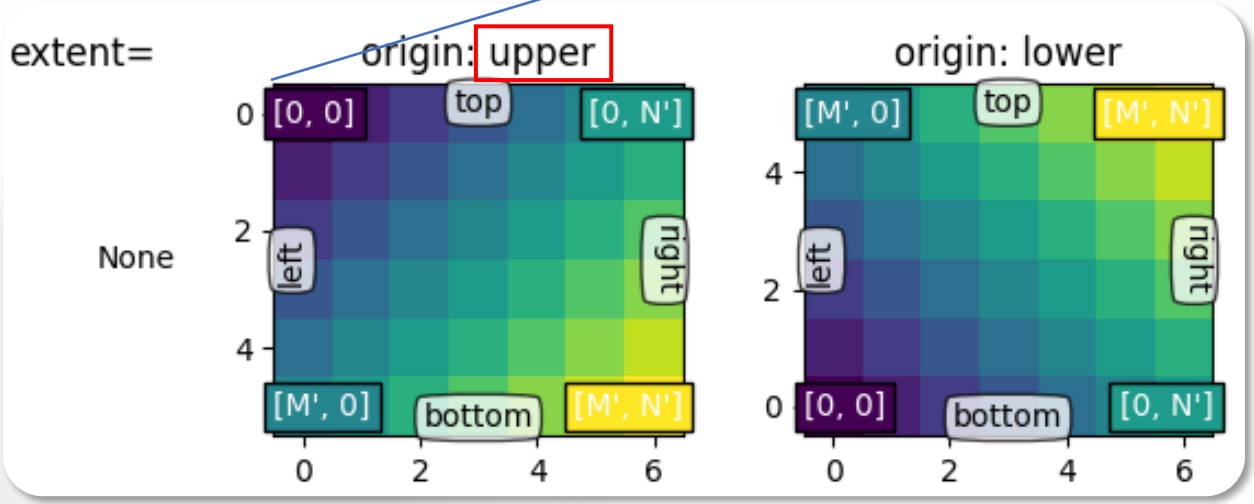
Countries

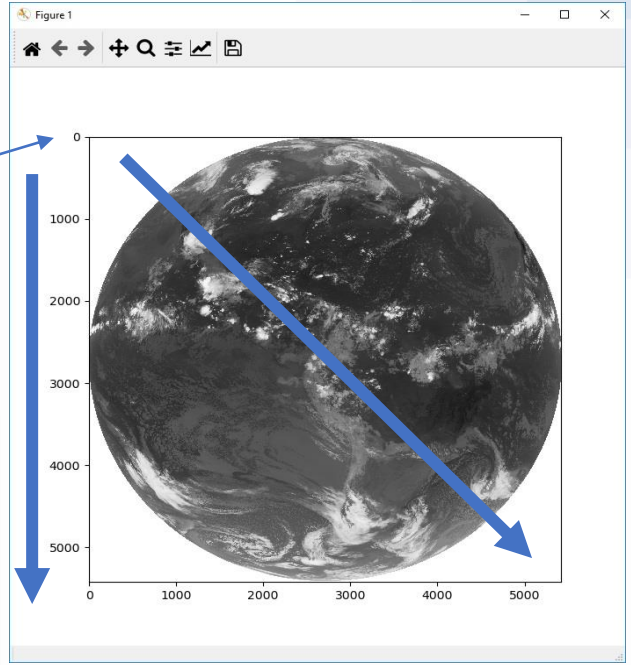
PLOT CONFIGURATION

origin of the array [0,0]

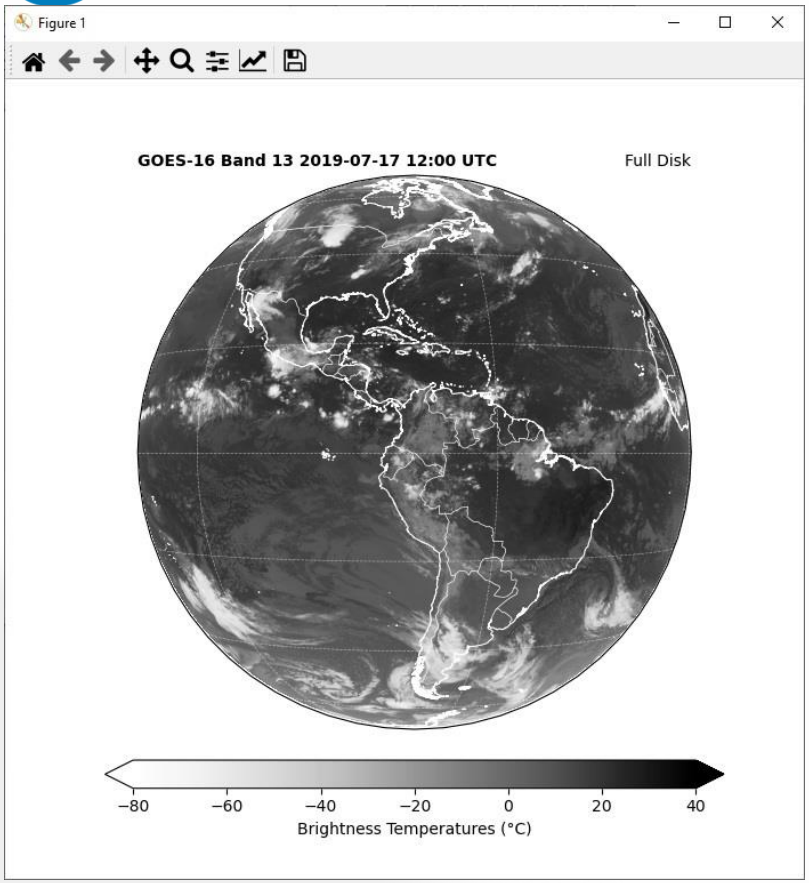
IMAGE GENERATION

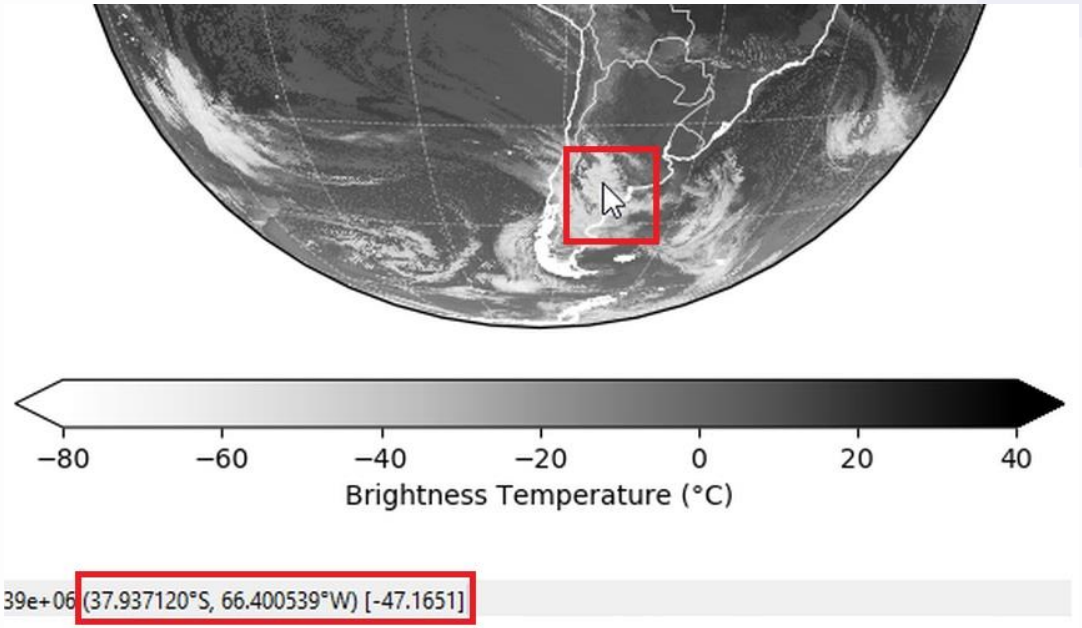
Script 3: Overlaying Maps with Cartopy





Script 3: Overlaying Maps with Cartopy





Script 3 (Extra): Time Required to Run the Script

Let's add the following commands after importing the libraries :

```
# Start the time counter
import time as t
print('Script started.')
start = t.time()
```

And the following commands after saving the image :

```
# Stop the time counter
print('Total processing time:', round((t.time() - start),2), 'seconds.')
```

```
(workshop) D:\VLAB\Python>python Script_03.py
Script started.
Total processing time: 33.85 seconds.
```

```
# Training: Python and GOES-R Imagery: Script 3 - Adding a Map with Cartopy
#-----------------------------------------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset          # Read / Write NetCDF4 files
import matplotlib.pyplot as plt      # Plotting library
from datetime import datetime        # Basic Dates and time types
import cartopy, cartopy.crs as ccrs  # Plot maps
#-----------------------------------------------------------------------------------------------------------------
# Start the time counter
import time as t
print('Script started.')
start = t.time()

# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values
data = file.variables['CMI'][:] - 273.15
#-----------------------------------------------------------------------------------------------------------------
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.Geostationary(central_longitude=-75.0, satellite_height=35786023.0))
img_extent = (-5434894.67527,5434894.67527,-5434894.67527,5434894.67527)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='white', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.5)
ax.gridlines(color='white', alpha=0.5, linestyle='--', linewidth=0.5)

# Plot the image
img = ax.imshow(data, vmin=-80, vmax=40, origin='upper', extent=img_extent, cmap='Greys')

# Add a colorbar
plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(file.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 Band 13 ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Full Disk', fontsize=10, loc='right')
#-----------------------------------------------------------------------------------------------------------------
# Save the image
plt.savefig('Image_03.png')

# Stop the time counter
print('Total processing time:', round((t.time() - start),2), 'seconds.')
```

Script 4: Reading Parameters from the Metadata

```
15
16  # Choose the plot size (width x height, in inches)
17  plt.figure(figsize=(7,7))
18
19  # Use the Geostationary projection in cartopy
20  longitude_of_projection_origin = file.variables['goes_imager_projection'].longitude_of_projection_origin
21  perspective_point_height = file.variables['goes_imager_projection'].perspective_point_height
22  ax = plt.axes(projection=ccrs.Geostationary(central_longitude=longitude_of_projection_origin, satellite_height=perspective_point_height))
23
24  # Extent of data in decimais (2712*0.000056*35786023.0
25  xmin = file.variables['x'][:].min()*perspective_point_height
26  xmax = file.variables['x'][:].max()*perspective_point_height
27  ymin = file.variables['y'][:].min()*perspective_point_height
28  ymax = file.variables['y'][:].max()*perspective_point_height
29  img_extent = (xmin, xmax, ymin, ymax)
30
31  # Add coastlines, borders and gridlines
32  ax.coastlines(resolution='10m', color='white', linewidth=0.8)
33  ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.5)
34  ax.gridlines(color='white', alpha=0.5, linestyle='--', linewidth=0.5)
35
36  # Plot the image
37  img = ax.imshow(data, vmin=-80, vmax=40, origin='upper', extent=img_extent, cmap='Greys')
38
39  # Add a colorbar
40  plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)
41
42  # Extract the date
43  date = (datetime.strptime(file.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))
44
45  # Add a title
46  plt.title('GOES-16 Band 13 ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
47  plt.title('Full Disk', fontsize=10, loc='right')
48
```

PLOT CONFIGURATION

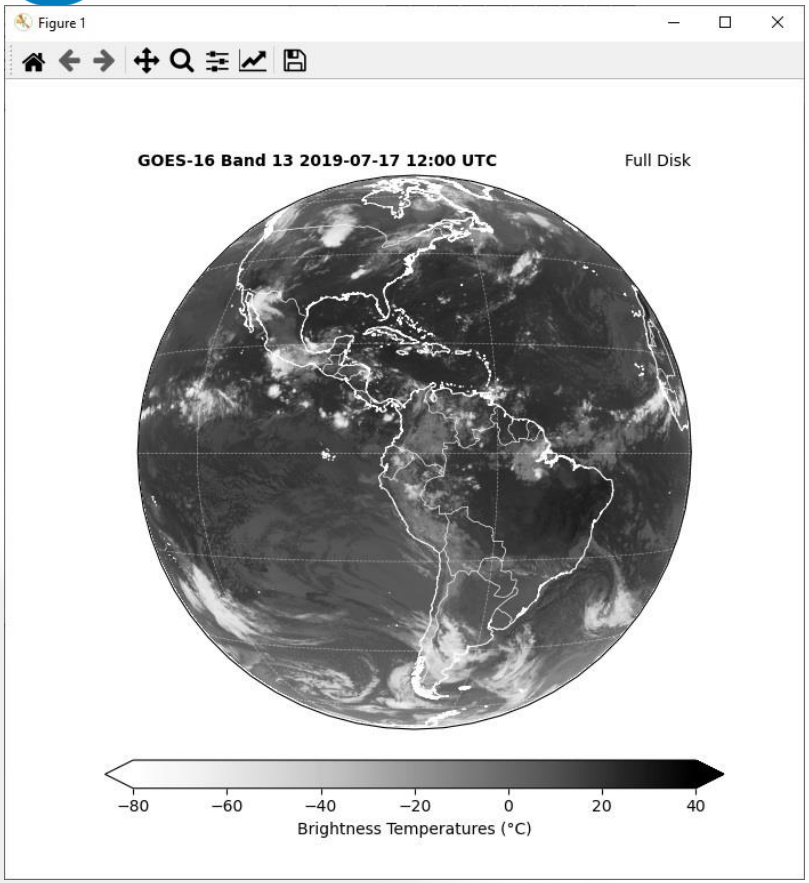
Half of the Full Disk

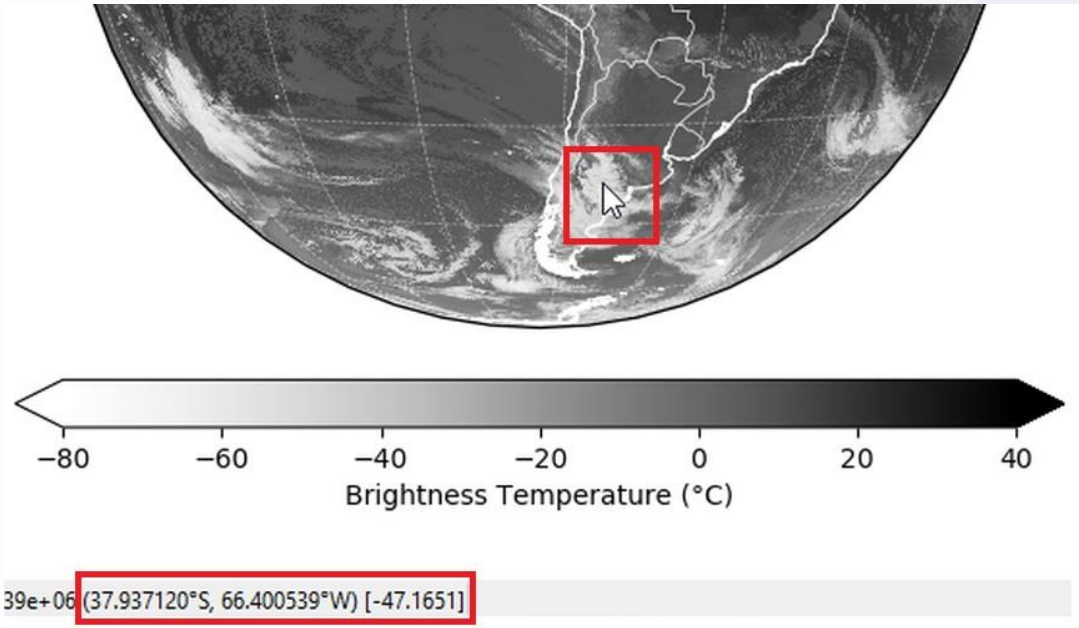
Satellite Altitude

Pixel size (radians)

READING THE PARAMETERS

Script 4: Reading Parameters from the Metadata





Script 4: Reading Parameters from the Metadata

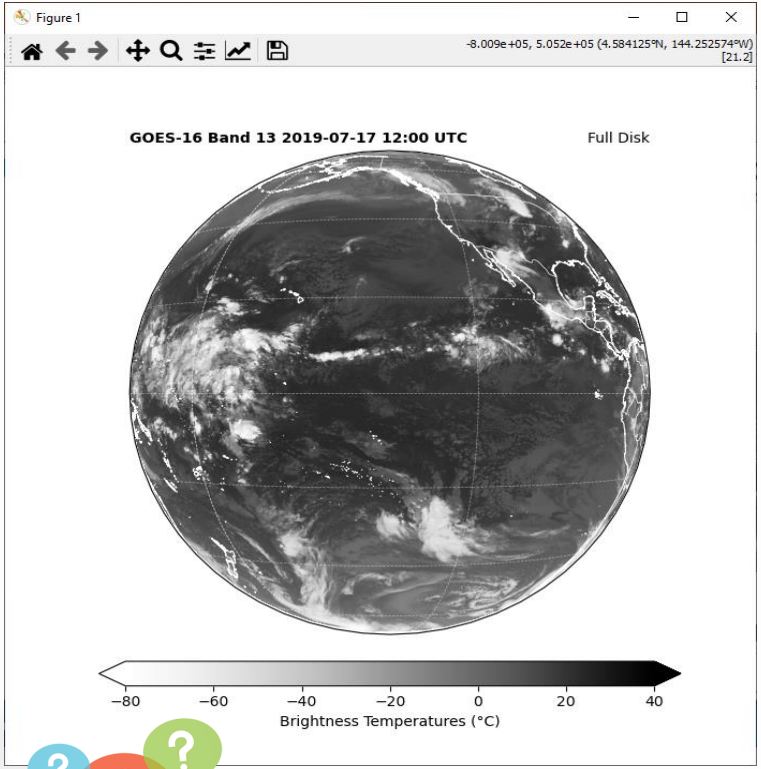
```
19  # Use the Geostationary projection in cartopy
20  ax = plt.axes(projection=ccrs.Geostationary(central_longitude=-75.0, satellite_height=35786023.0))
21  img_extent = (-5434894.67527,5434894.67527,-5434894.67527,5434894.67527)
22
```

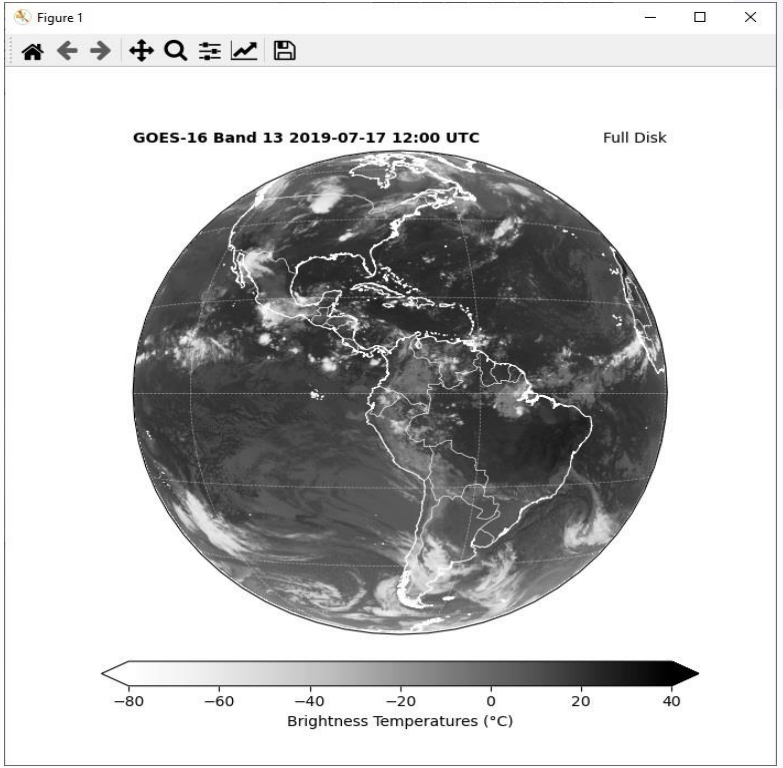
```
19  # Use the Geostationary projection in cartopy
20  longitude_of_projection_origin = file.variables['goes_imager_projection'].longitude_of_projection_origin
21  perspective_point_height = file.variables['goes_imager_projection'].perspective_point_height
22  ax = plt.axes(projection=ccrs.Geostationary(central_longitude=longitude_of_projection_origin, satellite_height=perspective_point_height))
23
24  # Extent of data in decimais (2712*0.000056*35786023.0)
25  xmin = file.variables['x'][:].min()*perspective_point_height
26  xmax = file.variables['x'][:].max()*perspective_point_height
27  ymin = file.variables['y'][:].min()*perspective_point_height
28  ymax = file.variables['y'][:].max()*perspective_point_height
29  img_extent = (xmin, xmax, ymin, ymax)
```

QUESTION: What are the advantages of reading this information from the file's metadata?

Script 4 (Extra): Reading Parameters from the Metadata

Same script, different satellites!





QUESTION: Something can still be improved. Can you identify it?

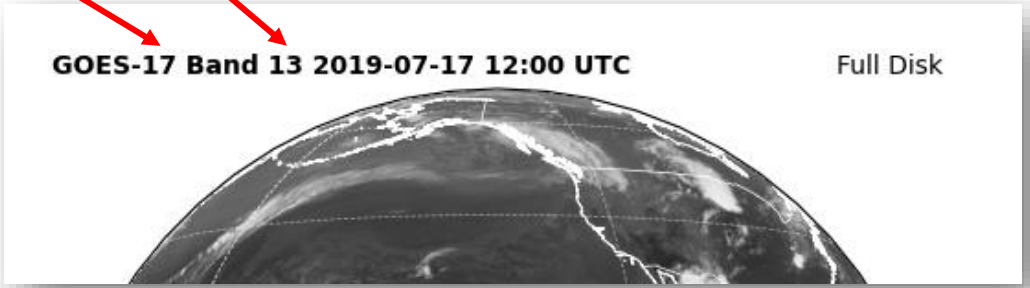
Script 4 (Extra): Reading Parameters from the Metadata

```
45  # Add a title
46  plt.title('GOES-16 Band 13 ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
47  plt.title('Full Disk', fontsize=10, loc='right')
48  #---------------------------------------------------------------------------------------------------------------
```

```
# Read the satellite
satellite = getattr(file, 'platform_ID')[1:3]
# Read the band number
band = str(file.variables['band_id'][0]).zfill(2)
```

```
45  # Read the satellite
46  satellite = getattr(file, 'platform_ID')[1:3]
47  # Read the band number
48  band = str(file.variables['band_id'][0]).zfill(2)
49
50  # Add a title
51  plt.title(f'GOES-{satellite} Band {band} ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
52  plt.title('Full Disk', fontsize=10, loc='right')
```

f -> expressions / variables enclosed in braces in the string



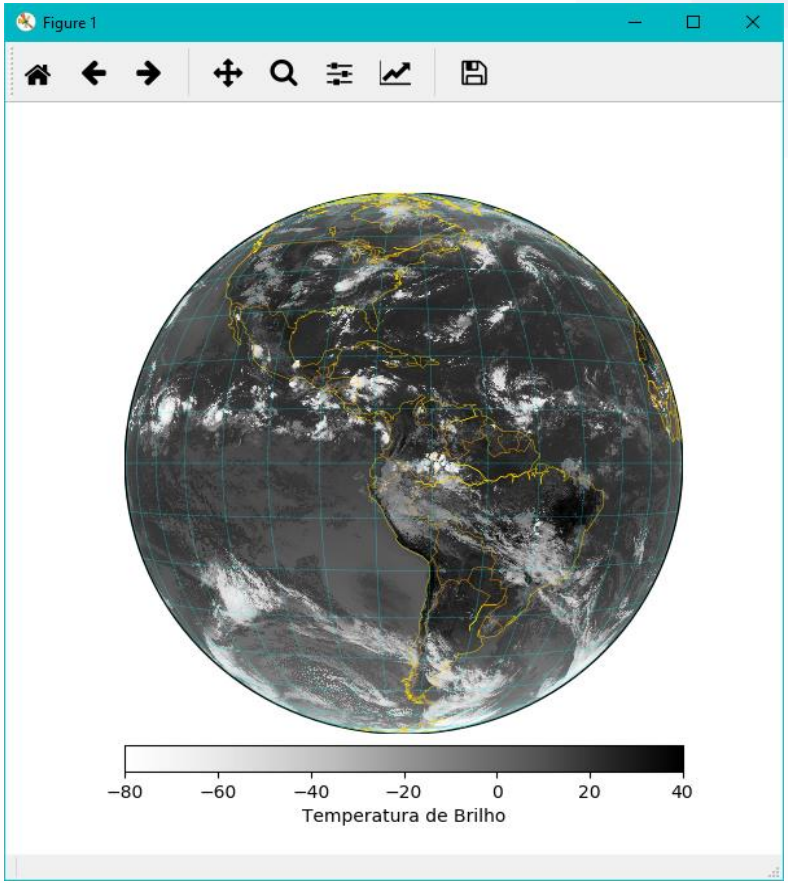
Some Quick Modifications

- The colors of the maps
- The thickness of the lines

Some Standard Color Names

Matplotlib predefined color names

black	linen	forestgreen	slategray
k	bisque	limegreen	lightsteelblue
dimgrey	darkorange	darkgreen	cornflowerblue
dimgray	burlywood	green	royalblue
grey	antiquewhite	g	ghostwhite
gray	tan	lime	lavender
darkgray	navajowhite	seagreen	midnightblue
darkgrey	blanchedalmond	mediumseagreen	navy
silver	papayawhip	springgreen	darkblue
lightgrey	moccasin	mintcream	mediumblue
lightgray	orange	mediumspringgreen	blue
gainsboro	wheat	mediumaquamarine	b
whitesmoke	oldlace	aquamarine	slateblue
white	floralwhite	turquoise	darkslateblue
w	darkgoldenrod	lightseagreen	mediumslateblue
snow	goldenrod	mediumturquoise	mediumpurple
rosybrown	cornsilk	azure	rebeccapurple
lightcoral	gold	lightcyan	blueviolet
indianred	lemonchiffon	paleturquoise	indigo
brown	khaki	darkslategray	darkorchid
firebrick	palegoldenrod	darkslategrey	darkviolet
maroon	darkkhaki	teal	mediumorchid
darkred	ivory	darkcyan	thistle
red	beige	c	plum
r	lightyellow	cyan	violet
mistyrose	lightgoldenrodyellow	aqua	purple
salmon	olive	darkturquoise	darkmagenta
tomato	y	cadetblue	m
darksalmon	yellow	powderblue	magenta
coral	olivedrab	lightblue	fuchsia
orangered	yellowgreen	deepskyblue	orchid
lightsalmon	darkolivegreen	skyblue	mediumvioletred
sienna	greenyellow	lightskyblue	deeppink
seashell	chartreuse	steelblue	hotpink
chocolate	lawngreen	aliceblue	lavenderblush
saddlebrown	honeydew	dodgerblue	palevioletred
sandybrown	darkseagreen	lightslategrey	crimson
peachpuff	palegreen	lightslategray	pink
peru	lightgreen	slategrey	lightpink



Script 5: Reading Shapefiles

```python
# Training: Python and GOES-R Imagery: Script 5 - Reading a Shapefile

# Required modules
from netCDF4 import Dataset                # Read / Write NetCDF4 files
import matplotlib.pyplot as plt            # Plotting library
from datetime import datetime              # Basic Dates and time types
import cartopy, cartopy.crs as ccrs        # Plot maps
import cartopy.io.shapereader as shpreader # Import shapefiles

# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values, and convert to Celsius
data = file.variables['CMI'][:] - 273.15

# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.Geostationary(central_longitude=-75.0, satellite_height=35786023.0))
img_extent = (-5434894.67527,5434894.67527,-5434894.67527,5434894.67527)

# Add a shapefile
# https://geoftp.ibge.gov.br/organizacao_do_territorio/malhas_territoriais/malhas_municipais/municipio_2015/Brasil/BR/br_unidades_da_federacao.zip
shapefile = list(shpreader.Reader('BR_UF_2019.shp').geometries())
ax.add_geometries(shapefile, ccrs.PlateCarree(), edgecolor='gold',facecolor='none', linewidth=0.3)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='white', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.8)
ax.gridlines(color='white', alpha=0.5, linestyle='--', linewidth=0.5)

# Plot the image
img = ax.imshow(data, vmin=-80, vmax=40, origin='upper', extent=img_extent, cmap='Greys')

# Add a colorbar
plt.colorbar(img, label='Brightness Temperature (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(file.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 Band 13 ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Full Disk', fontsize=10, loc='right')

# Save the image
plt.savefig('Image_05.png')

# Show the image
plt.show()
```

LIBRARIES

DATA READING AND MANIPULATION

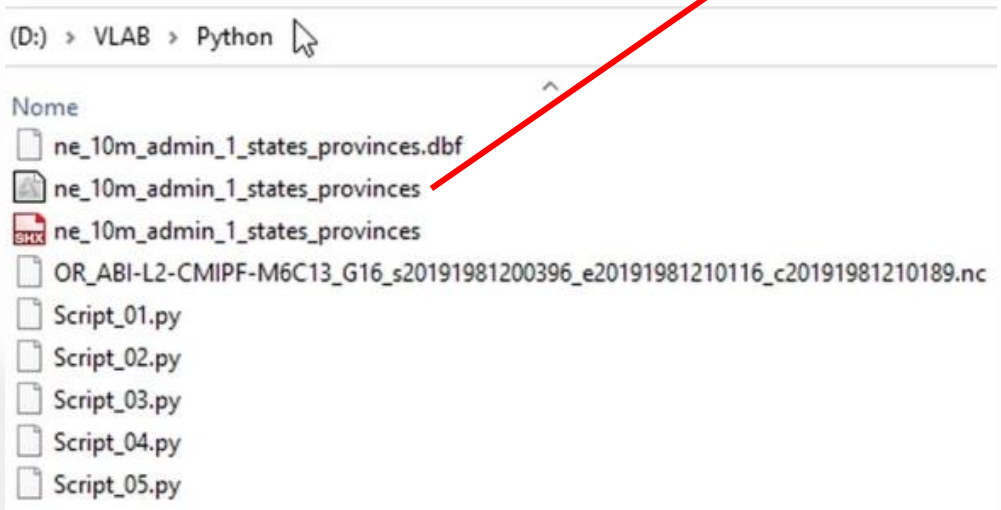
Shapefile reading

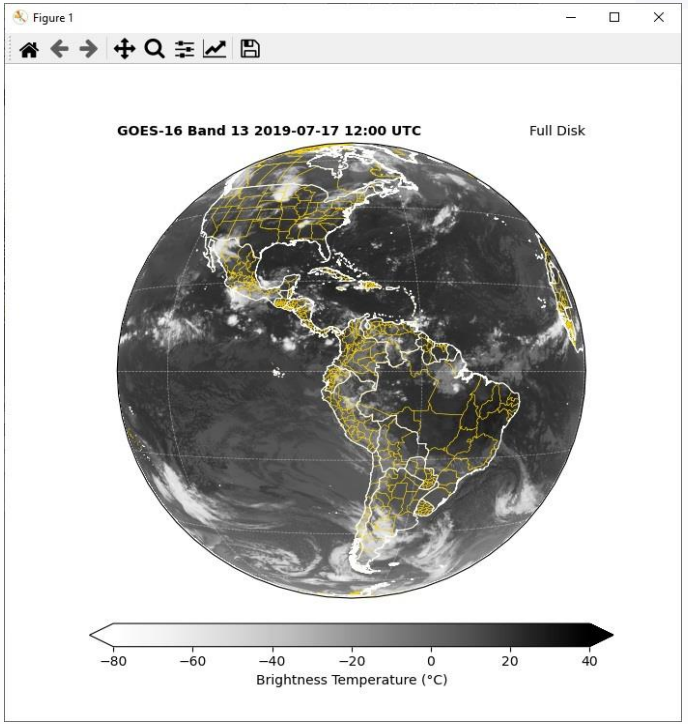
PLOT CONFIGURATION

IMAGE GENERATION

Script 5: Reading Shapefiles

```
24  # Add a shapefile
25  shapefile = list(shpreader.Reader('ne_10m_admin_1_states_provinces.shp').geometries())
26  ax.add_geometries(shapefile, ccrs.PlateCarree(), edgecolor='gold',facecolor='none', linewidth=0.3)
```

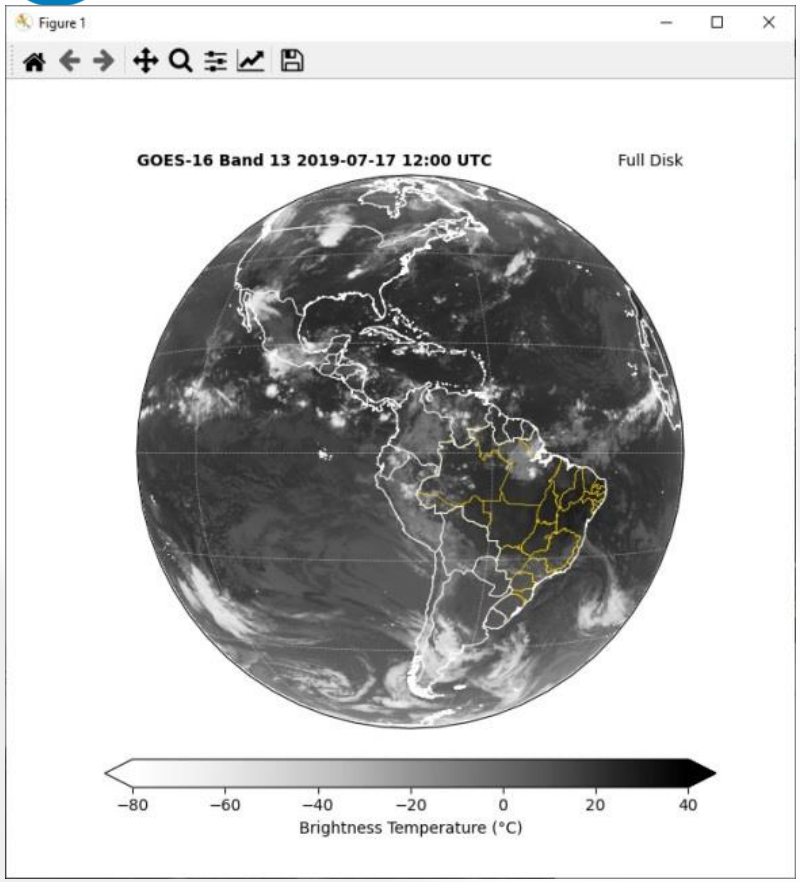


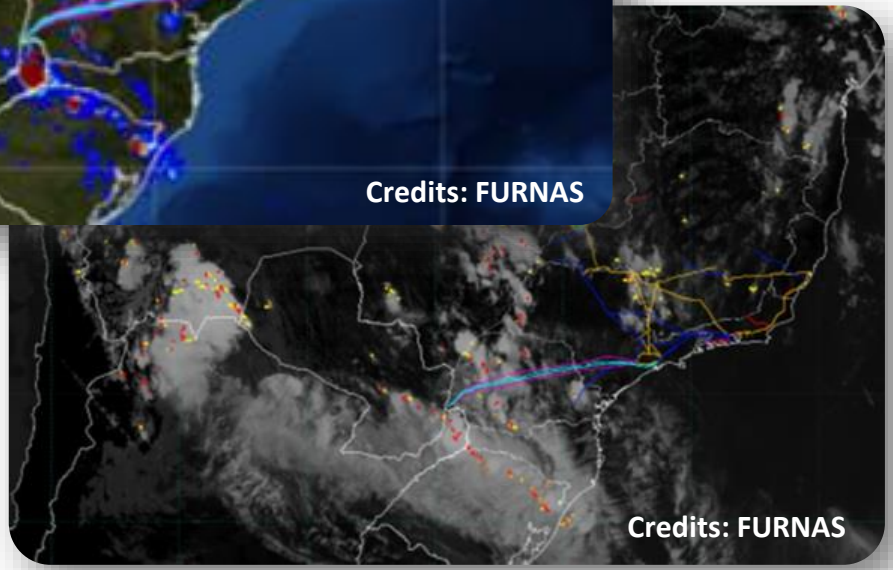


Some Quick Modifications

- The shapefile to be read
- The colors of the shapefile

Script 5: Reading Shapefiles (other examples)





Script 6: Basic ABI + GLM Plot

```
30  #-----------------------------------------------------------------------------------------------------------------
31  # Open the GLM file
32  # Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
33  fileGLM = Dataset("OR_GLM-L2-LCFA_G16_s20191981200000_e20191981200200_c20191981200224.nc")
34
35  e_lats = fileGLM.variables['event_lat'][:]
36  e_lons = fileGLM.variables['event_lon'][:]
37
38  g_lats = fileGLM.variables['group_lat'][:]
39  g_lons = fileGLM.variables['group_lon'][:]
40
41  f_lats = fileGLM.variables['flash_lat'][:]
42  f_lons = fileGLM.variables['flash_lon'][:]
43
44  img2 = ax.plot(e_lons,e_lats,'.r', markersize=10, transform=ccrs.PlateCarree(), alpha=0.01)
45  img3 = ax.plot(g_lons,g_lats,'.y', markersize=5, transform=ccrs.PlateCarree(), alpha=0.5)
46  img4 = ax.plot(f_lons,f_lats,'.g', markersize=2.5, transform=ccrs.PlateCarree(), alpha=1)
47  #-----------------------------------------------------------------------------------------------------------------
```

https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.plot.html

Type and Color of the Symbol

Size

Transparency

More information about GLM (events, groups and flashes):

https://rammb.cira.colostate.edu/training/visit/quick_guides/GLM_Quick_Guide_Detection_Methods_June_2018.pdf

Script 6: Basic ABI + GLM Plot

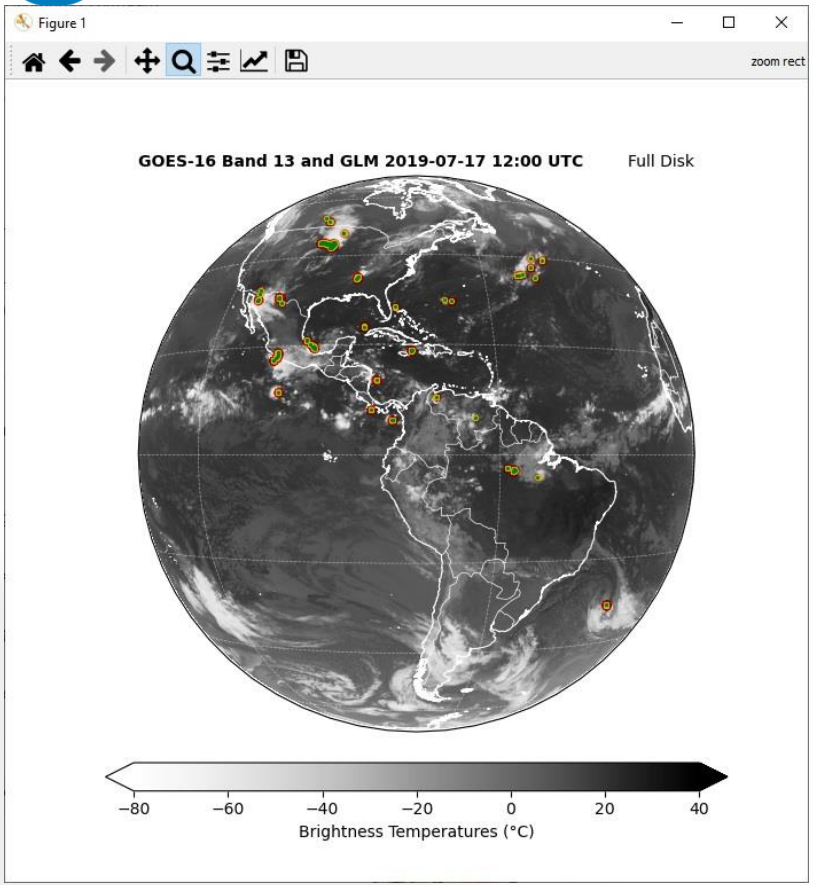
```python
# Training: Python and GOES-R Imagery: Script 6 - ABI and GLM
#-----------------------------------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset          # Read / Write NetCDF4 files
import matplotlib.pyplot as plt      # Plotting library
from datetime import datetime        # Basic Dates and time types
import cartopy, cartopy.crs as ccrs  # Plot maps
#-----------------------------------------------------------------------------------------------------------
# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values
data = file.variables['CMI'][:] - 273.15

# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.Geostationary(central_longitude=-75.0, satellite_height=35786023.0))
img_extent = (-5434894.67527,5434894.67527,-5434894.67527,5434894.67527)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='white', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.8)
ax.gridlines(color='white', alpha=0.5, linestyle='--', linewidth=0.5)

# Plot the image
img = ax.imshow(data, vmin=-80, vmax=40, origin='upper', extent=img_extent, cmap='Greys')
#-----------------------------------------------------------------------------------------------------------
# Open the GLM file
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
fileGLM = Dataset("OR_GLM-L2-LCFA_G16_s20191981200000_e20191981200200_c20191981200224.nc")

e_lats = fileGLM.variables['event_lat'][:]
e_lons = fileGLM.variables['event_lon'][:]

g_lats = fileGLM.variables['group_lat'][:]
g_lons = fileGLM.variables['group_lon'][:]

f_lats = fileGLM.variables['flash_lat'][:]
f_lons = fileGLM.variables['flash_lon'][:]

img2 = ax.plot(e_lons,e_lats,'.r', markersize=10, transform=ccrs.PlateCarree(), alpha=0.01)
img3 = ax.plot(g_lons,g_lats,'.y', markersize=5, transform=ccrs.PlateCarree(), alpha=0.5)
img4 = ax.plot(f_lons,f_lats,'.g', markersize=2.5, transform=ccrs.PlateCarree(), alpha=1)
#-----------------------------------------------------------------------------------------------------------
# Add a colorbar
plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(file.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 Band 13 and GLM ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Full Disk', fontsize=10, loc='right')
#-----------------------------------------------------------------------------------------------------------
# Save the image
plt.savefig('Image_06.png')
```

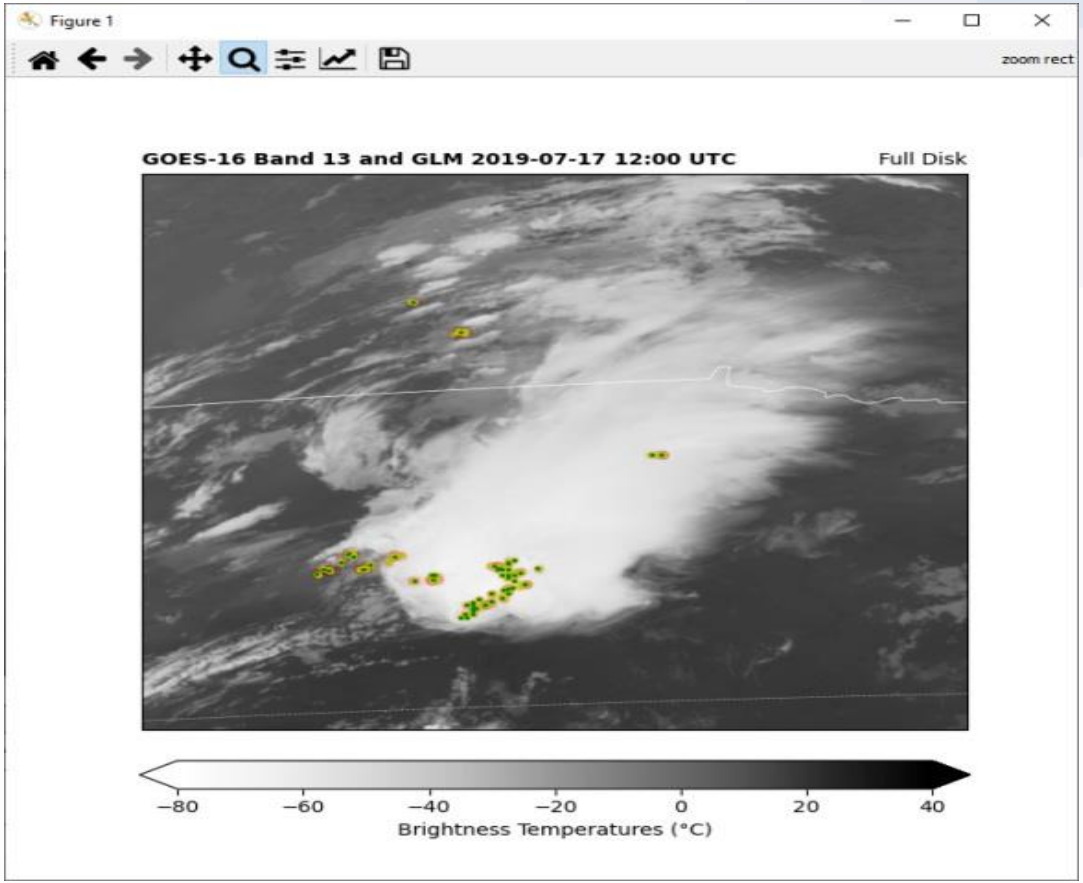
Same Content of Script 3

With the following additional “block”:

READING, MANIPULARION AND GLM CONF.

Script 6: Basic ABI + GLM Plot

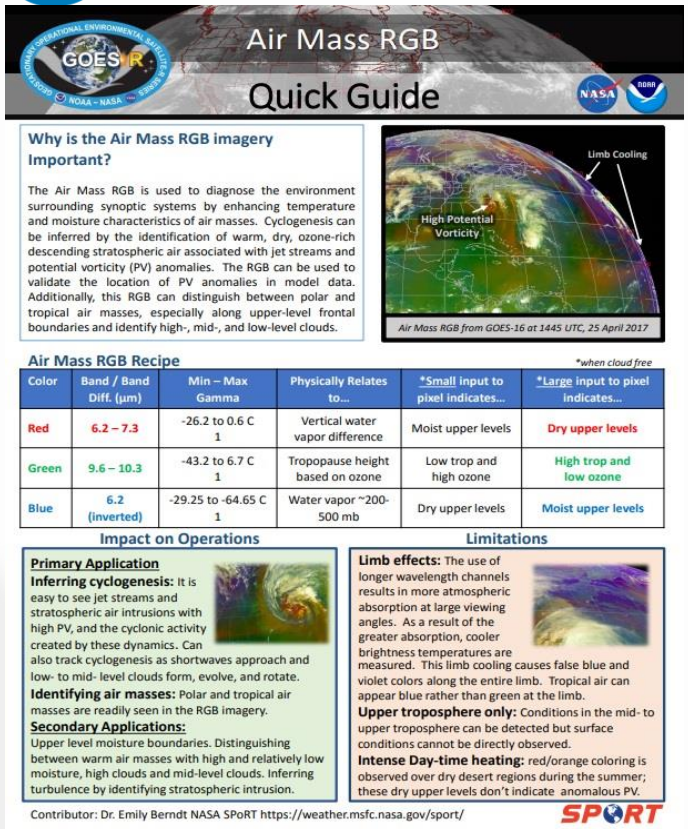




Some Quick Modifications

- The appearance of the points (colors and sizes)

Script 7: Creating RGB Composites (Airmass)



Air Mass RGB

Quick Guide

Why is the Air Mass RGB imagery Important?

The Air Mass RGB is used to diagnose the environment surrounding synoptic systems by enhancing temperature and moisture characteristics of air masses. Cyclogenesis can be inferred by the identification of warm, dry, ozone-rich descending stratospheric air associated with jet streams and potential vorticity (PV) anomalies. The RGB can be used to validate the location of PV anomalies in model data. Additionally, this RGB can distinguish between polar and tropical air masses, especially along upper-level frontal boundaries and identify high-, mid-, and low-level clouds.

Air Mass RGB from GOES-16 at 1445 UTC, 25 April 2017

Air Mass RGB Recipe *when cloud free

Color	Band / Band Diff. (µm)	Min – Max Gamma	Physically Relates to…	*Small input to pixel indicates…	*Large input to pixel indicates…
Red	6.2 – 7.3	-26.2 to 0.6 C 1	Vertical water vapor difference	Moist upper levels	Dry upper levels
Green	9.6 – 10.3	-43.2 to 6.7 C 1	Tropopause height based on ozone	Low trop and high ozone	High trop and low ozone
Blue	6.2 (inverted)	-29.25 to -64.65 C 1	Water vapor ~200-500 mb	Dry upper levels	Moist upper levels

Impact on Operations

Primary Application

Inferring cyclogenesis: It is easy to see jet streams and stratospheric air intrusions with high PV, and the cyclonic activity created by these dynamics. Can also track cyclogenesis as shortwaves approach and low- to mid- level clouds form, evolve, and rotate.

Identifying air masses: Polar and tropical air masses are readily seen in the RGB imagery.

Secondary Applications:

Upper level moisture boundaries. Distinguishing between warm air masses with high and relatively low moisture, high clouds and mid-level clouds. Inferring turbulence by identifying stratospheric intrusion.

Limitations

Limb effects: The use of longer wavelength channels results in more atmospheric absorption at large viewing angles. As a result of the greater absorption, cooler brightness temperatures are measured. This limb cooling causes false blue and violet colors along the entire limb. Tropical air can appear blue rather than green at the limb.

Upper troposphere only: Conditions in the mid- to upper troposphere can be detected but surface conditions cannot be directly observed.

Intense Day-time heating: red/orange coloring is observed over dry desert regions during the summer; these dry upper levels don't indicate anomalous PV.

Contributor: Dr. Emily Berndt NASA SPoRT https://weather.msfc.nasa.gov/sport/

SPORT

Air Mass RGB Recipe

Color	Band / Band Diff. (µm)	Min – Max Gamma
Red	6.2 – 7.3	-26.2 to 0.6 C 1
Green	9.6 – 10.3	-43.2 to 6.7 C 1
Blue	6.2 (inverted)	-29.25 to -64.65 C 1

http://rammb.cira.colostate.edu/training/visit/quick_guides/QuickGuide_GOESR_AirMassRGB_final.pdf

GOES-16 Channels (ABI Sensor)

ABI Band	Central Wavelength (µm)	Wavelength Range (µm)	Resolution (km)	Spectral range	Descriptive Name
1	0.47	0.45 - 0.49	1	Visible	Blue
2	0.64	0.68 - 0.68	0.5	Visible	Red
3	0.86	0.847 - 0.882	1	Near Infrared	Vegetation
4	1.37	1.366 - 1.380	2	Near Infrared	Cirrus
5	1.6	1.59 - 1.63	1	Near Infrared	Snow/Ice
6	2.2	2.22 - 2.27	2	Near Infrared	Cloud Particle Size
7	3.9	3.80 - 3.99	2	Shortwave Infrared	Shortwave window
8	6.2	5.79 - 6.59	2	Midwave Infrared	Upper-level water vapor
9	6.9	6.72 - 7.14	2	Midwave Infrared	Midlevel water vapor
10	7.3	7.24 - 7.43	2	Midwave Infrared	Lower / midlevel watervapor
11	8.4	8.23 - 8.66	2	Longwave Infrared	Cloud-top phase
12	9.6	9.42 - 9.80	2	Longwave Infrared	Ozone
13	10.3	10.18 - 10.48	2	Longwave Infrared	Clean longwave window
14	11.2	10.82 - 11.60	2	Longwave Infrared	Longwave window
15	12.3	11.83 - 12.75	2	Longwave Infrared	Dirty longwave window
16	13.3	12.99-13.56	2	Longwave Infrared	CO2

Script 7: Creating RGB Composites (Airmass)

```python
# Training: Python and GOES-R Imagery: Script 7 - RGB Creation
#-----------------------------------------------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset                # Read / Write NetCDF4 files
import matplotlib.pyplot as plt            # Plotting library
from datetime import datetime              # Basic Dates and time types
import cartopy, cartopy.crs as ccrs        # Plot maps
import numpy as np                         # Import the Numpy packag
#-----------------------------------------------------------------------------------------------------------------------
# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file1 = Dataset("OR_ABI-L2-CMIPF-M6C08_G16_s20191981200396_e20191981210104_c20191981210182.nc")
# Get the pixel values, and convert to Celsius
data1 = file1.variables['CMI'][:,:][::4 ,::4] - 273.15

# Open the GOES-R image
file2 = Dataset("OR_ABI-L2-CMIPF-M6C10_G16_s20191981200396_e20191981210116_c20191981210188.nc")
# Get the pixel values, and convert to Celsius
data2 = file2.variables['CMI'][:,:][::4 ,::4] - 273.15

# Open the GOES-R image
file3 = Dataset("OR_ABI-L2-CMIPF-M6C12_G16_s20191981200396_e20191981210111_c20191981210185.nc")
# Get the pixel values, and convert to Celsius
data3 = file3.variables['CMI'][:,:][::4 ,::4] - 273.15

# Open the GOES-R image
file4 = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")
# Get the pixel values, and convert to Celsius
data4 = file4.variables['CMI'][:,:][::4 ,::4] - 273.15
#-----------------------------------------------------------------------------------------------------------------------
```

LIBRARIES

DATA READING

Script 7: Creating RGB Composites (Airmass)

```
#---------------------------------------------------------------------------
# RGB Quick Guide: http://rammb.cira.colostate.edu/t

# RGB Components
R = data1 - data2
G = data3 - data4
B = data1

# Minimuns, Maximuns and Gamma
Rmin = -26.2
Rmax = 0.6

Gmin = -43.2
Gmax = 6.7

Bmin = -29.25
Bmax = -64.65

R[R > Rmax] = Rmax
R[R < Rmin] = Rmin

G[G > Gmax] = Gmax
G[G < Gmin] = Gmin

B[B < Bmax] = Bmax # Inverted!
B[B > Bmin] = Bmin # Inverted!

gamma_R = 1
gamma_G = 1
gamma_B = 1

# Normalize the data
R = ((R - Rmin) / (Rmax - Rmin)) ** (1/gamma_R)
G = ((G - Gmin) / (Gmax - Gmin)) ** (1/gamma_G)
B = ((B - Bmin) / (Bmax - Bmin)) ** (1/gamma_B)

# Create the RGB
RGB = np.stack([R, G, B], axis=2)

# Eliminate values outside the globe
mask = (RGB == [R[0,0],G[0,0],B[0,0]]).all(axis=2)
RGB[mask] = np.nan
#---------------------------------------------------------------------------
```

DATA MANIP.

Air Mass RGB Recipe

Color	Band / Band Diff. (µm)	Min – Max Gamma
Red	6.2 – 7.3	-26.2 to 0.6 C 1
Green	9.6 – 10.3	-43.2 to 6.7 C 1
Blue	6.2 (inverted)	-29.25 to -64.65 C 1

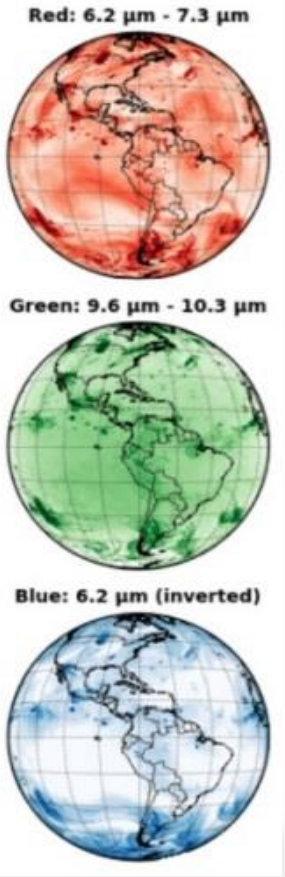
Script 7: Creating RGB Composites (Airmass)

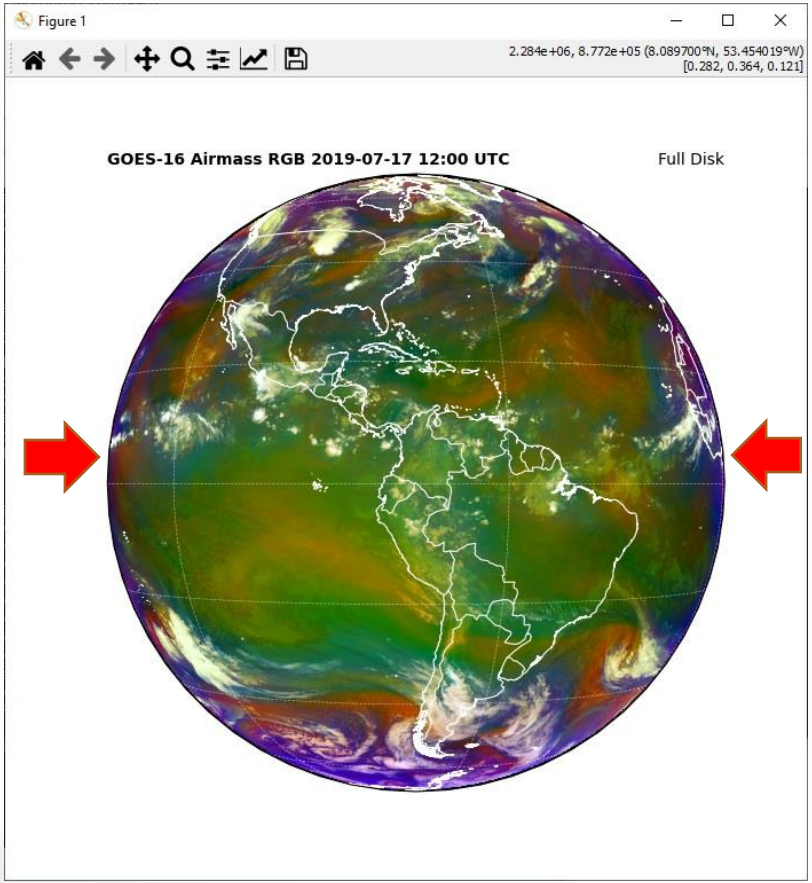
```
#-----------------------------------------------------------------------------------------------------------------------
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.Geostationary(central_longitude=-75.0, satellite_height=35786023.0))
img_extent = (-5434894.67527,5434894.67527,-5434894.67527,5434894.67527)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='white', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.8)
ax.gridlines(color='white', alpha=0.5, linestyle='--', linewidth=0.5)

# Plot the image
img = ax.imshow(RGB, origin='upper', extent=img_extent)

# Extract the date
date = (datetime.strptime(file1.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 Airmass RGB ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Full Disk', fontsize=10, loc='right')
#-----------------------------------------------------------------------------------------------------------------------
# Save the image
plt.savefig('Image_07.png')

# Show the image
plt.show()
```

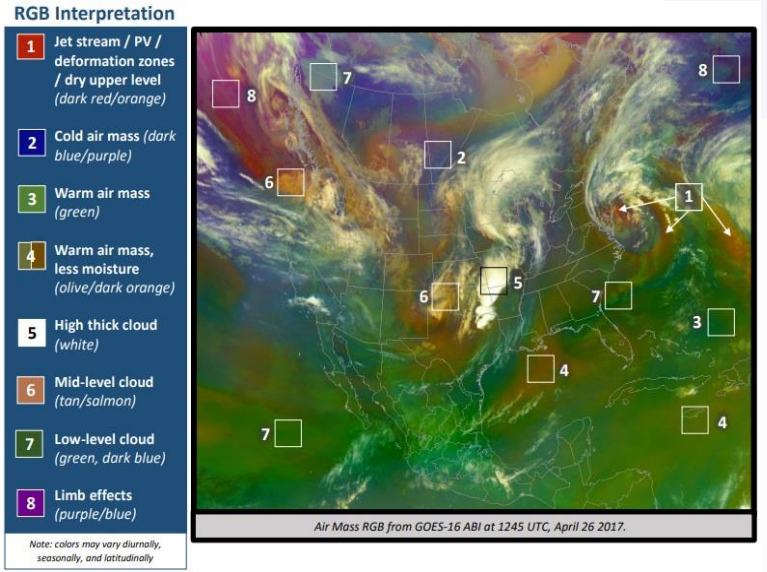
PLOT CONFIGURATION

IMAGE GENERATION

Script 7: Creating RGB Composites (Airmass)







Some Quick Modifications

- **Creation of other RGBs:**

  ex.: Ash, Dust, SO2

http://rammb.cira.colostate.edu/training/visit/quick_guides/

Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

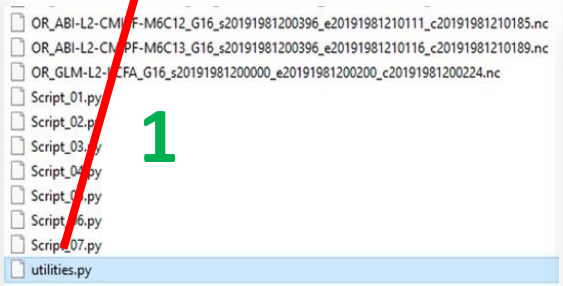
- **Calling functions from external files**
- **Creation of custom color palettes**
- **Multiple Plots**

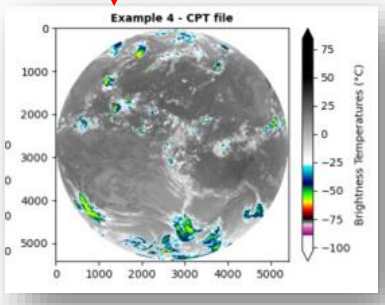
```
# Training: Python and GOES-R Imag
#-------------------------------------
# Required modules
from netCDF4 import Dataset
import matplotlib.pyplot as plt
from datetime import datetime
from matplotlib import cm
import numpy as np
from utilities import loadCPT
#-------------------------------------
```

```
54  #-------------------------------------------------------------
55
56  # COLORMAP EXAMPLE 4
57  # Converts a CPT file to be used in Python
58  # CPT archive: http://soliton.vm.bytemark.co.uk/pub/cpt-city/
59
60  cpt = loadCPT('IR4AVHRR6.cpt')
61  my_cmap4 = cm.colors.LinearSegmentedColormap('cpt', cpt)
62  vmin4 = -103.0
63  vmax4 = 84.0
64
65  #-------------------------------------------------------------
```

```
def loadCPT(path):

    try:
        f = open(path)
    except:
        print ("File ", path, "not found")
        return None

    lines = f.readlines()

    f.close()

    x = np.array([])
    r = np.array([])
    g = np.array([])
    b = np.array([])

    colorModel = 'RGB'

    for l in lines:
        ls = l.split()
        if l[0] == '#':
            if ls[-1] == 'HSV':
                colorModel = 'HSV'
                continue
            else:
                continue
        if ls[0] == 'B' or ls[0] == 'F' or ls[0] == 'N':
            pass
        else:
            x=np.append(x,float(ls[0]))
            r=np.append(r,float(ls[1]))
            g=np.append(g,float(ls[2]))
            b=np.append(b,float(ls[3]))
            xtemp = float(ls[4])
            rtemp = float(ls[5])
            gtemp = float(ls[6])
            btemp = float(ls[7])

            x=np.append(x,xtemp)
            r=np.append(r,rtemp)
            g=np.append(g,gtemp)
            b=np.append(b,btemp)

    if colorModel == 'HSV':
        for i in range(r.shape[0]):
            rr, gg, bb = colorsys.hsv_to_rgb(r[i]/360.,g[i],b[i])
        r[i] = rr ; g[i] = gg ; b[i] = bb

    if colorModel == 'RGB':
        r = r/255.0
        g = g/255.0
        b = b/255.0

    xNorm = (x - x[0])/(x[-1] - x[0])

    red   = []
    blue  = []
    green = []

    for i in range(len(x)):
        red.append([xNorm[i],r[i],r[i]])
        green.append([xNorm[i],g[i],g[i]])
        blue.append([xNorm[i],b[i],b[i]])

    colorDict = {'red': red, 'green': green, 'blue': blue}

    return colorDict
```

Función dentro del archivo utilities.py





1

2

3

4

Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

- **Calling functions from external files**
- **Creation of custom color palettes**
- **Multiple Plots**

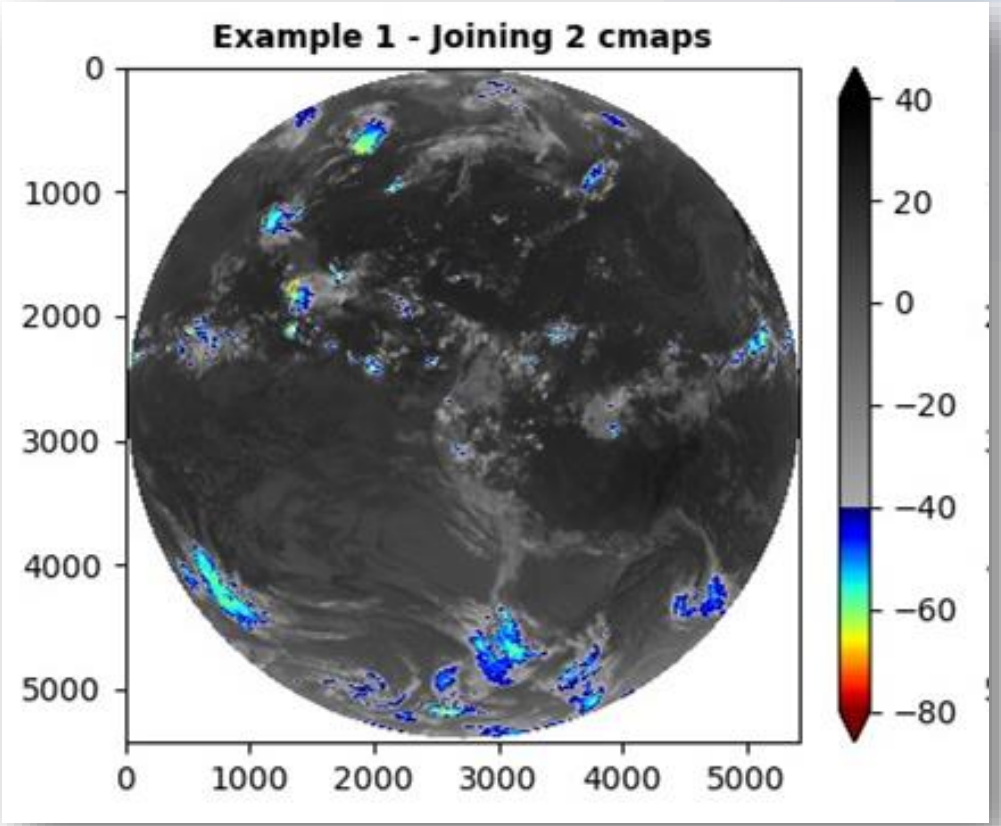
```
10  #---------------------------------------------------------------------------------------------
11
12  # COLORMAP EXAMPLE 1
13  # Custom colormap joining matplotlib colormaps
14  # Available matplotlib colormaps: https://matplotlib.org/stable/tutorials/colors/colormaps.html
15
16  vmin1 = -80                                                    # Min. value
17  vmax1 = 40                                                     # Max. value
18
19  gray_cmap = cm.get_cmap('gray_r', 120)                         # Read the reversed 'gray' cmap
20  gray_cmap = gray_cmap(np.linspace(0, 1, 120))                  # Create the array
21  jet_cmap  = cm.get_cmap('jet_r', 40)                           # Read the reversed 'jet' cmap
22  jet_cmap  = jet_cmap(np.linspace(0, 1, 40))                    # Create the array
23  gray_cmap[:40, :] = jet_cmap                                   # Join both cmaps arrays
24  my_cmap1 = cm.colors.ListedColormap(gray_cmap)                 # Create the custom colormap
25
26  #---------------------------------------------------------------------------------------------
```

Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

- **Calling functions from external files**
- **Creation of custom color palettes**
- **Multiple Plots**

```
#---------------------------------------------------------------------------

# COLORMAP EXAMPLE 1
# Custom colormap joining matplotlib colormaps
# Available matplotlib colormaps: https://matplotlib.org/stable/tutorials/colors/colormaps.html

vmin1 = -80                                          # Min. value
vmax1 = 40                                           # Max. value

gray_cmap = cm.get_cmap('gray_r', 120)               # Read the reversed 'gray' cmap
gray_cmap = gray_cmap(np.linspace(0, 1, 120))        # Create the array
jet_cmap  = cm.get_cmap('jet_r', 40)                 # Read the reversed 'jet' cmap
jet_cmap  = jet_cmap(np.linspace(0, 1, 40))          # Create the array
gray_cmap[:40, :] = jet_cmap                         # Join both cmaps arrays
my_cmap1 = cm.colors.ListedColormap(gray_cmap)       # Create the custom colormap

#---------------------------------------------------------------------------
```



Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

- Calling functions from external files
- **Creation of custom color palettes**
- Multiple Plots

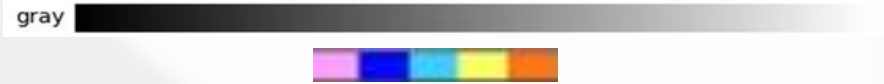
```
26  #---------------------------------------------------------------------------------------------------------------
27
28  # COLORMAP EXAMPLE 2
29  # Creating the INPE DISSM IR colormap
30  # Online color picker: https://imagecolorpicker.com/
31
32  vmin2 = -80                                                    # Min. value
33  vmax2 = 40                                                     # Max. value
34
35  gray_cmap = cm.get_cmap('gray_r', 120)                         # Read the reversed 'gray' cmap
36  gray_cmap = gray_cmap(np.linspace(0, 1, 120))                  # Create the array
37  colors = ["#ffa0ff", "#0806ff", "#3bcfff", "#feff65", "#ff7516"]  # Custom colors
38  my_colors = cm.colors.ListedColormap(colors)                   # Create a custom colormap
39  my_colors = my_colors(np.linspace(0, 1, 50))                   # Create the array
40  gray_cmap[:50, :] = my_colors                                  # Join both cmaps arrays
41  my_cmap2 = cm.colors.ListedColormap(gray_cmap)                 # Create the custom colormap
42
43  #---------------------------------------------------------------------------------------------------------------
```

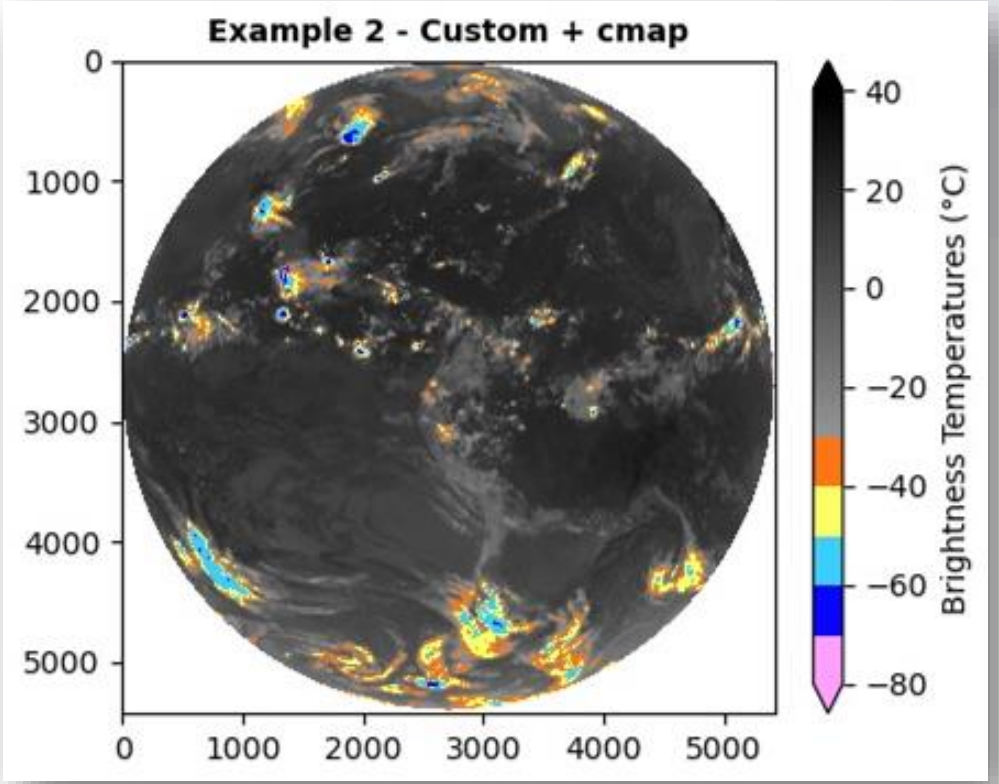
Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

- **Calling functions from external files**
- **Creation of custom color palettes**
- **Multiple Plots**

```
26  #---------------------------------------------------------------------------------------------
27
28  # COLORMAP EXAMPLE 2
29  # Creating the INPE DISSM IR colormap
30  # Online color picker: https://imagecolorpicker.com/
31
32  vmin2 = -80                                                 # Min. value
33  vmax2 = 40                                                  # Max. value
34
35  gray_cmap = cm.get_cmap('gray_r', 120)                      # Read the reversed 'gray' cmap
36  gray_cmap = gray_cmap(np.linspace(0, 1, 120))               # Create the array
37  colors = ["#ffa0ff", "#0806ff", "#3bcfff", "#feff65", "#ff7516"]  # Custom colors
38  my_colors = cm.colors.ListedColormap(colors)                # Create a custom colormap
39  my_colors = my_colors(np.linspace(0, 1, 50))                # Create the array
40  gray_cmap[:50, :] = my_colors                               # Join both cmaps arrays
41  my_cmap2 = cm.colors.ListedColormap(gray_cmap)              # Create the custom colormap
42
43  #---------------------------------------------------------------------------------------------
```





Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

- **Calling functions from external files**
- **Creation of custom color palettes**
- **Multiple Plots**

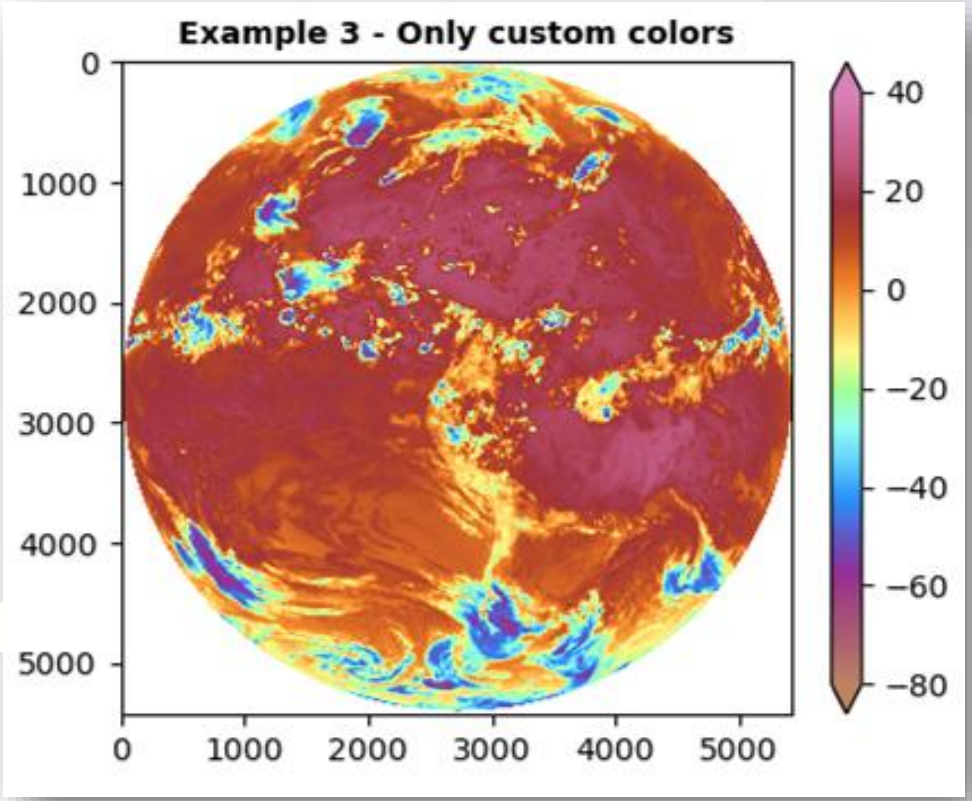
```
43  #---------------------------------------------------------------------------------------------------
44
45  # COLORMAP EXAMPLE 3
46  # Creating a linear colormap
47  # Online color picker: https://imagecolorpicker.com/
48
49  colors = ["#bc8462", "#ae656f", "#a44a79", "#962e97", "#6158c5", "#2b8ffb", "#5fcdff", "#94fff0", \
50  "#a5ff94", "#fff88c", "#ffbf52", "#ec7b27", "#b84827", "#a1333d", "#bd5478", "#cc6a99", "#d982b8"]
51  my_cmap3 = cm.colors.LinearSegmentedColormap.from_list("", colors)# Create a custom linear colormap
52  vmin3 = -80                                                       # Min. value
53  vmax3 = 40                                                        # Max. value
54
55  #---------------------------------------------------------------------------------------------------
```

Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

- **Calling functions from external files**
- **Creation of custom color palettes**
- **Multiple Plots**

```
43  #---------------------------------------------------------------------------------------------------
44
45  # COLORMAP EXAMPLE 3
46  # Creating a linear colormap
47  # Online color picker: https://imagecolorpicker.com/
48
49  colors = ["#bc8462", "#ae656f", "#a44a79", "#962e97", "#6158c5", "#2b8ffb", "#5fcdff", "#94fff0", \
50  "#a5ff94", "#fff88c", "#ffbf52", "#ec7b27", "#b84827", "#a1333d", "#bd5478", "#cc6a99", "#d982b8"]
51  my_cmap3 = cm.colors.LinearSegmentedColormap.from_list("", colors)# Create a custom linear colormap
52  vmin3 = -80                                                       # Min. value
53  vmax3 = 40                                                        # Max. value
54
55  #---------------------------------------------------------------------------------------------------
```



Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

- **Calling functions from external files**
- **Creation of custom color palettes**
- **Multiple Plots**

http://soliton.vm.bytemark.co.uk/pub/cpt-city/

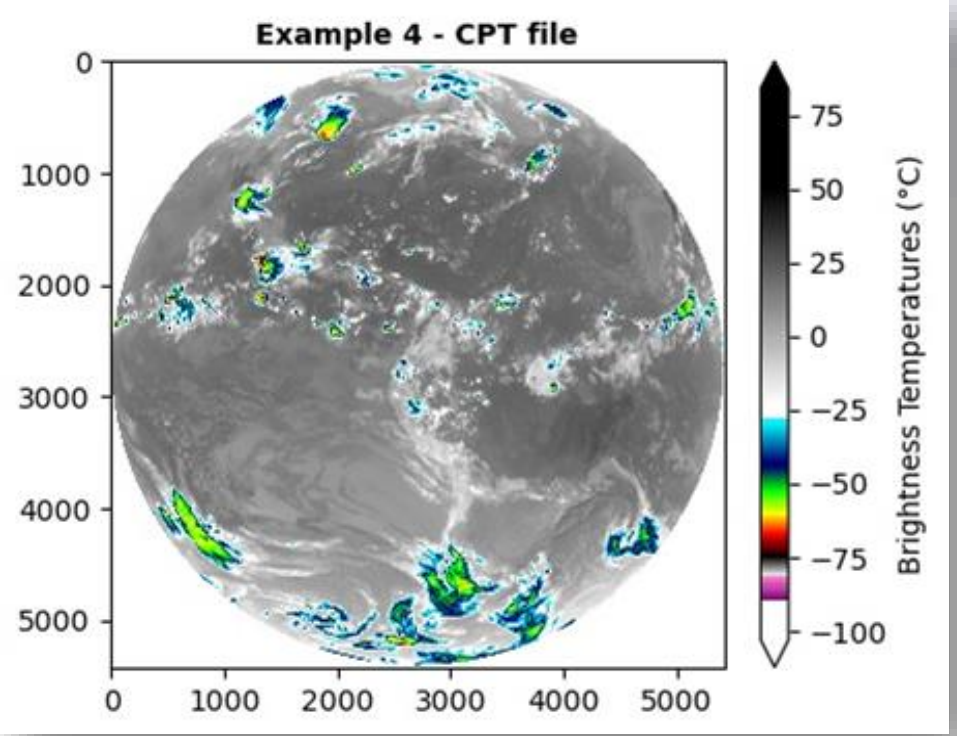
```
54  #---------------------------------------------------------------------
55
56  # COLORMAP EXAMPLE 4
57  # Converts a CPT file to be used in Python
58  # CPT archive: http://soliton.vm.bytemark.co.uk/pub/cpt-city/
59
60  cpt = loadCPT('IR4AVHRR6.cpt')
61  my_cmap4 = cm.colors.LinearSegmentedColormap('cpt', cpt)
62  vmin4 = -103.0
63  vmax4 = 84.0
64
65  #---------------------------------------------------------------------
```

Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

- **Calling functions from external files**
- **Creation of custom color palettes**
- **Multiple Plots**

```
54  #---------------------------------------------------------------------
55
56  # COLORMAP EXAMPLE 4
57  # Converts a CPT file to be used in Python
58  # CPT archive: http://soliton.vm.bytemark.co.uk/pub/cpt-city/
59
60  cpt = loadCPT('IR4AVHRR6.cpt')
61  my_cmap4 = cm.colors.LinearSegmentedColormap('cpt', cpt)
62  vmin4 = -103.0
63  vmax4 = 84.0
64
65  #---------------------------------------------------------------------
```



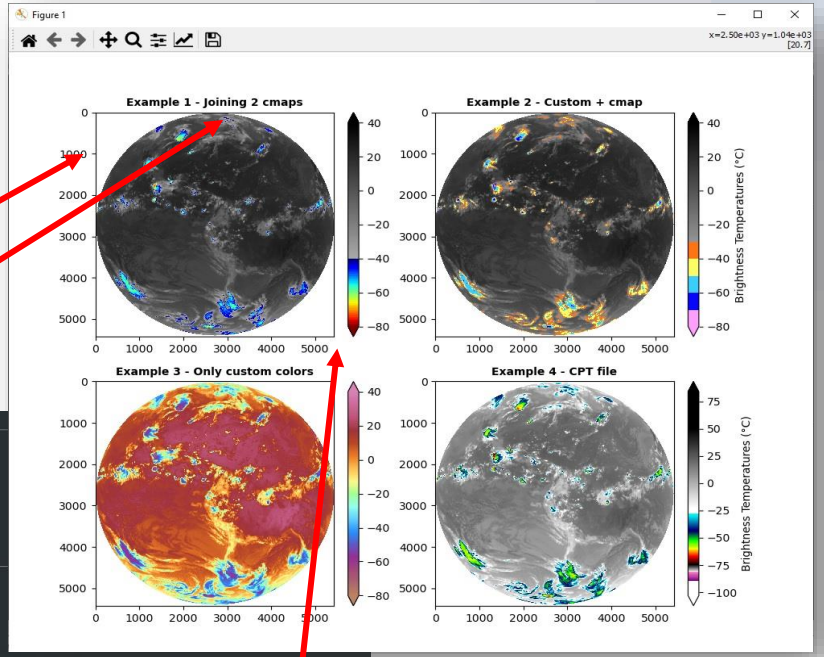
Script 8: Image Enhancements and Subplots

In this script we are going to see three new concepts:

- **Calling functions from external files**
- **Creation of custom color palettes**
- **Multiple Plots**

2 líneas x 2 columnas

```
74  #---------------------------------------------------------------------------
75
76  # Choose the plot size (width x height, in inches)
77  fig, axs = plt.subplots(2,2, figsize=(10,10)) # 2 rows x 2 columns
78
79  #---------------------------------------------------------------------------
80  # Plot 1 (first row, first column)
81
82  # Plot the image
83  img1 = axs[0,0].imshow(data, vmin=vmin1, vmax=vmax1, origin='upper', cmap=my_cmap1)
84
85  # Add a colorbar
86  plt.colorbar(img1, extend='both', orientation='vertical', pad=0.05, fraction=0.05, ax=axs[0,0])
87
88  # Add a title
89  axs[0,0].set_title('Example 1 - Joining 2 cmaps', fontweight='bold', fontsize=10, loc='center')
90  #---------------------------------------------------------------------------
```

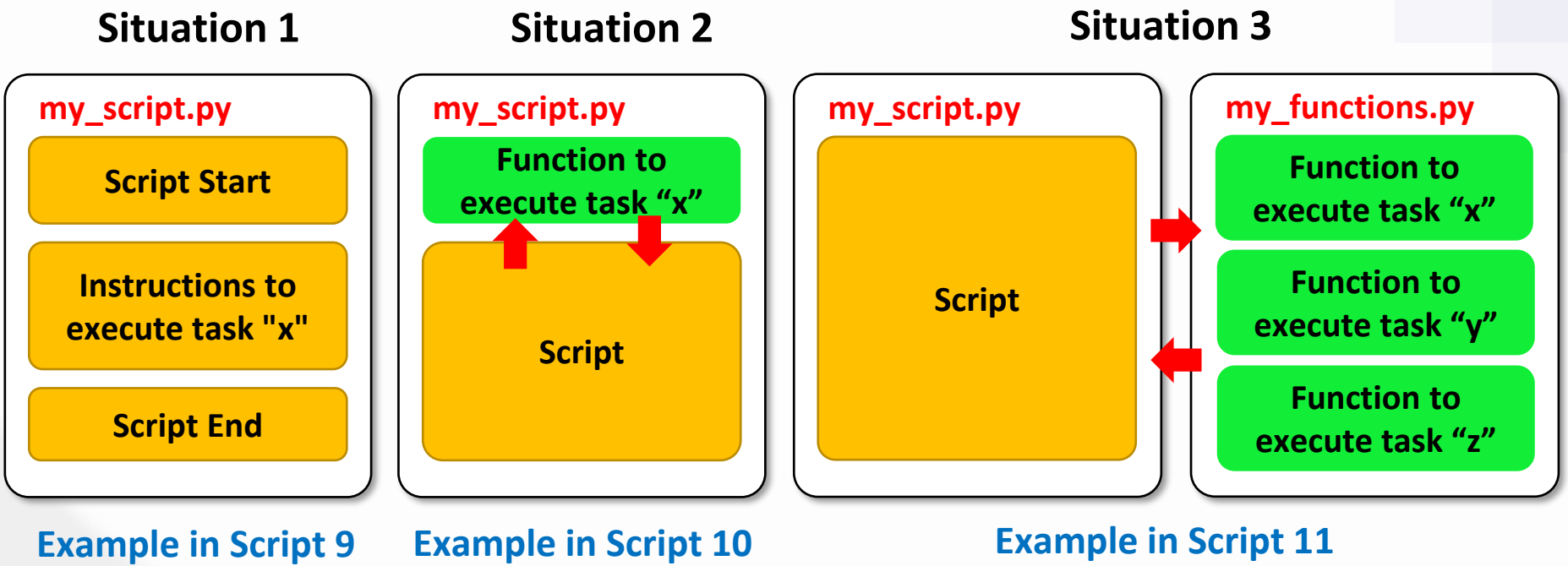


Note: there are several ways to create subplots, this is one of the examples

Part 2: More Advanced Concepts

PART I	PART II
Basic Plot / Reading Pixel Values	Downloading Data from the Cloud
Basic Operation / Colorbar / Title / Date	Functions
Overlaying Maps with Cartopy	Cropping a Full Disk Image
Reading the Metadata	Cropping a Full Disk Image and Creating an RGB Composite
Reading a Shapefile	Reprojection with GDAL
ABI + GLM (Basic Plot)	Level 2 Products (SST) and Data Quality Flags (DQF)
RGB Composites	Average and Accumulation
Custom Colormaps - Enhancing IR Channels	GLM Density and Heatmap

Functions: Improving Our Scripts



Example in Script 9

Example in Script 10

Example in Script 11

Script 9: Downloading Data from Amazon Using Scripts

```python
# Training: Python and GOES-R Imagery: Script 9 - Downloading data from AWS
#-------------------------------------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset           # Read / Write NetCDF4 files
import matplotlib.pyplot as plt       # Plotting library
from datetime import datetime         # Basic Dates and time types
import cartopy, cartopy.crs as ccrs   # Plot maps
import os                             # Miscellaneous operating system interfaces
import boto3                          # Amazon Web Services (AWS) SDK for Python
from botocore import UNSIGNED         # boto3 config
from botocore.config import Config    # boto3 config
#-------------------------------------------------------------------------------------------------------------
# Input and output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

#-------------------------------------------------------------------------------------------------------------
# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-CMIPF'
year = 2021
day_of_year = 37
hour = 17
min = 00
band = 1

# Initializes the S3 client
s3_client = boto3.client('s3', config=Config(signature_version=UNSIGNED))
#-------------------------------------------------------------------------------------------------------------
# File structure
prefix = f'{product_name}/{year}/{day_of_year:03.0f}/{hour:02.0f}/OR_{product_name}-M6C{band:02.0f}_G16_s{year}{day_of_year:03.0f}{hour:02.0f}{min:02.0f}'

# Seach for the file on the server
s3_result = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=prefix, Delimiter = "/")
#-------------------------------------------------------------------------------------------------------------
# Check if there are files available
if 'Contents' not in s3_result:
    # There are no files
    print("No files found for the date: ",year,day_of_year)
    quit()
else:
    # There are files
    for obj in s3_result['Contents']:
        key = obj['Key']
        # Print the file name
        print(key)
```

LIBRARIES

boto3

INPUT AND OUTPUT DIRECTORIES

https://noaa-goes16.s3.amazonaws.com/index.html

DESIRED DATA AND INITIALIZATION

CHECK IF THE DATA IS AVAILABLE

Script 9: Downloading Data from Amazon Using Scripts

```
# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-CMIPF'
year = 2021
day_of_year = 37
hour = 17
min = 00
band = 1

# Initializes the S3 client
s3_client = boto3.client('s3', config=Config(signature_version=UNSIGNED))
#-------------------------------------------------------------------------------------------------
# File structure
prefix = f'{product_name}/{year}/{day_of_year:03.0f}/{hour:02.0f}/OR_{product_name}-M6C{band:02.0f}_G16_s{year}{day_of_year:03.0f}{hour:02.0f}{min:02.0f}'
```

/ABI-L2-CMIPF/2019/198/12/OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc

```
33  # Seach for the file on the server
34  s3_result = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=prefix, Delimiter = "/")
35  #-------------------------------------------------------------------------------------------
```

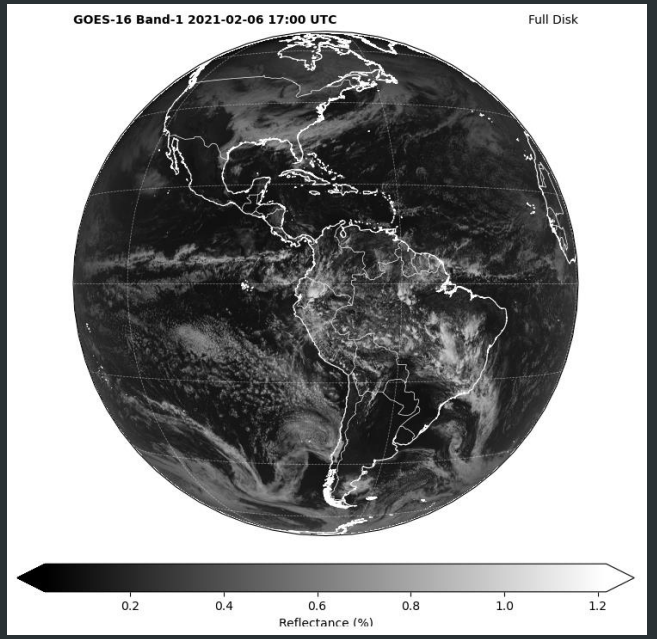
bucket_name = 'noaa-goes16'

Script 9: Downloading Data from Amazon Using Scripts

```
# Check if there are files available
if 'Contents' not in s3_result:
    # There are no files
    print("No files found for the date: ",year,day_of_year)
    exit()
else:
    # There are files
    for obj in s3_result['Contents']:
        key = obj['Key']
        # Print the file name
        print(key)
        file_name = key.split('/')[-1].split('.')[0]

        # Download the file
        if not os.path.exists(f'{input}/{file_name}.nc'):
            s3_client.download_file(bucket_name, key, f'{input}/{file_name}.nc')

        # If the file exists
        if (os.path.exists(f'{input}/{file_name}.nc')):

            # Open the GOES-R image
            file = Dataset(f'{input}/{file_name}.nc')

            # Get the pixel values
            data = file.variables['CMI'][:]
            #---------------------------------------------------------------------------------------------------------
            # Choose the plot size (width x height, in inches)
            plt.figure(figsize=(10,10))

            # Use the Geostationary projection in cartopy
            ax = plt.axes(projection=ccrs.Geostationary(central_longitude=-75.0, satellite_height=35786023.0))
            img_extent = (-5434894.67527,5434894.67527,-5434894.67527,5434894.67527)

            # Add coastlines, borders and gridlines
            ax.coastlines(resolution='10m', color='white', linewidth=0.8)
            ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.5)
            ax.gridlines(color='white', alpha=0.5, linestyle='--', linewidth=0.5)

            # Define the color scale based on the channel
            if band <= 6:
                colormap = "gray"   # Black to white for visible channels
                prodname = "Reflectance (%)"
            else:
                colormap = "gray_r" # White to black for IR channels
                prodname = "Brightness Temperatures (°C)"

            # Plot the image
            img = ax.imshow(data, origin='upper', extent=img_extent, cmap=colormap)

            # Add a colorbar
            plt.colorbar(img, label=prodname, extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

            # Extract the date
            date = (datetime.strptime(file.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))

            # Add a title
            plt.title('GOES-16 Band-' + str(band) + ' ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
            plt.title('Full Disk', fontsize=10, loc='right')
            #---------------------------------------------------------------------------------------------------------
            # Save the image
            plt.savefig(f'{output}/{file_name}.png')

            # Show the image
            plt.show()
```

DATA NOT AVAILABLE: END THE SCRIPT

DATA AVAILABLE: PLOT (eg: SCRIPT 3)



Script 9: Downloading Data from Amazon Using Scripts

```
35  #----------------------------------------------------------------------------------------------------------
36  # Check if there are files available
37  if 'Contents' not in s3_result:
38      # There are no files
39      print("No files found for the date: ",year,day_of_year)
40      quit()
41  else:
42      # There are files
43      for obj in s3_result['Contents']:
44          key = obj['Key']
45          # Print the file name
46          print(key)
47          file_name = key.split('/')[-1].split('.')[0]
48
49          # Download the file
50          if not os.path.exists(f'{input}/{file_name}.nc'):
51              s3_client.download_file(bucket_name, key, f'{input}/{file_name}.nc')
52
53          # If the file exists
54          if (os.path.exists(f'{input}/{file_name}.nc')):
55
56              # Open the GOES-R image
57              file = Dataset(f'{input}/{file_name}.nc')
58
59              # Get the pixel values
60              data = file.variables['CMI'][:]
61              #---------------------------------------------------------------------------------------------
62              # Choose the plot size (width x height, in inches)
63              plt.figure(figsize=(10,10))
```

For each file found...

File name found

If the file does not exist in the computer, download the file

If the file exists, process the file

Script 10: Downloading Data from Amazon Using Scripts (Function)

```python
# Training: Python and GOES-R Imagery: Script 10 - Downloading data from AWS (function)
#
# Required modules
from netCDF4 import Dataset           # Read / Write NetCDF4 files
import matplotlib.pyplot as plt       # Plotting library
from datetime import datetime         # Basic Dates and time types
import cartopy, cartopy.crs as ccrs   # Plot maps
import os                             # Miscellaneous operating system interfaces
import boto3                          # Amazon Web Services (AWS) SDK for Python
from botocore import UNSIGNED         # boto3 config
from botocore.config import Config    # boto3 config
#
# Function to download files
def download_file(s3_client, prefix):
    # Seach for the file on the server
    s3_result = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=prefix, Delimiter = "/")
    # Check if there are files available
    if 'Contents' not in s3_result:
        # There are no files
        print("No files found for the date: ",year,day_of_year)
        quit()
    else:
        # There are files
        for obj in s3_result['Contents']:
            # Print the file name
            key = obj['Key']
            print(key)
            file_name = key.split('/')[-1].split('.')[0]

            # Download the file
            if not os.path.exists(f'{input}/{file_name}.nc'):
                s3_client.download_file(bucket_name, key, f'{input}/{file_name}.nc')

    return file_name
#-----------------------------------------------------------------------------------------
```

LIBRARIES

FUNCTION TO DOWNLOAD DATA

INPUT:

“s3_client” and “prefix”

Execute the download

OUTPUT:

“file_name”

Script 10: Downloading Data from Amazon Using Scripts (Function)

```
35  #-------------------------------------------------------------------------------------------------------
36  # Input and output directories
37  input = "Samples"; os.makedirs(input, exist_ok=True)
38  output = "Output"; os.makedirs(output, exist_ok=True)
39
40  # AMAZON repository information
41  # https://noaa-goes16.s3.amazonaws.com/index.html
42  bucket_name = 'noaa-goes16'
43  product_name = 'ABI-L2-CMIPF'
44  year = 2021
45  day_of_year = 37
46  hour = 18
47  min = 00
48  band = 13
49
50  # Initializes the S3 client
51  s3_client = boto3.client('s3', config=Config(signature_version=UNSIGNED))
52  #-------------------------------------------------------------------------------------------------------
53  # File structure
54  prefix = f'{product_name}/{year}/{day_of_year:03.0f}/{hour:02.0f}/OR_{product_name}-M6C{band:02.0f}_G16_s{year}{day_of_year:03.0f}{hour:02.0f}{min:02.0f}'
55
56  # Download the file
57  file_name = download_file(s3_client, prefix)
58  #-------------------------------------------------------------------------------------------------------
59  # Open the GOES-R image
60  file = Dataset(f'{input}/{file_name}.nc')
61
62  # Get the pixel values
63  data = file.variables['CMI'][:]
64  #-------------------------------------------------------------------------------------------------------
65  # Choose the plot size (width x height, in inches)
66  plt.figure(figsize=(10,10))
```

DESIRED DATA AND INITIALIZATION

Function calling

START THE PROCESSING

Script 11: Downloading Data from Amazon Using Scripts (utilities.py)

```python
# Training: Python and GOES-R Imagery: Script 11 - Downloading data from AWS import function
#-------------------------------------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset                # Read / Write NetCDF4 files
import matplotlib.pyplot as plt            # Plotting library
from datetime import datetime              # Basic Dates and time types
import cartopy, cartopy.crs as ccrs        # Plot maps
import os                                  # Miscellaneous operating system interfaces
from utilities import download_CMI         # Our own utilities
#-------------------------------------------------------------------------------------------------------------
# Input and output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-CMIPF'
yyyymmddhhmn = '202102181800'
band = '13'

# Download the file
file_name = download_CMI(yyyymmddhhmn, band, input)
#-------------------------------------------------------------------------------------------------------------
# Open the GOES-R image
file = Dataset(f'{input}/{file_name}.nc')

# Get the pixel values
data = file.variables['CMI'][:]
#-------------------------------------------------------------------------------------------------------------
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(10,10))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.Geostationary(central_longitude=-75.0, satellite_height=35786023.0))
img_extent = (-5434894.67527,5434894.67527,-5434894.67527,5434894.67527)
```

LIBRARIES

Importing the “download_CMI” function

DESIRED DATA

call of the function that is in the “utilities.py” file

START THE PROCESSING

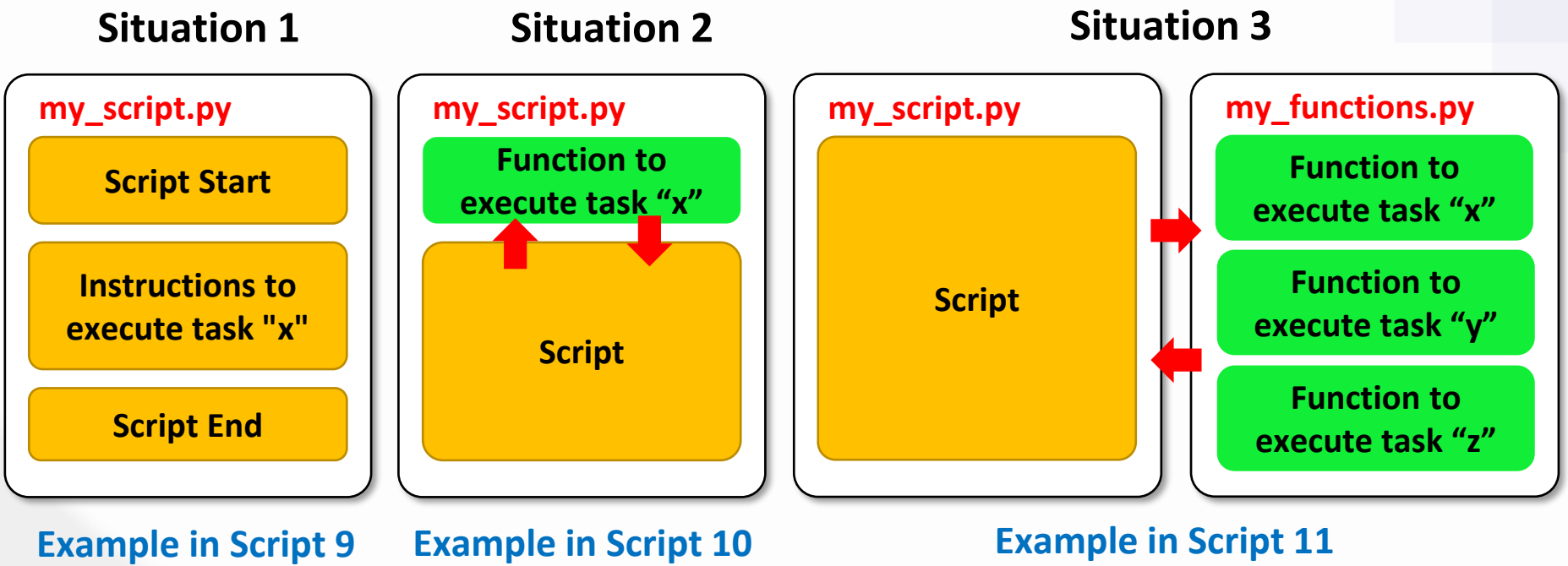
Script 11: Downloading Data from Amazon Using Scripts (utilities.py)

```
23  # Download the file
24  file_name = download_CMI(yyyymmddhhmn, band, input)
25
```

Function inside the utilities.py file

```
90  #---------------------------------------------------------------------------
91  def download_CMI(yyyymmddhhmn, band, path_dest):
92
93      os.makedirs(path_dest, exist_ok=True)
94
95      year = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%Y')
96      day_of_year = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%j')
97      hour = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%H')
98      min = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%M')
99
100     # AMAZON repository information
101     # https://noaa-goes16.s3.amazonaws.com/index.html
102     bucket_name = 'noaa-goes16'
103     product_name = 'ABI-L2-CMIPF'
104
105     # Initializes the S3 client
106     s3_client = boto3.client('s3', config=Config(signature_version=UNSIGNED))
107     #---------------------------------------------------------------------------
108     # File structure
109     prefix = f'{product_name}/{year}/{day_of_year}/{hour}/OR_{product_name}-M6C{int(band):02
110
111     # Seach for the file on the server
112     s3_result = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=prefix, Delimiter = "/"
113
114     #---------------------------------------------------------------------------
115     # Check if there are files available
116     if 'Contents' not in s3_result:
117       # There are no files
118       print(f'No files found for the date: {yyyymmddhhmn}, Band-{band}')
119       return -1
120     else:
121       # There are files
122       for obj in s3_result['Contents']:
123         key = obj['Key']
124         # Print the file name
125         file_name = key.split('/')[-1].split('.')[0]
126
127         # Download the file
128         if os.path.exists(f'{path_dest}/{file_name}.nc'):
129           print(f'File {path_dest}/{file_name}.nc exists')
130         else:
131           print(f'Downloading file {path_dest}/{file_name}.nc')
132           s3_client.download_file(bucket_name, key, f'{path_dest}/{file_name}.nc')
133     return f'{file_name}'
134
135 #---------------------------------------------------------------------------
```

Functions: Improving Our Scripts



Example in Script 9

Example in Script 10

Example in Script 11

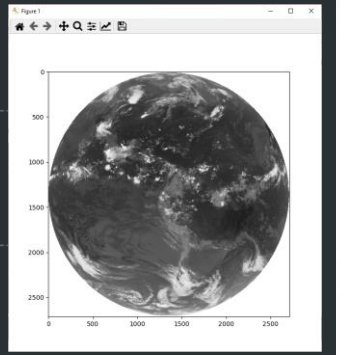
Script 12: "Cropping" a Full Disk Image

QUESTION: How many lines and columns does a 2 km Full Disk image have?

Let's go back to Script 1! Let's add the following command:

```
# Print the dimension of our data
print("Array dimension: ", data.shape)
```

```
# Training: Python and GOES-R Imagery: Script 1 - Basic Plot / Extracting Pixel Values
#-----------------------------------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset      # Read / Write NetCDF4 files
import matplotlib.pyplot as plt  # Plotting library
#-----------------------------------------------------------------------------------------------------------
# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values
data = file.variables['CMI'][:]

# Print the dimension of our data
print("Array dimension: ", data.shape)
#-----------------------------------------------------------------------------------------------------------
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Plot the image
plt.imshow(data, vmin=193, vmax=313, cmap='Greys')
#-----------------------------------------------------------------------------------------------------------
# Save the image
plt.savefig('Image_01.png')

# Show the image
plt.show()
```



When running, we will see the following in the terminal:

```
(workshop) D:\VLAB\Python>python Script_01.py
Data shape:  (5424, 5424)
```

Therefore:

2 km Bands: 5424 x 5424
1 km Bands: 10848 x 10848
0.5 km Bands: 21696 x 21696

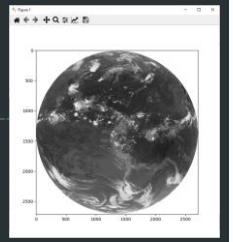
Script 12: “Cropping” a Full Disk Image

QUESTION: How to "decrease" the size of our data?

Alternative 1: Read the data at every “x” pixels

```
# Get the pixel values
data = file.variables['CMI'][:,:][::2 ,::2]
```

```
# Training: Python and GOES-R Imagery: Script 1 - Basic Plot / Extracting Pixel Values
#-------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset      # Read / Write NetCDF4 files
import matplotlib.pyplot as plt  # Plotting library
#-------------------------------------------------------------------------------
# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values
data = file.variables['CMI'][:,:][::2 ,::2]

# Print the dimension of our data
print("Array dimension: ", data.shape)
#-------------------------------------------------------------------------------
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Plot the image
plt.imshow(data, vmin=193, vmax=313, cmap='Greys')
#-------------------------------------------------------------------------------
# Save the image
plt.savefig('Image_01.png')

# Show the image
plt.show()
```

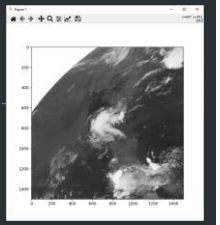


```
(workshop) D:\VLAB\Python>python Script_01.py
Array dimension:  (2712, 2712)
```

Alternative 2: Read only one region of the full disk

```
# Get the pixel values
data = file.variables['CMI'] [500:2000,500:2000]
```

```
# Training: Python and GOES-R Imagery: Script 1 - Basic Plot / Extracting Pixel Values
#-------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset      # Read / Write NetCDF4 files
import matplotlib.pyplot as plt  # Plotting library
#-------------------------------------------------------------------------------
# Open the GOES-R image
# Download files at this link: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi
file = Dataset("OR_ABI-L2-CMIPF-M6C13_G16_s20191981200396_e20191981210116_c20191981210189.nc")

# Get the pixel values
data = file.variables['CMI'][500:2000,500:2000]

# Print the dimension of our data
print("Array dimension: ", data.shape)
#-------------------------------------------------------------------------------
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(7,7))

# Plot the image
plt.imshow(data, vmin=193, vmax=313, cmap='Greys')
#-------------------------------------------------------------------------------
# Save the image
plt.savefig('Image_01.png')

# Show the image
plt.show()
```



```
(workshop) D:\VLAB\Python>python Script_01.py
Array dimension:  (1500, 1500)
```

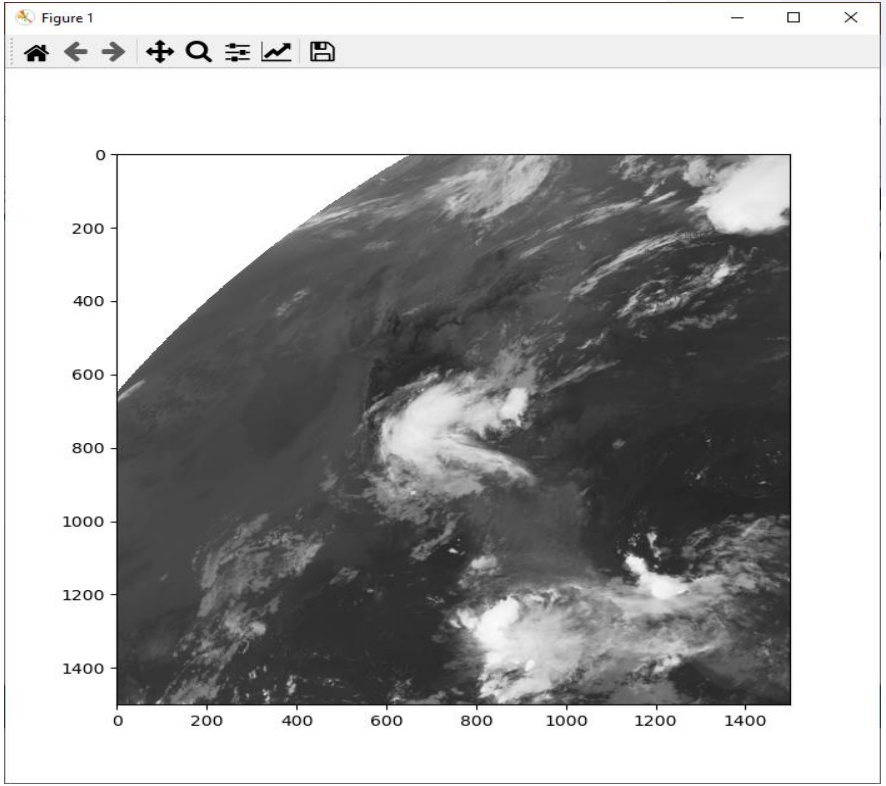
Script 12: “Cropping” a Full Disk Image

QUESTION: How to "decrease" the size of our data?

Alternative 2: Read only a region of the full disk

```
# Get the pixel values
data = file.variables['CMI'] [500:2000,500:2000]
```

```
(workshop) D:\VLAB\Python>python Script_01.py
Array dimension:  (1500, 1500)
```



Conclusion: It is not very intuitive to “crop” the Full Disk using lines and columns. How to crop it using coordinates?

Script 12: "Cropping" a Full Disk Image

In the script utilities.py we have the necessary functions!

```python
# Training: Python and GOES-R Imagery: Script 12 - Cropping the Full Disk
#-------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset                 # Read / Write NetCDF4 files
import matplotlib.pyplot as plt             # Plotting library
from datetime import datetime               # Basic Dates and time types
import cartopy, cartopy.crs as ccrs         # Plot maps
import os                                   # Miscellaneous operating system interfaces
from utilities import download_CMI          # Our own utilities
from utilities import geo2grid, convertExtent2GOESProjection        # Our own utilities
#
# Input and output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# Desired extent
extent = [-64.0, -36.0, -40.0, -15.0] # Min lon, Max lon, Min lat, Max lat

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-CMIPF'
yyyymmddhhmn = '202102181800'
band = '13'

# Download the file
file_name = download_CMI(yyyymmddhhmn, band, input)

#-------------------------------------------------------------------------------
# Open the GOES-R image
file = Dataset(f'{input}/{file_name}.nc')

# Convert lat/lon to grid-coordinates
lly, llx = geo2grid(extent[1], extent[0], file)
ury, urx = geo2grid(extent[3], extent[2], file)

# Get the pixel values
data = file.variables['CMI'][ury:lly, llx:urx]
#
```

Importing the geo2grid and convertExtent2GOESProjection functions

Defining the region of interest

Downloading the data (eg.: script 11)

Calling the function that calculates rows and columns

Functions inside the utilities.py file

```python
#
# Functions to convert lat / lon extent to array indices
def geo2grid(lat, lon, nc):

    # Apply scale and offset
    xscale, xoffset = nc.variables['x'].scale_factor, nc.variables['x'].add_offset
    yscale, yoffset = nc.variables['y'].scale_factor, nc.variables['y'].add_offset

    x, y = latlon2xy(lat, lon)
    col = (x - xoffset)/xscale
    lin = (y - yoffset)/yscale
    return int(lin), int(col)

def latlon2xy(lat, lon):
    # goes_imagery_projection:semi_major_axis
    req = 6378137 # meters
    #  goes_imagery_projection:inverse_flattening
    invf = 298.257222096
    # goes_imagery_projection:semi_minor_axis
    rpol = 6356752.31414 # meters
    e = 0.0818191910435
    # goes_imagery_projection:perspective_point_height + goes_imagery_projection:semi_major_axis
    H = 42164160 # meters
    # goes_imagery_projection: longitude_of_projection_origin
    lambda0 = -1.308996939

    # Convert to radians
    latRad = lat * (math.pi/180)
    lonRad = lon * (math.pi/180)

    # (1) geocentric latitude
    Phi_c = math.atan(((rpol * rpol)/(req * req)) * math.tan(latRad))
    # (2) geocentric distance to the point on the ellipsoid
    rc = rpol/(math.sqrt(1 - ((e * e) * (math.cos(Phi_c) * math.cos(Phi_c)))))
    # (3) sx
    sx = H - (rc * math.cos(Phi_c) * math.cos(lonRad - lambda0))
    # (4) sy
    sy = -rc * math.cos(Phi_c) * math.sin(lonRad - lambda0)
    # (5)
    sz = rc * math.sin(Phi_c)

    # x,y
    x = math.asin((-sy)/math.sqrt((sx*sx) + (sy*sy) + (sz*sz)))
    y = math.atan(sz/sx)

    return x, y

# Function to convert lat / lon extent to GOES-16 extents
def convertExtent2GOESProjection(extent):
    # GOES-16 viewing point (satellite position) height above the earth
    GOES16_HEIGHT = 35786023.0
    # GOES-16 longitude position
    GOES16_LONGITUDE = -75.0

    a, b = latlon2xy(extent[1], extent[0])
    c, d = latlon2xy(extent[3], extent[2])
    return (a * GOES16_HEIGHT, c * GOES16_HEIGHT, b * GOES16_HEIGHT, d * GOES16_HEIGHT)
```

Script 12: "Cropping" a Full Disk Image

In the script utilities.py we have the necessary functions!

```python
# Training: Python and GOES-R Imagery: Script 12 - Cropping the Full Disk
#-----------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset                 # Read / Write NetCDF4 files
import matplotlib.pyplot as plt             # Plotting library
from datetime import datetime               # Basic Dates and time types
import cartopy, cartopy.crs as ccrs         # Plot maps
import os                                   # Miscellaneous operating system interfaces
from utilities import download_CMI          # Our own utilities
from utilities import geo2grid, convertExtent2GOESProjection      # Our own utilities
#-----------------------------------------------------------------------------
# Input and output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# Desired extent
extent = [-64.0, -36.0, -40.0, -15.0] # Min lon, Max lon, Min lat, Max lat

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-CMIPF'
yyyymmddhhmn = '202102181800'
band = '13'

# Download the file
file_name = download_CMI(yyyymmddhhmn, band, input)

#-----------------------------------------------------------------------------
# Open the GOES-R image
file = Dataset(f'{input}/{file_name}.nc')

# Convert lat/lon to grid-coordinates
lly, llx = geo2grid(extent[1], extent[0], file)
ury, urx = geo2grid(extent[3], extent[2], file)

# Get the pixel values
data = file.variables['CMI'][ury:lly, llx:urx]
#-----------------------------------------------------------------------------
```

Input:
desired coordinates and file

Output: lines and columns

Funciones dentro del Functions inside the utilities.py file

```python
#-----------------------------------------------------------------------------
# Functions to convert lat / lon extent to array indices
def geo2grid(lat, lon, nc):

    # Apply scale and offset
    xscale, xoffset = nc.variables['x'].scale_factor, nc.variables['x'].add_offset
    yscale, yoffset = nc.variables['y'].scale_factor, nc.variables['y'].add_offset

    x, y = latlon2xy(lat, lon)
    col = (x - xoffset)/xscale
    lin = (y - yoffset)/yscale
    return int(lin), int(col)

def latlon2xy(lat, lon):
    # goes_imagery_projection:semi_major_axis
    req = 6378137 # meters
    #  goes_imagery_projection:inverse_flattening
    invf = 298.257222096
    # goes_imagery_projection:semi_minor_axis
    rpol = 6356752.31414 # meters
    e = 0.0818191910435
    # goes_imagery_projection:perspective_point_height + goes_imagery_projection:semi_major_axis
    H = 42164160 # meters
    # goes_imagery_projection: longitude_of_projection_origin
    lambda0 = -1.308996939

    # Convert to radians
    latRad = lat * (math.pi/180)
    lonRad = lon * (math.pi/180)

    # (1) geocentric latitude
    Phi_c = math.atan(((rpol * rpol)/(req * req)) * math.tan(latRad))
    # (2) geocentric distance to the point on the ellipsoid
    rc = rpol/(math.sqrt(1 - ((e * e) * (math.cos(Phi_c) * math.cos(Phi_c)))))
    # (3) sx
    sx = H - (rc * math.cos(Phi_c) * math.cos(lonRad - lambda0))
    # (4) sy
    sy = -rc * math.cos(Phi_c) * math.sin(lonRad - lambda0)
    # (5)
    sz = rc * math.sin(Phi_c)

    # x,y
    x = math.asin((-sy)/math.sqrt((sx*sx) + (sy*sy) + (sz*sz)))
    y = math.atan(sz/sx)

    return x, y

# Function to convert lat / lon extent to GOES-16 extents
def convertExtent2GOESProjection(extent):
    # GOES-16 viewing point (satellite position) height above the earth
    GOES16_HEIGHT = 35786023.0
    # GOES-16 longitude position
    GOES16_LONGITUDE = -75.0

    a, b = latlon2xy(extent[1], extent[0])
    c, d = latlon2xy(extent[3], extent[2])
    return (a * GOES16_HEIGHT, c * GOES16_HEIGHT, b * GOES16_HEIGHT, d * GOES16_HEIGHT)
```

Script 12: “Cropping” a Full Disk Image

```
# Training: Python and GOES-R Imagery: Script 12 - Cropping the Full Disk

# Required modules
from netCDF4 import Dataset                # Read / Write NetCDF4 files
import matplotlib.pyplot as plt            # Plotting library
from datetime import datetime              # Basic Dates and time types
import cartopy, cartopy.crs as ccrs        # Plot maps
import os                                  # Miscellaneous operating system interfaces
from utilities import download_CMI         # Our own utilities
from utilities import geo2grid, convertExtent2GOESProjection      # Our own utilities

input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# Desired extent
extent = [-64.0, -36.0, -40.0, -15.8] # Min Lon, Max Lon, Min Lat, Max Lat

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-CMIPF'
yyyymmddhhmn = '202102181800'
band = '13'

# Download the file
file_name = download_CMI(yyyymmddhhmn, band, input)

#--------------------------------------------------------------------------------------------------
# Open the GOES-R image
file = Dataset(f'{input}/{file_name}.nc')

# Convert lat/lon to grid coordinates
lly, llx = geo2grid(extent[1], extent[0], file)
ury, urx = geo2grid(extent[3], extent[2], file)

# Get the pixel values
data = file.variables['CMI'][ury:lly, llx:urx]
#--------------------------------------------------------------------------------------------------
# Compute data extent in GOES projection-coordinates
img_extent = convertExtent2GOESProjection(extent)

#--------------------------------------------------------------------------------------------------
plt.figure(figsize=(10,10))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.Geostationary(central_longitude=-75.0, satellite_height=35786023.0))

# Define the color scale based on the channel
colormap = "gray_r" # White to black for IR channels

# Plot the image
img = ax.imshow(data, origin='upper', extent=img_extent, cmap=colormap)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='white', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.5)
ax.gridlines(color='white', alpha=0.5, linestyle='--', linewidth=0.5)

# Add a colorbar
plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(file.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 Band 13 ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Req.: ' + str(extent) , fontsize=10, loc='right')
#--------------------------------------------------------------------------------------------------
# Save the image
plt.savefig(f'{output}/{file_name}.png', bbox_inches='tight', pad_inches=0, dpi=300)

# Show the image
plt.show()
```

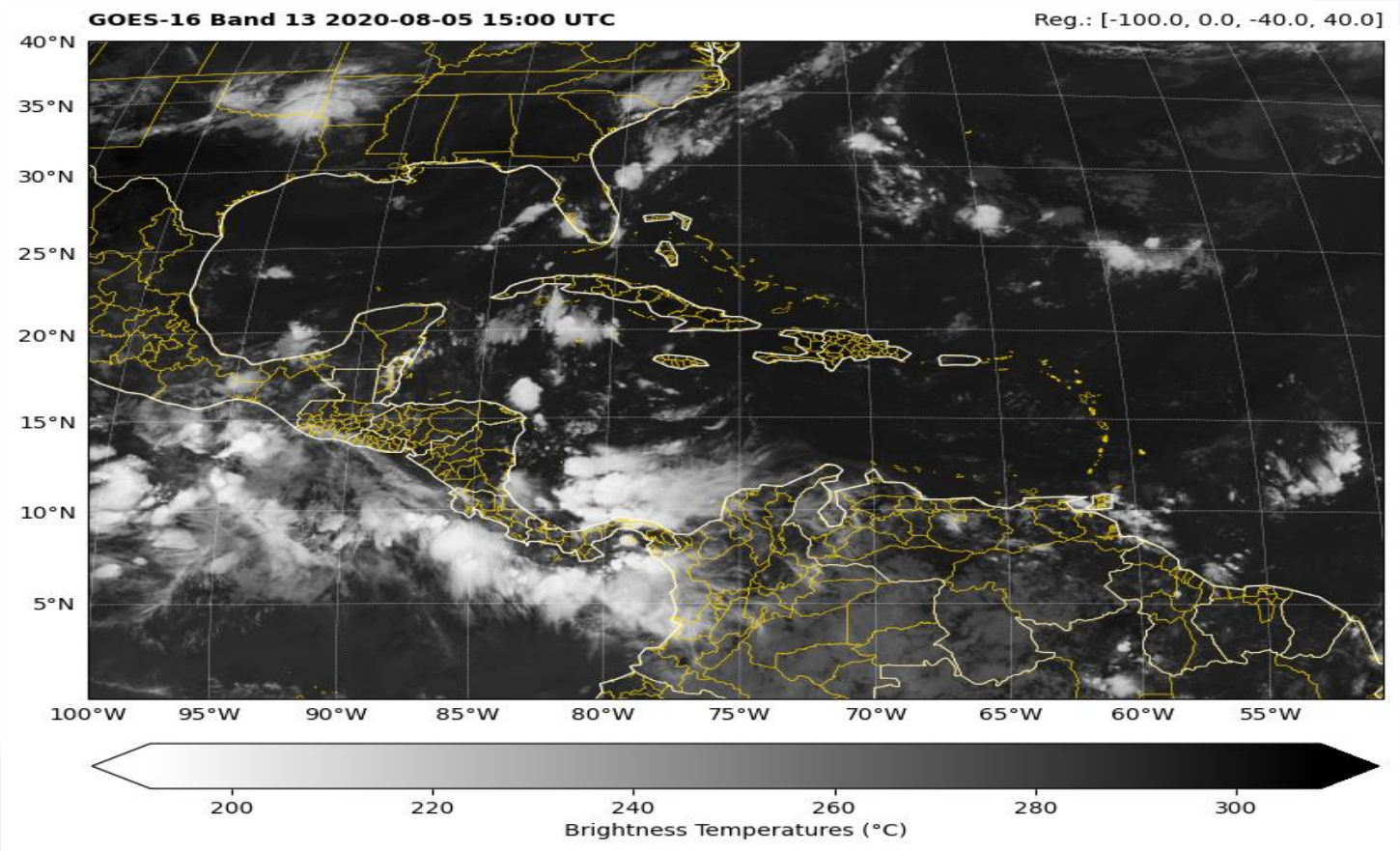
LIBRARIES AND FUNCTIONS

DOWNLOAD THE DATA

CROPPING

PLOT AND IMAGE GENERATION

Script 12: “Cropping” a Full Disk Image



Script 12 (Extra): Image Size = Data Size

```
46  # Choose the plot size (data size, in pixels)
47  # Print the dimension of our data
48  print("Array dimension: ", data.shape)
49  dpi = 150
50  fig = plt.figure(figsize=(data.shape[1]/dpi, data.shape[0]/dpi), dpi=dpi)
51
52  # Use the Geostationary projection in cartopy
53  ax = plt.axes([0, 0, 1, 1], projection=ccrs.Geostationary(central_longitude=-75.0, satellite_he
54
55  # Define the color scale based on the channel
56  colormap = "gray_r" # White to black for IR channels
57
58  # Plot the image
59  img = ax.imshow(data, origin='upper', extent=img_extent, cmap=colormap)
60
61  # Add coastlines, borders and gridlines
62  ax.coastlines(resolution='10m', color='white', linewidth=0.8)
63  ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.5)
64  ax.gridlines(color='white', alpha=0.5, linestyle='--', linewidth=0.5)
65
66  # Add a colorbar
67  #plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal
68
69  # Extract the date
70  date = (datetime.strptime(file.time_coverage_start, '%Y-%m-%dT%H:%M:%S.%fZ'))
71
72  # Add a title
73  #plt.title('GOES-16 Band 13 ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fo
74  #plt.title('Reg.: ' + str(e                    rsize=10, loc='right')
75  #-------------------------------------------------------------------------------------------
76  # Save the image
77  plt.savefig(f'{output}/{file_name}.png')#, bbox_inches='tight', pad_inches=0, dpi=300)
78
79  # Show the image
80  plt.show()
```

No aparece

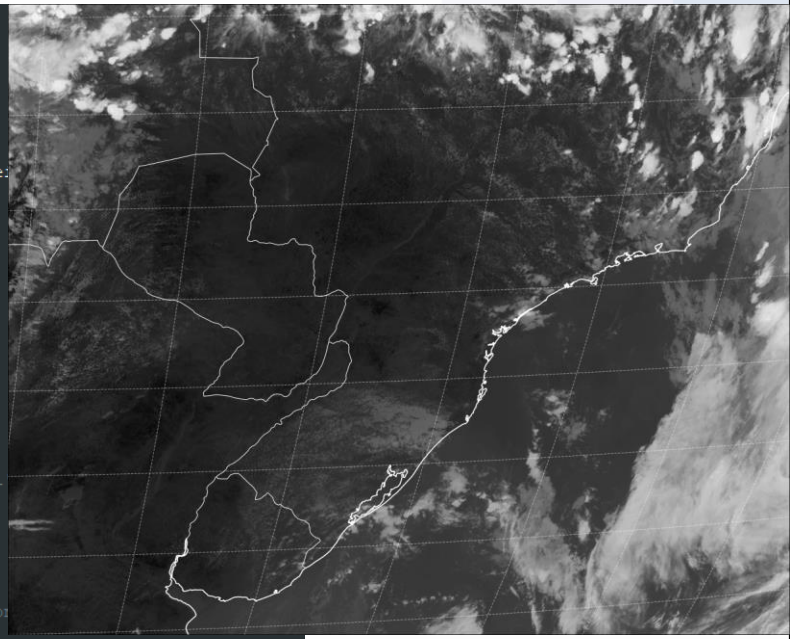
Add

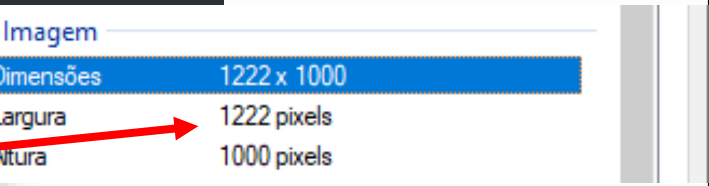
$$\frac{\text{pixels} * \text{dpi}}{\text{dpi}}$$

Remove

Remove

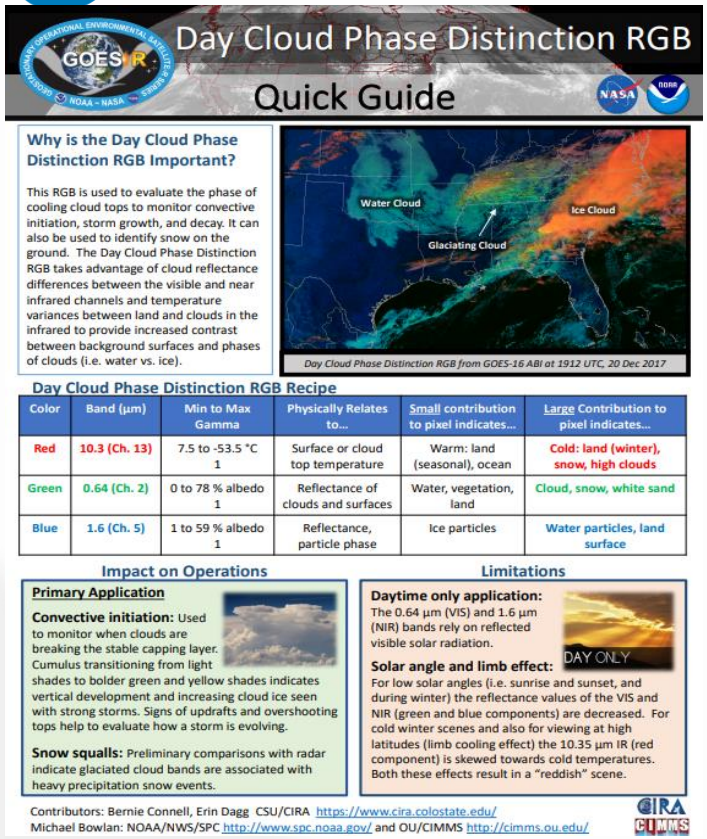
Remove






```
(workshop) D:\VLAB\Python>python Script_12.py
File Samples/OR_ABI-L2-CMIPF-M6C13_G16_s20210491800058_e20210491809378_c20210491809458.nc exists
Array dimension:  (1000, 1222)
```

Script 13: Cropping a Full Disk Image and Creating an RGB Composite



Day Cloud Phase Distinction RGB Recipe

Color	Band (µm)	Min to Max Gamma	Physically Relates to...
Red	10.3 (Ch. 13)	7.5 to -53.5 °C 1	Surface or cloud top temperature
Green	0.64 (Ch. 2)	0 to 78 % albedo 1	Reflectance of clouds and surfaces
Blue	1.6 (Ch. 5)	1 to 59 % albedo 1	Reflectance, particle phase

https://rammb.cira.colostate.edu/training/visit/quick_guides/QuickGuide_DayCloudPhaseDistinction_final_v2.pdf

Script 13: Cropping a Full Disk Image and Creating an RGB Composite

```python
# Desired extent
extent = [-64.0, -36.0, -40.0, -15.0] # Min lon, Max lon, Min lat, Max lat
```

Defining the desired region

```python
# Download the necessary bands from AWS
file_ch13 = download_CMI(yyyymmddhhmn, 13, input)
file_ch02 = download_CMI(yyyymmddhhmn, 2, input)
file_ch05 = download_CMI(yyyymmddhhmn, 5, input)

# Open the GOES-R images
file_ch13 = Dataset(f'{input}/{file_ch13}.nc')
file_ch02 = Dataset(f'{input}/{file_ch02}.nc')
file_ch05 = Dataset(f'{input}/{file_ch05}.nc')
```

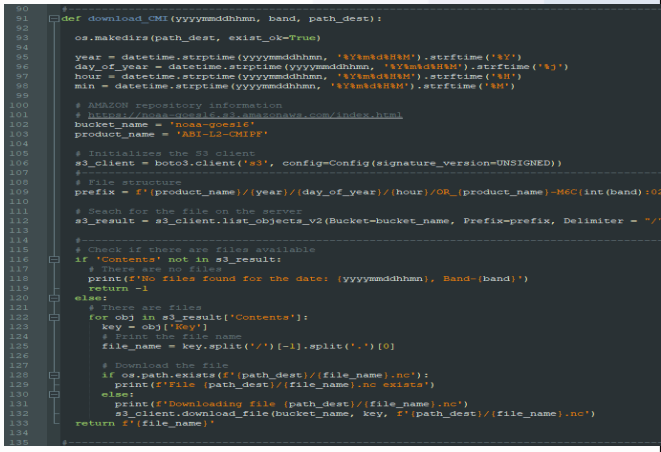
Download the necessary files

```python
# Convert lat/lon to grid-coordinates
lly, llx = geo2grid(extent[1], extent[0], file_ch13)
ury, urx = geo2grid(extent[3], extent[2], file_ch13)

# Get the pixel values
data_ch13 = file_ch13.variables['CMI'][ury:lly, llx:urx] - 273.15

# Convert lat/lon to grid-coordinates
lly, llx = geo2grid(extent[1], extent[0], file_ch02)
ury, urx = geo2grid(extent[3], extent[2], file_ch02)

# Get the pixel values
data_ch02 = file_ch02.variables['CMI'][ury:lly, llx:urx][::4 ,::4]

# Convert lat/lon to grid-coordinates
lly, llx = geo2grid(extent[1], extent[0], file_ch05)
ury, urx = geo2grid(extent[3], extent[2], file_ch05)

# Get the pixel values
data_ch05 = file_ch05.variables['CMI'][ury:lly, llx:urx][::2 ,::2]
```

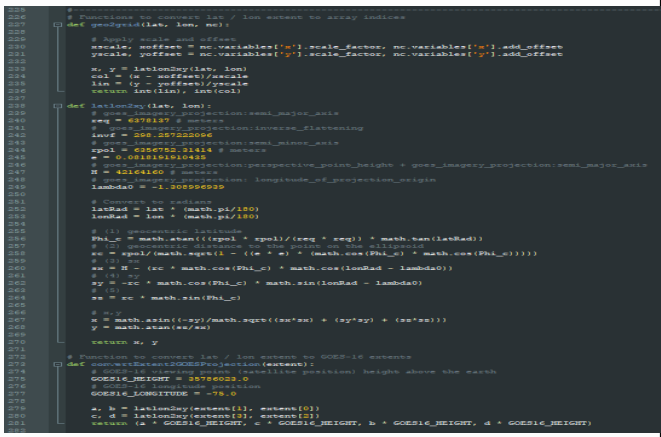
Definition of lines and columns

```python
# Make the arrays equal size
data_ch02 = data_ch02[0:data_ch13.shape[0], 0:data_ch13.shape[1]]
data_ch05 = data_ch05[0:data_ch13.shape[0], 0:data_ch13.shape[1]]

# Compute data-extent in GOES projection-coordinates
img_extent = convertExtent2GOESProjection(extent)
```

Equals the dimension of the data



```python
def download_CMI(yyyymmddhhmn, band, path_dest):

    os.makedirs(path_dest, exist_ok=True)

    year = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%Y')
    day_of_year = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%j')
    hour = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%H')
    min = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%M')

    # AMAZON repository information
    # https://noaa-goes16.s3.amazonaws.com/index.html
    bucket_name = 'noaa-goes16'
    product_name = 'ABI-L2-CMIPF'

    # Initializes the S3 client
    s3_client = boto3.client('s3', config=Config(signature_version=UNSIGNED))

    # File structure
    prefix = f'{product_name}/{year}/{day_of_year}/{hour}/OR_{product_name}-M6C{int(band):02

    # Search for the file on the server
    s3_result = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=prefix, Delimiter = "/

    # Check if there are files available
    if 'Contents' not in s3_result:
        # There are no files
        print(f'No files found for the date: {yyyymmddhhmn}, Band={band}')
        return -1
    else:
        # There are files
        for obj in s3_result['Contents']:
            key = obj['Key']
            # Print the file name
            file_name = key.split('/')[-1].split('.')[0]

            # Download the file
            if os.path.exists(f'{path_dest}/{file_name}.nc'):
                print(f'File {path_dest}/{file_name}.nc exists')
            else:
                print(f'Downloading file {path_dest}/{file_name}.nc')
                s3_client.download_file(bucket_name, key, f'{path_dest}/{file_name}.nc')
    return f'{file_name}'
```



```python
# Functions to convert lat / lon extent to array indices
def geo2grid(lat, lon, nc):

    # Apply scale and offset
    xscale, xoffset = nc.variables['x'].scale_factor, nc.variables['x'].add_offset
    yscale, yoffset = nc.variables['y'].scale_factor, nc.variables['y'].add_offset

    x, y = latlon2xy(lat, lon)
    col = (x - xoffset)/xscale
    lin = (y - yoffset)/yscale
    return int(lin), int(col)

def latlon2xy(lat, lon):
    # goes_imagery_projection:semi_major_axis
    req = 6378137 # meters
    #  goes_imagery_projection:inverse_flattening
    invf = 298.2572220960
    # goes_imagery_projection:semi_minor_axis
    rpol = 6356752.31414 # meters
    e = 0.0818191910435
    # goes_imagery_projection:perspective_point_height + goes_imagery_projection:semi_major_axis
    H = 42164160 # meters
    # goes_imagery_projection: longitude_of_projection_origin
    lambda0 = -1.308996939

    # Convert to radians
    latRad = lat * (math.pi/180)
    lonRad = lon * (math.pi/180)

    # (1) geocentric latitude
    Phi_c = math.atan(((rpol * rpol)/(req * req)) * math.tan(latRad))
    # (2) geocentric distance to the point on the ellipsoid
    re = rpol/(math.sqrt(1 - ((e * e) * (math.cos(Phi_c) * math.cos(Phi_c)))))
    # (3) sx
    sx = H - (rc * math.cos(Phi_c) * math.cos(lonRad - lambda0))
    # (4) sy
    sy = -rc * math.cos(Phi_c) * math.sin(lonRad - lambda0)
    # (5)
    sz = rc * math.sin(Phi_c)
    # =y
    x = math.asin((-sy)/math.sqrt((sx*sx) + (sy*sy) + (sz*sz)))
    y = math.atan(sz/sx)
    return x, y

# Function to convert lat / lon extent to GOES-16 extents
def convertExtent2GOESProjection(extent):
    # GOES-16 viewing point (satellite position) height above the earth
    GOES16_HEIGHT = 35786023.0
    # GOES-16 East-West position
    GOES16_LONGITUDE = -75.0

    a, b = latlon2xy(extent[1], extent[0])
    c, d = latlon2xy(extent[3], extent[2])
    return (a * GOES16_HEIGHT, c * GOES16_HEIGHT, b * GOES16_HEIGHT, d * GOES16_HEIGHT)
```

Script 13: Cropping a Full Disk Image and Creating an RGB Composite

QUESTION: Why are we reading channel 2 at every 4 and channel 5 at every 2?

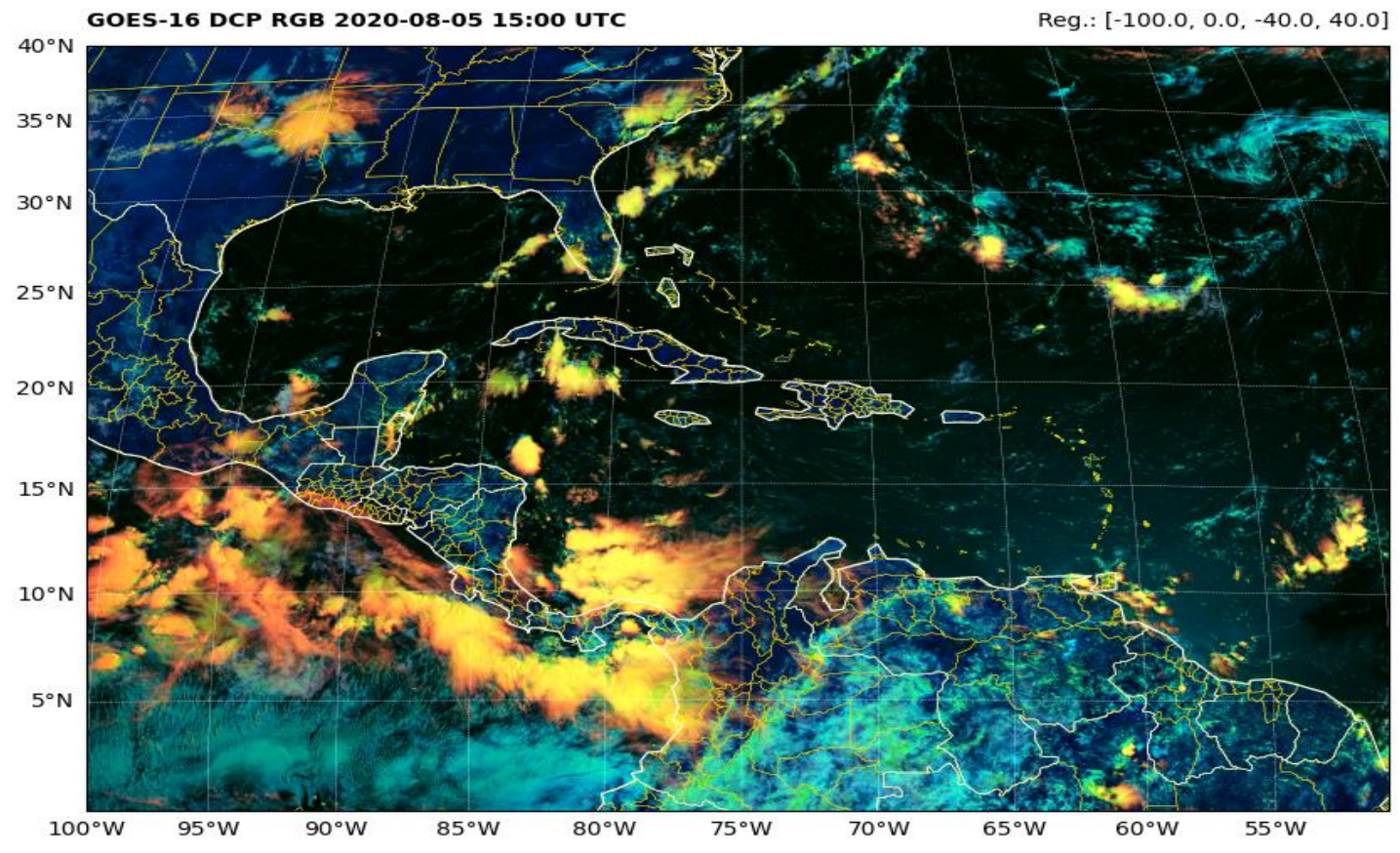
```
33  # Open the GOES-R images
34  file_ch13 = Dataset(f'{input}/{file_ch13}.nc')
35  file_ch02 = Dataset(f'{input}/{file_ch02}.nc')
36  file_ch05 = Dataset(f'{input}/{file_ch05}.nc')
37  #---------------------------------------------------------------------------
38  # Convert lat/lon to grid-coordinates
39  lly, llx = geo2grid(extent[1], extent[0], file_ch13)
40  ury, urx = geo2grid(extent[3], extent[2], file_ch13)
41
42  # Get the pixel values
43  data_ch13 = file_ch13.variables['CMI'][ury:lly, llx:urx] - 273.15
44  #---------------------------------------------------------------------------
45  # Convert lat/lon to grid-coordinates
46  lly, llx = geo2grid(extent[1], extent[0], file_ch02)
47  ury, urx = geo2grid(extent[3], extent[2], file_ch02)
48
49  # Get the pixel values
50  data_ch02 = file_ch02.variables['CMI'][ury:lly, llx:urx][::4 ,::4]
51  #---------------------------------------------------------------------------
52  # Convert lat/lon to grid-coordinates
53  lly, llx = geo2grid(extent[1], extent[0], file_ch05)
54  ury, urx = geo2grid(extent[3], extent[2], file_ch05)
55
56  # Get the pixel values
57  data_ch05 = file_ch05.variables['CMI'][ury:lly, llx:urx][::2 ,::2]
58  #---------------------------------------------------------------------------
```

GOES-16 Channels (ABI Sensor)

ABI Band	Central Wavelength (µm)	Wavelength Range (µm)	Resolution (km)	Spectral range	Descriptive Name
1	0.47	0.45 - 0.49	1	Visible	Blue
2	0.64	0.68 - 0.68	0.5	Visible	Red
3	0.86	0.847 - 0.882	1	Near Infrared	Vegetation
4	1.37	1.366 - 1.380	2	Near Infrared	Cirrus
5	1.6	1.59 - 1.63	1	Near Infrared	Snow/Ice
6	2.2	2.22 - 2.27	2	Near Infrared	Cloud Particle Size
7	3.9	3.80 - 3.99	2	Shortwave Infrared	Shortwave window
8	6.2	5.79 - 6.59	2	Midwave Infrared	Upper-level water vapor
9	6.9	6.72 - 7.14	2	Midwave Infrared	Midlevel water vapor
10	7.3	7.24 - 7.43	2	Midwave Infrared	Lower / midlevel watervapor
11	8.4	8.23 - 8.66	2	Longwave Infrared	Cloud-top phase
12	9.6	9.42 - 9.80	2	Longwave Infrared	Ozone
13	10.3	10.18 - 10.48	2	Longwave Infrared	Clean longwave window
14	11.2	10.82 - 11.60	2	Longwave Infrared	Longwave window
15	12.3	11.83 - 12.75	2	Longwave Infrared	Dirty longwave window
16	13.3	12.99-13.56	2	Longwave Infrared	CO2

Script 13: Cropping a Full Disk Image and Creating an RGB Composite

Script 13: Cropping a Full Disk Image and Creating an RGB Composite



Script 14: Reprojection With GDAL

```
# Training: Python and GOES-R Imagery: Script 14 - Reprojection with GDAL
#
# Required modules
from netCDF4 import Dataset                    # Read / Write NetCDF4 files
import matplotlib.pyplot as plt                # Plotting
from datetime import datetime                  # Basic Dates and time types
import cartopy, cartopy.crs as ccrs            # Plot maps
import os                                      # Miscellaneous operating system interfaces
from osgeo import osr                          # Python bindings for GDAL
from osgeo import gdal                         # Python bindings for GDAL
import numpy as np                             # Scientific computing with Python
from utilities import download_CMI             # Our function for download
gdal.PushErrorHandler('CPLQuietErrorHandler')  # Ignore GDAL warnings
#
# Input and output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# Desired extent
extent = [-64.0, -35.0, -35.0, -15.0]  # Min lon, Max lon, Min lat, Max lat

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-CMIPF'
yyyymmddhhmn = '202102181800'
band = '13'

# Download the file
file_name = download_CMI(yyyymmddhhmn, band, input)
#
```

LIBRARIES AND FUNCTIONS

GDAL Libraries

Prevent warnings for appearing

DOWNLOAD THE DATA

Extensión deseada

Script 14: Reprojection With GDAL

```
33
34  # Variable
35  var = 'CMI'
36
37  # Open the file
38  img = gdal.Open(f'NETCDF:{input}/{file_name}.nc:' + var)
39
40  # Read the header metadata
41  metadata = img.GetMetadata()
42  scale = float(metadata.get(var + '#scale_factor'))
43  offset = float(metadata.get(var + '#add_offset'))
44  undef = float(metadata.get(var + '#_FillValue'))
45  dtime = metadata.get('NC_GLOBAL#time_coverage_start')
46
47  # Load the data
48  ds = img.ReadAsArray(0, 0, img.RasterXSize, img.RasterYSize).astype(float)
49
50  # Apply the scale, offset and convert to celsius
51  ds = (ds * scale + offset) - 273.15
52
53  # Read the original file projection and configure the output projection
54  source_prj = osr.SpatialReference()
55  source_prj.ImportFromProj4(img.GetProjectionRef())
56
57  target_prj = osr.SpatialReference()
58  target_prj.ImportFromProj4("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
59
60  # Reproject the data
61  GeoT = img.GetGeoTransform()
62  driver = gdal.GetDriverByName('MEM')
63  raw = driver.Create('raw', ds.shape[0], ds.shape[1], 1, gdal.GDT_Float32)
64  raw.SetGeoTransform(GeoT)
65  raw.GetRasterBand(1).WriteArray(ds)
66
67  # Define the parameters of the output file
68  kwargs = {'format': 'netCDF', \
69            'srcSRS': source_prj, \
70            'dstSRS': target_prj, \
71            'outputBounds': (extent[0], extent[3], extent[2], extent[1]), \
72            'outputBoundsSRS': target_prj, \
73            'outputType': gdal.GDT_Float32, \
74            'srcNodata': undef, \
75            'dstNodata': 'nan', \
76            'xRes': 0.02, \
77            'yRes': 0.02, \
78            'resampleAlg': gdal.GRA_NearestNeighbour}
79
80  # Write the reprojected file on disk
81  gdal.Warp(f'{output}/{file_name}_ret.nc', raw, **kwargs)
82
```

REPROJECTION AND CROPPING

Variable to be read

Read (specifies the drive [NetCDF, GeoTIFF, GRIB, etc])

Metadata (scale, offset, NaN, start of scan, etc)

Read only the data and convert to an array

Origin projection (geos)

Desired projection (equidistant)

We are creating a new GDAL file (header + data), with the same origin shape and desired projection

Arguments with all the necessary specifications

https://gdal.org/python/osgeo.gdal-module.html

Desired resolution (degrees)

Write new NetCDF file (already reprojected)

Script 14: Reprojection With GDAL

gdal.org/python/osgeo.gdal-module.html

```
WarpOptions(options=None, format=None, outputBounds=None, outputBoundsSRS=None, xRes=None, yRes=None, targetAlignedPixels=False, width=0, height=0, srcSRS=None, dstSRS=None,
coordinateOperation=None, srcAlpha=False, dstAlpha=False, warpOptions=None, errorThreshold=None, warpMemoryLimit=None, creationOptions=None,
outputType=gdalconst.GDT_Unknown, workingType=gdalconst.GDT_Unknown, resampleAlg=None, srcNodata=None, dstNodata=None, multithread=False, tps=False, rpc=False, geoloc=False,
polynomialOrder=None, transformerOptions=None, cutlineDSName=None, cutlineLayer=None, cutlineWhere=None, cutlineSQL=None, cutlineBlend=None, cropToCutline=False,
copyMetadata=True, metadataConflictValue=None, setColorInterpretation=False, overviewLevel='AUTO', callback=None, callback_data=None)

Create a WarpOptions() object that can be passed to gdal.Warp()
Keyword arguments are :
  options --- can be be an array of strings, a string or let empty and filled from other keywords.
  format --- output format ("GTiff", etc...)
  outputBounds --- output bounds as (minX, minY, maxX, maxY) in target SRS
  outputBoundsSRS --- SRS in which output bounds are expressed, in the case they are not expressed in dstSRS
  xRes, yRes --- output resolution in target SRS
  targetAlignedPixels --- whether to force output bounds to be multiple of output resolution
  width --- width of the output raster in pixel
  height --- height of the output raster in pixel
  srcSRS --- source SRS
  dstSRS --- output SRS
  coordinateOperation -- coordinate operation as a PROJ string or WKT string
  srcAlpha --- whether to force the last band of the input dataset to be considered as an alpha band
  dstAlpha --- whether to force the creation of an output alpha band
  outputType --- output type (gdalconst.GDT_Byte, etc...)
  workingType --- working type (gdalconst.GDT_Byte, etc...)
  warpOptions --- list of warping options
  errorThreshold --- error threshold for approximation transformer (in pixels)
  warpMemoryLimit --- size of working buffer in MB
  resampleAlg --- resampling mode
  creationOptions --- list of creation options
  srcNodata --- source nodata value(s)
  dstNodata --- output nodata value(s)
  multithread --- whether to multithread computation and I/O operations
  tps --- whether to use Thin Plate Spline GCP transformer
  rpc --- whether to use RPC transformer
  geoloc --- whether to use GeoLocation array transformer
  polynomialOrder --- order of polynomial GCP interpolation
  transformerOptions --- list of transformer options
  cutlineDSName --- cutline dataset name
  cutlineLayer --- cutline layer name
  cutlineWhere --- cutline WHERE clause
  cutlineSQL --- cutline SQL statement
  cutlineBlend --- cutline blend distance in pixels
  cropToCutline --- whether to use cutline extent for output bounds
  copyMetadata --- whether to copy source metadata
  metadataConflictValue --- metadata data conflict value
  setColorInterpretation --- whether to force color interpretation of input bands to output bands
  overviewLevel --- To specify which overview level of source files must be used
  callback --- callback method
  callback_data --- user data for callback
```

Parameters that can be specified

https://gdal.org/python/osgeo.gdal-module.html

Script 14: Reprojection With GDAL

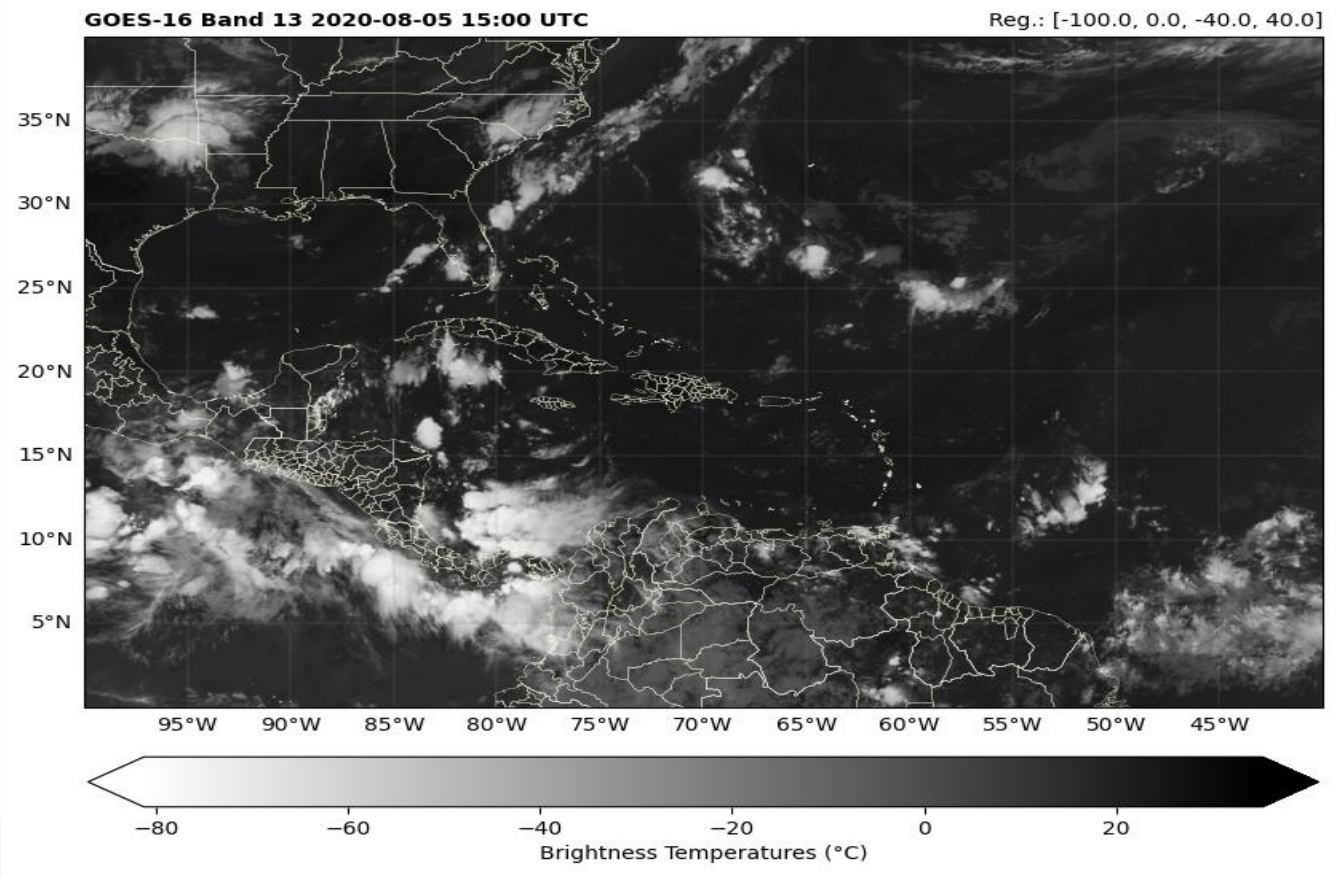
PLOT AND IMAGE GENERATION

```
# Open the reprojected GOES-R image
file = Dataset(f'{output}/{file_name}_ret.nc')

# Get the pixel values
data = file.variables['Band1'][:]
#---------------------------------------------------------------------------------------------------
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(10,10))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.PlateCarree())

# Define the image extent
img_extent = [extent[0], extent[2], extent[1], extent[3]]

# Define the color scale based on the channel
colormap = "gray_r" # White to black for IR channels

# Plot the image
img = ax.imshow(data, origin='upper', extent=img_extent, cmap=colormap)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='white', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.5)
gl = ax.gridlines(crs=ccrs.PlateCarree(), color='gray', alpha=1.0, linestyle='--', linewidth=0.25, xlocs=np.arange(-180, 180, 5), ylocs=np.arange(-90, 90, 5), draw_labels=True
gl.top_labels = False
gl.right_labels = False

# Add a colorbar
plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(dtime, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 Band 13 ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Reg.: ' + str(extent) , fontsize=10, loc='right')
#---------------------------------------------------------------------------------------------------
# Save the image
plt.savefig(f'{output}/{file_name}_ret.png', bbox_inches='tight', pad_inches=0, dpi=300)

# Show the image
plt.show()
```

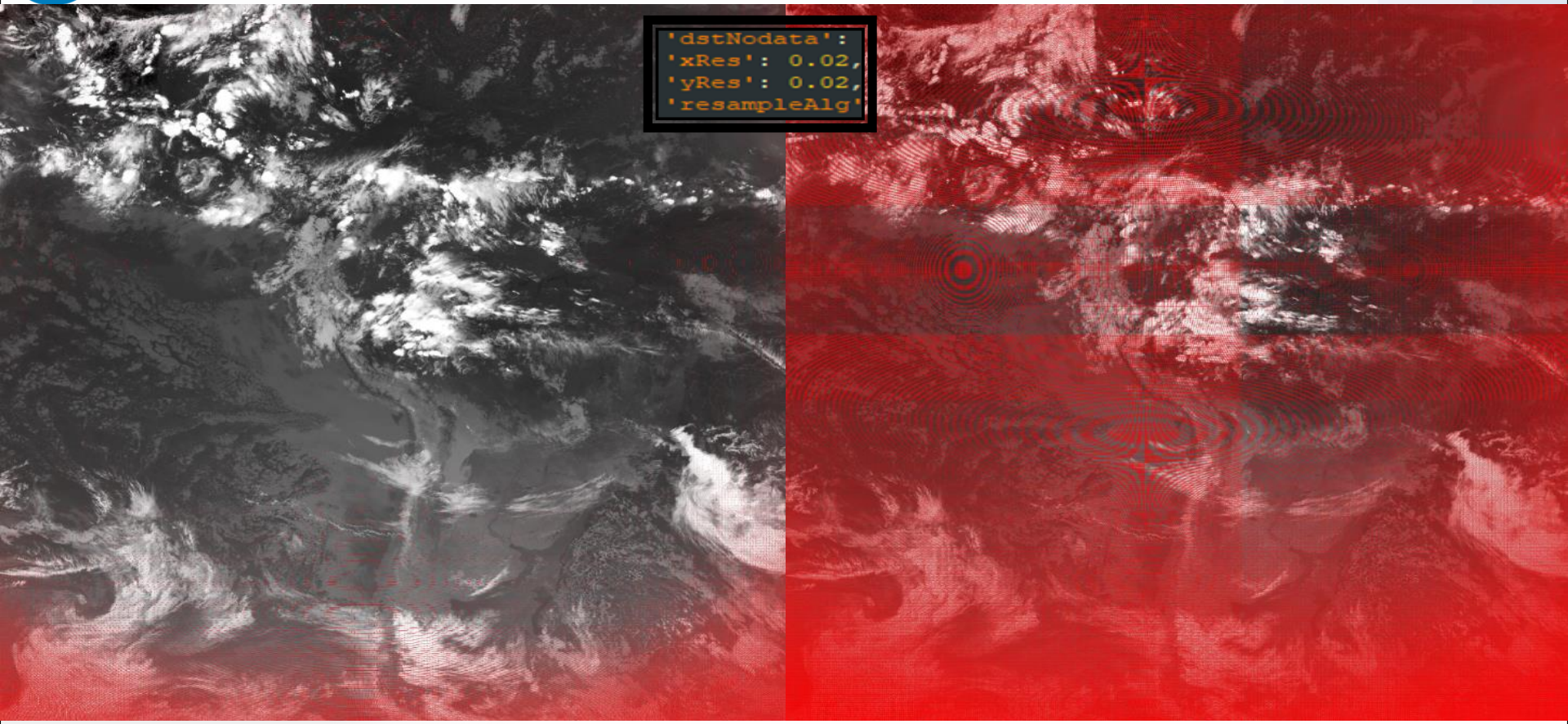
Dataset inside the new NetCDF

Setting the Gridlines / axes

Script 14: Reprojection With GDAL



Script 14: Reprojection With GDAL



Script 15: Level 2 Products (SST) and Quality Flags (DQF)

```
# Training: Python and GOES-R Imagery: Script 15 - Level 2 Products (SST) and Data Quality Flags

# Required modules
from netCDF4 import Dataset                 # Read / Write NetCDF4 files
import matplotlib.pyplot as plt             # Plotting library
from datetime import datetime               # Basic Dates and time types
import cartopy, cartopy.crs as ccrs         # Plot maps
import os                                   # Miscellaneous operating system interfaces
from osgeo import gdal                      # Python bindings for GDAL
import numpy as np                          # Scientific computing with Python
from utilities import download_PROD         # Our function for download
from utilities import reproject             # Our function for reproject
gdal.PushErrorHandler('CPLQuietErrorHandler')  # Ignore GDAL warnings
#---------------------------------------------------------------------------------------------------

# Input and output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# Desired extent
extent = [-60.0, -40.0, -35.0, -20.0] # Min lon, Max lon, Min lat, Max lat

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-SSTF'
yyyymmddhhmn = '202102181800'

# Download the file
file_name = download_PROD(yyyymmddhhmn, product_name, input)

#---------------------------------------------------------------------------------------------------
```

LIBRARIES AND FUNCTIONS

External Functions

DOWNLOAD THE DATA

Desired extent

Script 15: Level 2 Products (SST) and Quality Flags (DQF)

REPROJECTION AND CROPPING

```
32
33  # Variable
34  var = 'SST'
35
36  # Open the file
37  img = gdal.Open(f'NETCDF:{input}/{file_name}.nc:' + var)
38
39  # Data Quality Flag (DQF)
40  dqf = gdal.Open(f'NETCDF:{input}/{file_name}.nc:DQF')
41
42  # Read the header metadata
43  metadata = img.GetMetadata()
44  scale = float(metadata.get(var + '#scale_factor'))
45  offset = float(metadata.get(var + '#add_offset'))
46  undef = float(metadata.get(var + '#_FillValue'))
47  dtime = metadata.get('NC_GLOBAL#time_coverage_start')
48
49  # Load the data
50  ds = img.ReadAsArray(0, 0, img.RasterXSize, img.RasterYSize).astype(float)
51  ds_dqf = dqf.ReadAsArray(0, 0, dqf.RasterXSize, dqf.RasterYSize).astype(float)
52
53  # Apply the scale, offset and convert to celsius
54  ds = (ds * scale + offset) - 273.15
55
56  # Apply NaN's where the quality flag is greater than 1
57  ds[ds_dqf > 1] = np.nan
58
59  # Reproject the file
60  filename_ds = f'{output}/{file_name}_ret.nc'
61  reproject(filename_ds, img, ds, extent, undef)
62
63  #--------------------------------------------------------------------------------------------------
64  # Open the reprojected GOES-R image
65  file = Dataset(filename_ds)
66
67  # Get the pixel values
68  data = file.variables['Band1'][:]
69  #--------------------------------------------------------------------------------------------------
```

Variable to be read (SST)

Data Quality Flag

Reprojection function call

Script 15: Level 2 Products (SST) and Quality Flags (DQF)

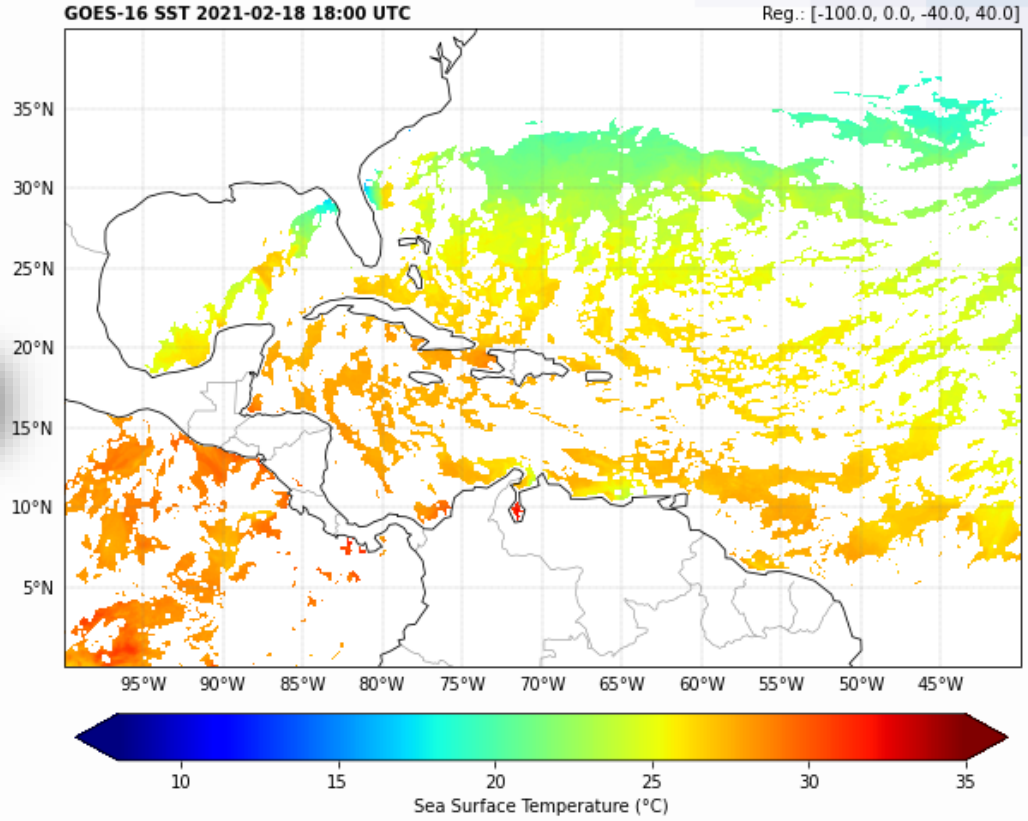
PLOT AND IMAGE GENERATION

```
#-------------------------------------------------------------------------------------------------
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(10,10))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.PlateCarree())

# Define the image extent
img_extent = [extent[0], extent[2], extent[1], extent[3]]

# Plot the image
img = ax.imshow(data, vmin=8, vmax=35, cmap='jet', origin='upper', extent=img_extent)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='black', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='black', linewidth=0.5)
gl = ax.gridlines(crs=ccrs.PlateCarree(), color='gray', alpha=1.0, linestyle='--', linewidth=0.25, xlocs=np.arange(-180, 180, 5), ylocs=np.arange(-90, 90, 5), draw_labels=True)
gl.top_labels = False
gl.right_labels = False

# Add a colorbar
plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(dtime, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 SST ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Reg.: ' + str(extent) , fontsize=10, loc='right')
#-------------------------------------------------------------------------------------------------
# Save the image
plt.savefig(f'{output}/{file_name}_ret.png', bbox_inches='tight', pad_inches=0, dpi=300)

# Show the image
plt.show()
```

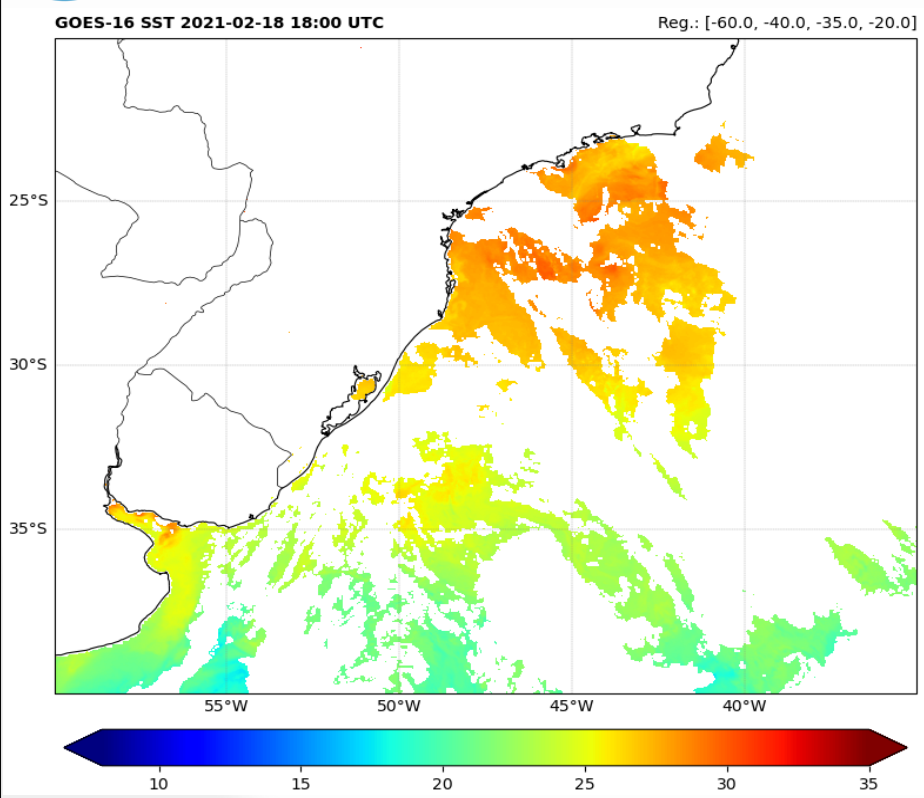
Script 15: Level 2 Products (SST) and Quality Flags (DQF)

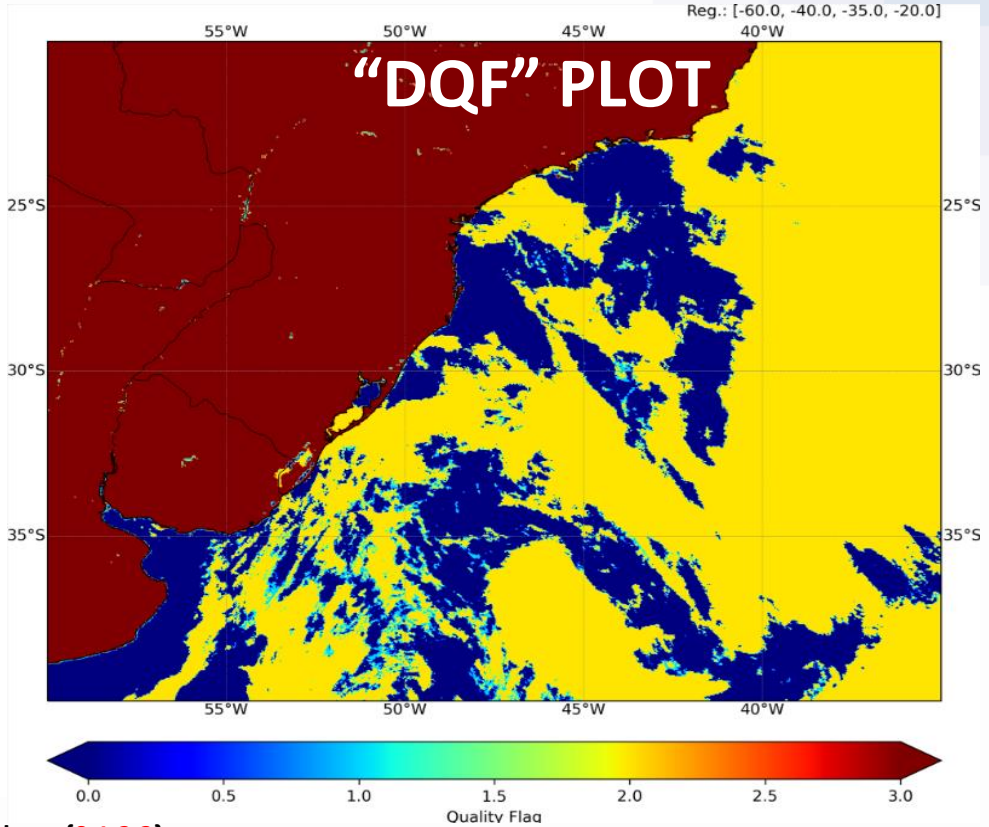
```
56  # Apply NaN's where the quality flag is greater than 1
57  ds[ds_dqf > 1] = np.nan
58
```

Blank regions



Script 15: Level 2 Products (SST) and Quality Flags (DQF)





flag_values={0,1,2,3}

flag_meanings=good_quality_qf degraded_quality_qf severely_degraded_quality_qf invalid_due_to_unprocessed_qf

Script 16: Level 2 Product (SST) and Daily Average

```python
# Training: Python and GOES-R Imagery: Script 16 - Level 2 Products (SST) and Average
#-----------------------------------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset                    # Read / Write NetCDF4 files
import matplotlib.pyplot as plt                # Plotting library
from datetime import datetime                  # Basic Dates and time types
import cartopy, cartopy.crs as ccrs            # Plot maps
import os                                      # Miscellaneous operating system interfaces
from osgeo import gdal                         # Python bindings for GDAL
import numpy as np                             # Scientific computing with Python
from utilities import download_PROD            # Our function for download
from utilities import reproject                # Our function for reproject
gdal.PushErrorHandler('CPLQuietErrorHandler')  # Ignore GDAL warnings

#-----------------------------------------------------------------------------------------------------------
# Input and output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-SSTF'
yyyymmdd = '20210218'

# Desired extent
extent = [-60.0, -40.0, -35.0, -20.0] # Min lon, Max lon, Min lat, Max lat
```

LIBRARIES AND FUNCTIONS

External Functions

DESIRED DATA

Desired extent

Script 16: Level 2 Product (SST) and Daily Average

```
sum_ds = np.zeros((5424,5424))
count_ds = np.zeros((5424,5424))
#----------------------------------------------------------------------------
for hour in np.arange(0,23,1):

    # Date structure
    yyyymmddhhmn = f'{yyyymmdd}{hour:02.0f}00'

    # Download the file
    file_name = download_PROD(yyyymmddhhmn, product_name, input)
    #----------------------------------------------------------------------------
    # Variable
    var = 'SST'

    # Open the GOES-R image
    file = Dataset(f'{input}/{file_name}.nc')

    # Open the file
    img = gdal.Open(f'NETCDF:{input}/{file_name}.nc:' + var)

    # Data Quality Flag (DQF)
    dqf = gdal.Open(f'NETCDF:{input}/{file_name}.nc:DQF')

    # Read the header metadata
    metadata = img.GetMetadata()
    scale = float(metadata.get(var + '#scale_factor'))
    offset = float(metadata.get(var + '#add_offset'))
    undef = float(metadata.get(var + '#_FillValue'))
    dtime = metadata.get('NC_GLOBAL#time_coverage_start')

    # Load the data
    ds = img.ReadAsArray(0, 0, img.RasterXSize, img.RasterYSize).astype(float)
    ds_dqf = dqf.ReadAsArray(0, 0, dqf.RasterXSize, dqf.RasterYSize).astype(float)

    # Apply the scale, offset and convert to celsius
    ds = (ds * scale + offset) - 273.15

    # Apply NaN's where the quality flag is greater than 1
    ds[ds_dqf > 1] = np.nan

    # Calculate the sum
    sum_ds = np.nansum(np.dstack((sum_ds,ds)),2)

    count_ds = np.nansum(np.dstack((count_ds,(ds/ds))),2)
    #----------------------------------------------------------------------------
# Calculate the sum
ds_day = np.empty((5424,5424))
ds_day[::] = np.nan
ds_day[count_ds!=0] = sum_ds[count_ds!=0]/count_ds[count_ds!=0]
```

Creation of the arrays

24-hour loop or “x” hours

Data structure (every hour, with zero at the end)

Download the file for the “x” hour

Previous procedure

Sum, disregarding NaN's, keeping data in 2 dimensions

Divide the data by itself. We will have a grid of 1 and NaN's. Then we do the sum to check the number of valid values

Calculate the average where it is not zero

CALCULATES THE DAILY AVERAGE

```
    # Calculate the sum
    sum_ds = np.nansum(np.dstack((sum_ds,ds)),2)

    count_ds = np.nansum(np.dstack((count_ds,(ds/ds))),2)
    #----------------------------------------------------------------------------
# Calculate the sum
ds_day = np.empty((5424,5424))
ds_day[::] = np.nan
ds_day[count_ds!=0] = sum_ds[count_ds!=0]/count_ds[count_ds!=0]
```

Script 16: Level 2 Product (SST) and Daily Average

1	1	1
1	1	NaN
1	1	1

+

NaN	1	1
1	1	1
1	1	1

=

NaN	2	2
2	2	NaN
2	2	2

np.nansum(

1	1	1
1	1	NaN
1	1	1

,

NaN	1	1
1	1	1
1	1	1

) =

1	2	2
2	2	1
2	2	2

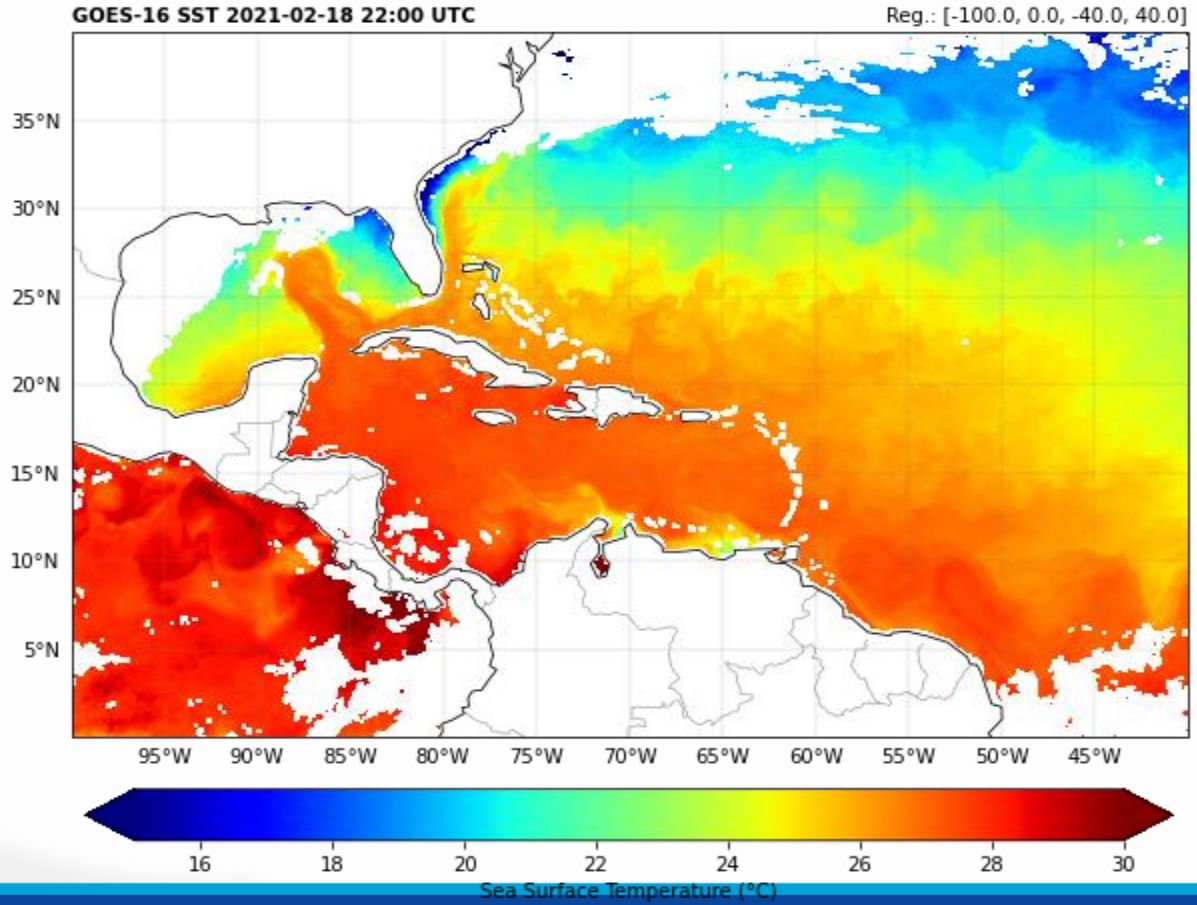
Script 16: Level 2 Product (SST) and Daily Average

```
# Reproject the file
filename_ds = f'{output}/{file_name}_ret.nc'
reproject(filename_ds, img, ds_day, extent, undef)
#-------------------------------------------------------------------------------------------------------------
# Open the reprojected GOES-R image
file = Dataset(filename_ds)

# Get the pixel values
data = file.variables['Band1'][:]
#-------------------------------------------------------------------------------------------------------------
# Create the plot with a specific size (in inches)
plt.figure(figsize=(10,10))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.PlateCarree())

# Define the image extent
img_extent = [extent[0], extent[2], extent[1], extent[3]]

# Plot the image
img = ax.imshow(data, vmin=15, vmax=30, cmap='jet', origin='upper', extent=img_extent)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='black', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='black', linewidth=0.5)
gl = ax.gridlines(crs=ccrs.PlateCarree(), color='gray', alpha=1.0, linestyle='--', linewidth=0.25, xlocs=np.arange(-180, 180, 5), ylocs=np.arange(-90, 90, 5), draw_labels=True)
gl.top_labels = False
gl.right_labels = False

plt.xlim(extent[0], extent[2])
plt.ylim(extent[1], extent[3])

# Add a colorbar
plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(dtime, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('GOES-16 SST ' + date.strftime('%Y-%m-%d %H:%M') + ' UTC', fontweight='bold', fontsize=10, loc='left')
plt.title('Reg.: ' + str(extent) , fontsize=10, loc='right')
#-------------------------------------------------------------------------------------------------------------
# Save the image
plt.savefig(f'{output}/SST_DAY_RET_{yyyymmdd}.png', bbox_inches='tight', pad_inches=0, dpi=300)

# Show the image
plt.show()
```

REPROJECTION OF THE AVERAGE (only one time)

PLOT AND IMAGE GENERATION

Script 16: Level 2 Product (SST) and Daily Average



Script 17: Level 2 Product (RRQPE) and Data Accumulation

```
# Training: Python and GOES-R Imagery: Script 17 - Level 2 Products (RRQPE) and Data Accumulation
#-----------------------------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset                    # Read / Write NetCDF4 files
import matplotlib.pyplot as plt                # Plotting library
from datetime import timedelta, date, datetime # Basic Dates and time types
import cartopy, cartopy.crs as ccrs            # Plot maps
import os                                      # Miscellaneous operating system interfaces
from osgeo import gdal                         # Python bindings for GDAL
import numpy as np                             # Scientific computing with Python
from matplotlib import cm                      # Colormap handling utilities
from utilities import download_PROD            # Our function for download
from utilities import reproject                # Our function for reproject
gdal.PushErrorHandler('CPLQuietErrorHandler')  # Ignore GDAL warnings

#-----------------------------------------------------------------------------------------------------
# Input and output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# Desired extent
extent = [-74.0, -34.1, -34.8, 5.5] # Min lon, Max lon, Min lat, Max lat

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-RRQPEF'
yyyymmdd = '20201217'
```

LIBRARIES AND FUNCTIONS

External functions

DESIRED DATA

Desired extent

Script 17: Level 2 Product (RRQPE) and Data Accumulation

```
# Initial time and date
yyyy = datetime.strptime(yyyymmdd, '%Y%m%d').strftime('%Y')
mm = datetime.strptime(yyyymmdd, '%Y%m%d').strftime('%m')
dd = datetime.strptime(yyyymmdd, '%Y%m%d').strftime('%d')

date_ini = str(datetime(int(yyyy),int(mm),int(dd),12,0) - timedelta(hours=23))
date_end = str(datetime(int(yyyy),int(mm),int(dd),12,0))

acum = np.zeros((5424,5424))
```

separates day month and year

START AND END TIME / DATE

Initial time (data – 23 hours)

Creation of the 2 km array

Data from 1:00 p.m. to 12:00 p.m. the following day

ACCUMULATION

```
# Accumulation
while (date_ini <= date_end):

    # Date structure
    yyyymmddhhmn = datetime.strptime(date_ini, '%Y-%m-%d %H:%M:%S').strftime('%Y%m%d%H%M')

    # Download the file
    file_name = download_PROD(yyyymmddhhmn, product_name, input)
    #------------------------------------------------------------------------------
    # Variable
    var = 'RRQPE'

    # Open the file
    img = gdal.Open(f'NETCDF:{input}/{file_name}.nc:' + var)
    dqf = gdal.Open(f'NETCDF:{input}/{file_name}.nc:DQF')

    # Read the header metadata
    metadata = img.GetMetadata()
    scale = float(metadata.get(var + '#scale_factor'))
    offset = float(metadata.get(var + '#add_offset'))
    undef = float(metadata.get(var + '#_FillValue'))
    dtime = metadata.get('NC_GLOBAL#time_coverage_start')

    # Load the data
    ds = img.ReadAsArray(0, 0, img.RasterXSize, img.RasterYSize).astype(float)
    ds_dqf = dqf.ReadAsArray(0, 0, dqf.RasterXSize, dqf.RasterYSize).astype(float)

    # Remove undef
    ds[ds == undef] = np.nan

    # Apply the scale, offset and convert to celsius
    ds = (ds * scale + offset)

    # Apply NaN's where the quality flag is greater than 1
    ds[ds_dqf > 0] = np.nan

    # Sum the instantaneous value in the accumulation
    acum = np.nansum(np.dstack((acum, ds)),2)

    # Increment 1 hour
    date_ini = str(datetime.strptime(date_ini, '%Y-%m-%d %H:%M:%S') + timedelta(hours=1))
    #------------------------------------------------------------------------------
```

```
81      # Sum the instantaneous value in the accumulation
82      acum = np.nansum(np.dstack((acum, ds)),2)
```

Standard procedure

Sum, disregarding NaN's

increase the time

Script 17: Level 2 Product (RRQPE) and Data Accumulation

1	1	1
1	1	NaN
1	1	1

+

NaN	1	1
1	1	1
1	1	1

=

NaN	2	2
2	2	NaN
2	2	2

np.nansum(

1	1	1
1	1	NaN
1	1	1

,

NaN	1	1
1	1	1
1	1	1

) =

1	2	2
2	2	1
2	2	2

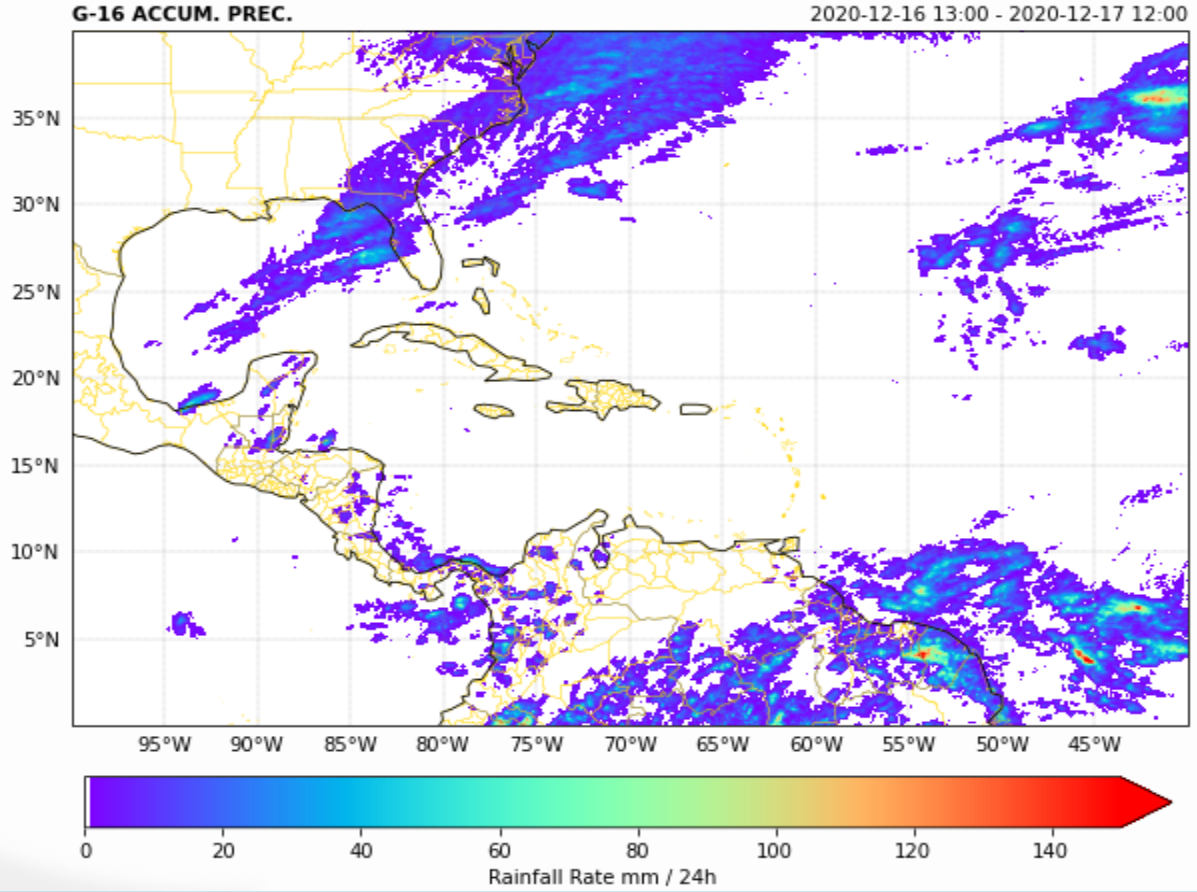
Script 17: Level 2 Product (RRQPE) and Data Accumulation

REPROJECTION OF THE ACCUMULATION (only one time)

PLOT AND IMAGE GENERATION

```
# Reproject the file
filename_acum = f'{output}/prec_acum_ret_{yyyymmddhhmn}.nc'
reproject(filename_acum, img, acum, extent, undef)
#---------------------------------------------------------------------------------------
# Open the reprojected GOES-R image
file = Dataset(filename_acum)

# Get the pixel values
data = file.variables['Band1'][:]
#
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(10,10))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.PlateCarree())

# Define the image extent
img_extent = [extent[0], extent[2], extent[1], extent[3]]

# Modify the colormap to zero values are white
colormap = cm.get_cmap('rainbow', 240)
newcolormap = colormap(np.linspace(0, 1, 240))
newcolormap[:1, :] = np.array([1, 1, 1, 1])
cmap = cm.colors.ListedColormap(newcolormap)

# Plot the image
img = ax.imshow(data, vmin=0, vmax=150, cmap=cmap, origin='upper', extent=img_extent)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='black', linewidth=0.8)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='black', linewidth=0.5)
gl = ax.gridlines(crs=ccrs.PlateCarree(), color='gray', alpha=1.0, linestyle='--', linewidth=0.25, xlocs=np.arange(-180, 180, 5), ylocs=np.arange(-90, 90, 5), draw_labels=True)
gl.top_labels = False
gl.right_labels = False

# Add a colorbar
plt.colorbar(img, label='Rainfall Rate mm / 24h', extend='max', orientation='horizontal', pad=0.05, fraction=0.05)

# Extract the date
date = (datetime.strptime(dtime, '%Y-%m-%dT%H:%M:%S.%fZ'))

# Add a title
plt.title('G-16 ACCUM. PREC.', fontweight='bold', fontsize=10, loc='left')
date_ini = datetime(int(yyyy),int(mm),int(dd),12,0) - timedelta(hours=23)
date_end = datetime(int(yyyy),int(mm),int(dd),12,0)
plt.title(date_ini.strftime('%Y-%m-%d %H:%M') + " - " + date_end.strftime('%Y-%m-%d %H:%M'), fontsize=10, loc='right')
#---------------------------------------------------------------------------------------
# Save the image
plt.savefig(f'{output}/PREC_ACUM_RET_{yyyymmddhhmn}.png', bbox_inches='tight', pad_inches=0, dpi=300)

# Show the image
plt.show()
```

Script 17: Level 2 Product (RRQPE) and Data Accumulation



Script 18: GLM Density

```
# Training: Python and GOES-R Imagery: Script 19 - GLM Density
#-------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset                          # Read/Write NetCDF4 files
import matplotlib.pyplot as plt                      # Plotting library
import cartopy, cartopy.crs as ccrs                  # Plot maps
from datetime import timedelta, date, datetime       # Basic Dates and time types
import cartopy, cartopy.crs as ccrs                  # Plot maps
import os                                            # Miscellaneous operating system interfaces
from osgeo import gdal                               # Python bindings for GDAL
import numpy as np                                   # Scientific computing with Python
from matplotlib import cm                            # Colormap handling utilities
from utilities import download_CMI, download_GLM     # Our function for download
from utilities import reproject                      # Our function for reproject
gdal.PushErrorHandler('CPLQuietErrorHandler')        # Ignore GDAL warnings
#-------------------------------------------------------------------------------
# Input and output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# Desired extent
extent = [-74.0, -34.1, -34.8, 5.5] # Min lon, Max lon, Min lat, Max lat

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
yyyymmddhhmn = '202102081800'

#-------------------------------------------------------------------------------
# Get the Band 13 Data

# Download the file
file_ir = download_CMI(yyyymmddhhmn, 13, input)
#-------------------------------------------------------------------------------
```

LIBRARIES AND FUNCTIONS

External functions

DOWNLOADS THE IR DATA

Desired extent

Script 18: GLM Density

REPROJECTION (BAND 13)

```
35  #------------------------------------------------------------------------------------------------------
36  # Variable
37  var = 'CMI'
38
39  # Open the file
40  img = gdal.Open(f'NETCDF:{input}/{file_ir}.nc:' + var)
41
42  # Data Quality Flag (DQF)
43  dqf = gdal.Open(f'NETCDF:{input}/{file_ir}.nc:DQF')
44
45  # Read the header metadata
46  metadata = img.GetMetadata()
47  scale = float(metadata.get(var + '#scale_factor'))
48  offset = float(metadata.get(var + '#add_offset'))
49  undef = float(metadata.get(var + '#_FillValue'))
50  dtime = metadata.get('NC_GLOBAL#time_coverage_start')
51
52  # Load the data
53  ds_cmi = img.ReadAsArray(0, 0, img.RasterXSize, img.RasterYSize).astype(float)
54  ds_dqf = dqf.ReadAsArray(0, 0, dqf.RasterXSize, dqf.RasterYSize).astype(float)
55
56  # Apply the scale, offset and convert to celsius
57  ds_cmi = (ds_cmi * scale + offset) - 273.15
58
59  # Apply NaN's where the quality flag is greater than 1
60  ds_cmi[ds_dqf > 1] = np.nan
61
62  # Reproject the file
63  filename_ret = f'{output}/IR_{yyyymmddhhmn}.nc'
64  reproject(filename_ret, img, ds_cmi, extent, undef)
65
66  # Open the reprojected GOES-R image
67  file = Dataset(filename_ret)
68
69  # Get the pixel values
70  data = file.variables['Band1'][:]
71  #------------------------------------------------------------------------------------------------------
```

Script 18: GLM Density

```
71  #-------------------------------------------------------------------------------------------------------------
72  # Get the GLM Data
73
74  # Initialize arrays for latitude, longitude, and event energy
75  lats = np.array([])
76  lons = np.array([])
77  energies = np.array([])
78  #-------------------------------------------------------------------------------------------------------------
79  # Initial time and date
80  yyyy = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%Y')
81  mm = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%m')
82  dd = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%d')
83  hh = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%H')
84  mn = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%M')
85
86  date_ini = str(datetime(int(yyyy),int(mm),int(dd),int(hh),int(mn)) - timedelta(minutes=10))
87  date_end = str(datetime(int(yyyy),int(mm),int(dd),int(hh),int(mn)))
88
89  # GLM accumulation loop
90  while (date_ini <= date_end):
91
92      # Date structure
93      yyyymmddhhmnss = datetime.strptime(date_ini, '%Y-%m-%d %H:%M:%S').strftime('%Y%m%d%H%M%S')
94
95      # Download the file
96      file_glm = download_GLM(yyyymmddhhmnss, input)
97
98      # Read the file
99      glm = Dataset(f'{input}/{file_glm}.nc')
100
101     # Append lats / longs / event energies
102     lats = np.append(lats, glm.variables['event_lat'][:])
103     lons = np.append(lons, glm.variables['event_lon'][:])
104     energies = np.append(energies, glm.variables['event_energy'][:])
105
106     # Increment the date_ini
107     date_ini = str(datetime.strptime(date_ini, '%Y-%m-%d %H:%M:%S') + timedelta(seconds=20))
108 #
```

GLM 10 MINUTE ACCUMULATION

Creation of the arrays

Separates starting year, month, day, hour and minutes

Starting date / time, 10 minutes before the end

Loop between start and end date

Data structure

Data download

Concatenating lats and lons

Increase 20 seconds

Script 18: GLM Density

```
# Stack and transpose the lat lons
values = np.vstack((lats, lons)).T

# Get the counts
points, counts = np.unique(values, axis=0, return_counts=True)

# Get the counts indices
idx = counts.argsort()
```

Transpose lats and lons (to get them in pairs) and create the stack

Unique points and how many occurrences in these points

Indices of the array of occurrences, sorted

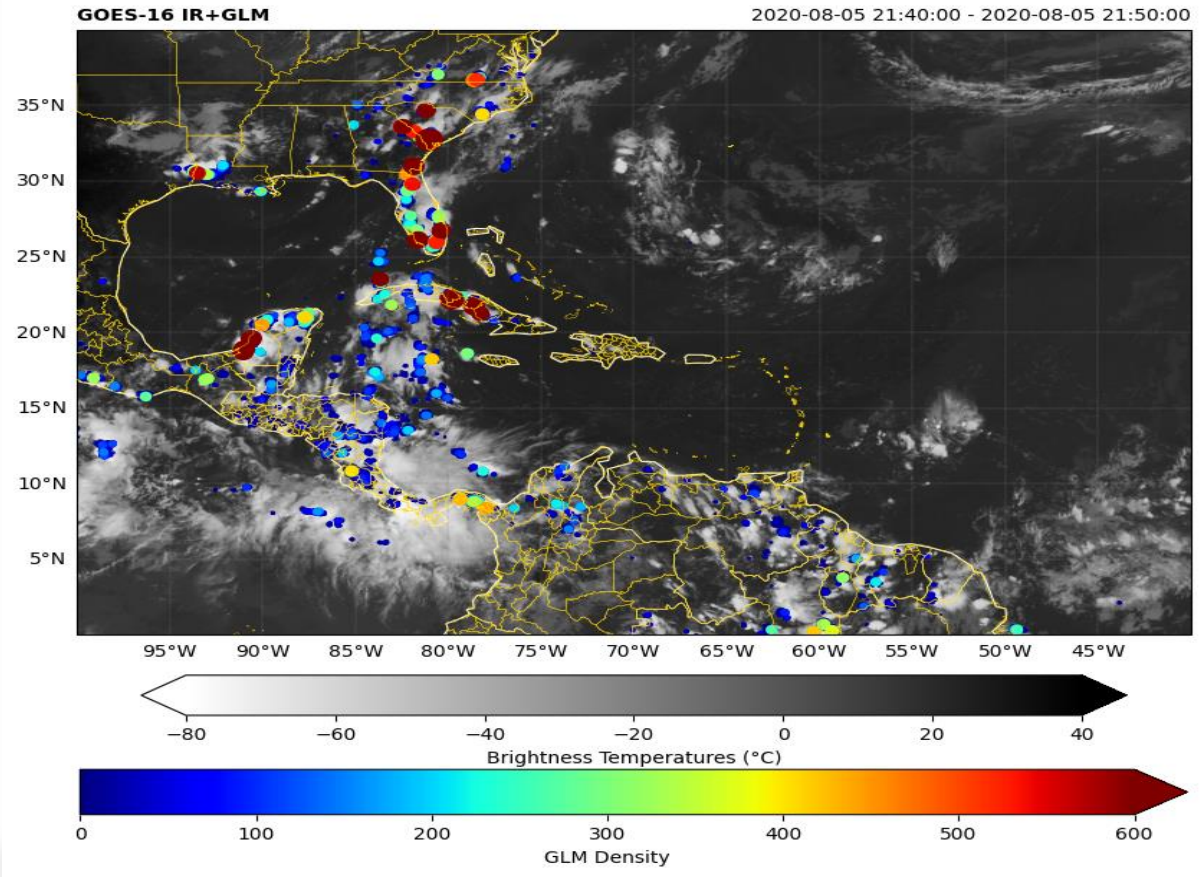
CALCULATES THE DENSITY

PLOT AND IMAGE GENERATION

```
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(10,10))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.PlateCarree())

# Define the image extent
ax.set_extent([extent[0], extent[2], extent[1], extent[3]], ccrs.PlateCarree())

# Define the data extent
img_extent = [extent[0], extent[2], extent[1], extent[3]]

# Plot the image
img = ax.imshow(data, vmin=-80, vmax=40, cmap='gray_r', origin='upper', extent=img_extent, zorder=1)

# Plot the GLM Data
glm = plt.scatter(points[idx,1], points[idx,0], vmin=0, vmax=600, s=counts[idx]*0.1, c=counts[idx], cmap="jet", zorder=2)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='white', linewidth=0.8, zorder=3)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.5, zorder=4)
gl = ax.gridlines(crs=ccrs.PlateCarree(), color='gray', alpha=1.0, linestyle='--', linewidth=0.25, xlocs=np.arange(-180, 180, 5), ylocs=np.arange(-90, 90, 5), draw_labels=True)
gl.top_labels = False
gl.right_labels = False

# Add the glm colorbar
plt.colorbar(glm, label='GLM Density', extend='max', orientation='vertical', pad=0.05, fraction=0.05)

# Add the img colorbar
plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Add a title
plt.title('GOES-16 IR+GLM', fontweight='bold', fontsize=10, loc='left')
date_ini = str(datetime(int(yyyy),int(mm),int(dd),int(hh),int(mn)) - timedelta(minutes=10))
date_end = str(datetime(int(yyyy),int(mm),int(dd),int(hh),int(mn)))
plt.title(str(date_ini) + " - " + str(date_end), fontsize="10", loc="right")

#---------------------------------------------------------------------------------------------------
# Save the image
plt.savefig(f'{output}/GLM_DENS_{yyyymmddhhmn}.png', bbox_inches='tight', pad_inches=0, dpi=300)

# Show the image
plt.show()
```

Creates the scatter plot based on the coordinates of the occurrences with size and color according to the number of occurrences

Script 18: GLM Density



Script 19: GLM Heatmap

```
# Training: Python and GOES-R Imagery: Script 20 - GLM Density (hotspots)
#-------------------------------------------------------------------------------------------------
# Required modules
from netCDF4 import Dataset                          # Read / Write NetCDF4 files
import matplotlib.pyplot as plt                      # Plotting library
import cartopy, cartopy.crs as ccrs                  # Plot maps
from datetime import timedelta, date, datetime       # Basic Dates and time types
import cartopy, cartopy.crs as ccrs                  # Plot maps
import os                                            # Miscellaneous operating system interfaces
from osgeo import gdal                               # Python bindings for GDAL
import numpy as np                                   # Scientific computing with Python
from matplotlib import cm                            # Colormap handling utilities
from utilities import download_CMI, download_GLM     # Our function for download
from utilities import reproject                      # Our function for reproject
from scipy.ndimage.filters import gaussian_filter    # To make a heatmap
gdal.PushErrorHandler('CPLQuietErrorHandler')        # Ignore GDAL warnings
#-------------------------------------------------------------------------------------------------
# Input and Output directories
input = "Samples"; os.makedirs(input, exist_ok=True)
output = "Output"; os.makedirs(output, exist_ok=True)

# Desired extent
loni = -74
lonf = -34.1
lati = -34.8
latf = 5.5
extent = [loni, lonf, lati, latf] # Min lon, Max lon, Min lat, Max lat

# AMAZON repository information
# https://noaa-goes16.s3.amazonaws.com/index.html
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-RRQPEF'
yyyymmddhhmn = '202102081800'

#-------------------------------------------------------------------------------------------------
# Get the Band 13 Data

# Download the file
file_ir = download_CMI(yyyymmddhhmn, 13, input)
#-------------------------------------------------------------------------------------------------
```

LIBRARIES AND FUNCTIONS

External functions

DOWNLOADS THE IR DATA

Desired extension

Script 19: GLM Heatmap

REPROJECTION (BAND 13)

```
35  #-------------------------------------------------------------------------------------------------------
36  # Variable
37  var = 'CMI'
38
39  # Open the file
40  img = gdal.Open(f'NETCDF:{input}/{file_ir}.nc:' + var)
41
42  # Data Quality Flag (DQF)
43  dqf = gdal.Open(f'NETCDF:{input}/{file_ir}.nc:DQF')
44
45  # Read the header metadata
46  metadata = img.GetMetadata()
47  scale = float(metadata.get(var + '#scale_factor'))
48  offset = float(metadata.get(var + '#add_offset'))
49  undef = float(metadata.get(var + '#_FillValue'))
50  dtime = metadata.get('NC_GLOBAL#time_coverage_start')
51
52  # Load the data
53  ds_cmi = img.ReadAsArray(0, 0, img.RasterXSize, img.RasterYSize).astype(float)
54  ds_dqf = dqf.ReadAsArray(0, 0, dqf.RasterXSize, dqf.RasterYSize).astype(float)
55
56  # Apply the scale, offset and convert to celsius
57  ds_cmi = (ds_cmi * scale + offset) - 273.15
58
59  # Apply NaN's where the quality flag is greater than 1
60  ds_cmi[ds_dqf > 1] = np.nan
61
62  # Reproject the file
63  filename_ret = f'{output}/IR_{yyyymmddhhmn}.nc'
64  reproject(filename_ret, img, ds_cmi, extent, undef)
65
66  # Open the reprojected GOES-R image
67  file = Dataset(filename_ret)
68
69  # Get the pixel values
70  data = file.variables['Band1'][:]
71  #-------------------------------------------------------------------------------------------------------
```

Script 19: GLM Heatmap

GLM 10 MINUTE ACCUMULATION (same procedure)

```
76  #----------------------------------------------------------------------------------------------------------------
77  # Get the GLM Data
78
79  # Initialize arrays for latitude, longitude, and event energy
80  lats = np.array([])
81  lons = np.array([])
82  #----------------------------------------------------------------------------------------------------------------
83  # Initial time and date
84  yyyy = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%Y')
85  mm = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%m')
86  dd = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%d')
87  hh = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%H')
88  mn = datetime.strptime(yyyymmddhhmn, '%Y%m%d%H%M').strftime('%M')
89
90  date_ini = str(datetime(int(yyyy),int(mm),int(dd),int(hh),int(mn)) - timedelta(minutes=10))
91  date_end = str(datetime(int(yyyy),int(mm),int(dd),int(hh),int(mn)))
92
93  # GLM accumulation loop
94  while (date_ini <= date_end):
95
96      # Date structure
97      yyyymmddhhmnss = datetime.strptime(date_ini, '%Y-%m-%d %H:%M:%S').strftime('%Y%m%d%H%M%S')
98
99      # Download the file
100     file_glm = download_GLM(yyyymmddhhmnss, input)
101
102     # Read the file
103     glm = Dataset(f'{input}/{file_glm}.nc')
104
105     # Append lats / longs / event energies
106     lats = np.append(lats, glm.variables['event_lat'][:])
107     lons = np.append(lons, glm.variables['event_lon'][:])
108
109     # Increment the date_ini
110     date_ini = str(datetime.strptime(date_ini, '%Y-%m-%d %H:%M:%S') + timedelta(seconds=20))
111 #----------------------------------------------------------------------------------------------------------------
```

Script 19: GLM Heatmap

```
# Stack and transpose the lat lons
values = np.vstack((lons,lats)).T

values = values[(values[:,0] >= loni) & (values[:,0] <= lonf)]
values = values[(values[:,1] >= lati) & (values[:,1] <= latf)]

from scipy.ndimage.filters import gaussian_filter
heatmap, x, y = np.histogram2d(values[:,0], values[:,1], bins=(120,120), range=[[loni,lonf],[lati,latf]])
heatmap = gaussian_filter(heatmap,1)

print(heatmap.min(), heatmap.max(), heatmap.shape)

#----------------------------------------------------------------------------------------------------
# Choose the plot size (width x height, in inches)
plt.figure(figsize=(10,10))

# Use the Geostationary projection in cartopy
ax = plt.axes(projection=ccrs.PlateCarree())

# Define the image extent
ax.set_extent([extent[0], extent[2], extent[1], extent[3]], ccrs.PlateCarree())

# Define the data extent
img_extent = [extent[0], extent[2], extent[1], extent[3]]

# Plot the image
img = ax.imshow(data, vmin=-50, vmax=80, cmap='gray_r', origin='upper', extent=img_extent)

# Plot the GLM Data
glm = ax.imshow(heatmap.T, vmin=0, vmax=600, origin="lower", cmap="hot", interpolation="nearest", extent=img_extent, alpha=0.6)

# Add coastlines, borders and gridlines
ax.coastlines(resolution='10m', color='white', linewidth=0.8, zorder=3)
ax.add_feature(cartopy.feature.BORDERS, edgecolor='white', linewidth=0.5, zorder=4)
gl = ax.gridlines(crs=ccrs.PlateCarree(), color='gray', alpha=1.0, linestyle='--', linewidth=0.25, xlocs=np.arange(-180, 180, 5), ylocs=np.arange(-90, 90, 5), draw_labels=True)
gl.top_labels = False
gl.right_labels = False

# Add the img colorbar
#plt.colorbar(img, label='Brightness Temperatures (°C)', extend='both', orientation='horizontal', pad=0.05, fraction=0.05)

# Add the glm colorbar
plt.colorbar(glm, label='GLM Density', extend='max', orientation='vertical', pad=0.05, fraction=0.05)

# Add a title
plt.title('GOES-16 IR+GLM', fontweight='bold', fontsize=10, loc='left')
date_ini = str(datetime(int(yyyy),int(mm),int(dd),int(hh),int(mn)) - timedelta(minutes=10))
date_end = str(datetime(int(yyyy),int(mm),int(dd),int(hh),int(mn)))
plt.title(str(date_ini) + " - " + str(date_end), fontsize="10", loc="right")

#----------------------------------------------------------------------------------------------------
# Save the image
plt.savefig(f'{output}/GLM_DENS_HOT_{yyyymmddhhmn}.png', bbox_inches='tight', pad_inches=0, dpi=300)

# Show the image
plt.show()
```

Transpose lats and lons (to get them in pairs) and create the stack

Values of the desired area

Specifying the region

CALCULATES THE HEATMAP

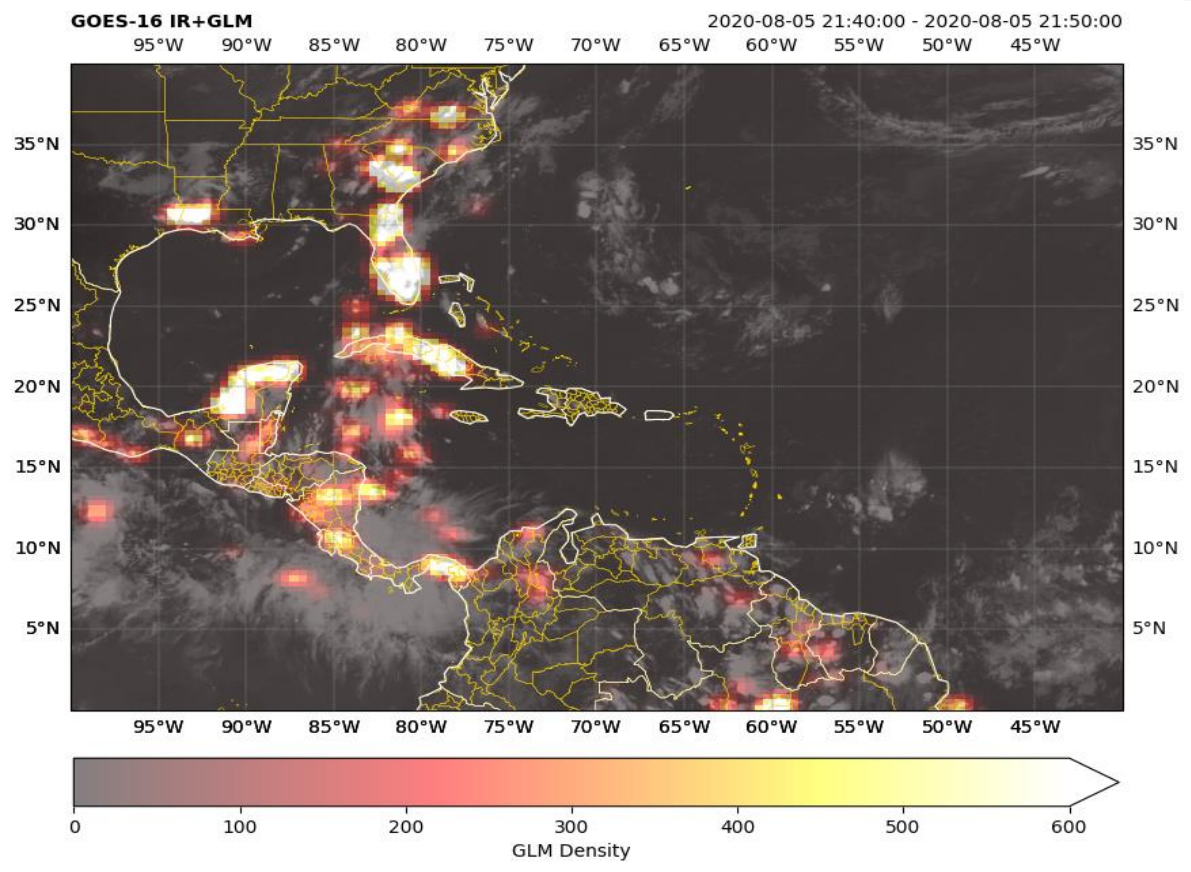
Calculate a 120 x 120 grid with the values

Smooth points with Gaussian filter

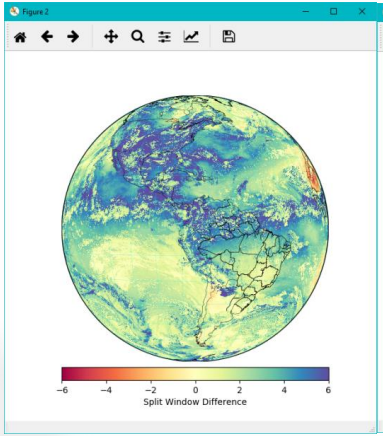
PLOT AND IMAGE GENERATION

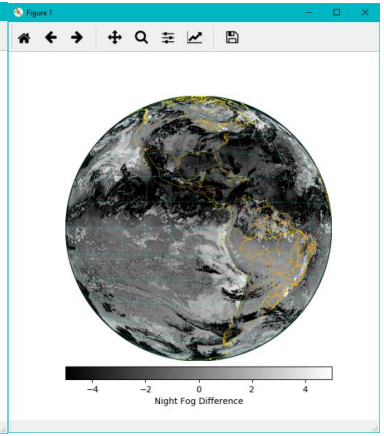
Create the plot "heatmap" according to the densities

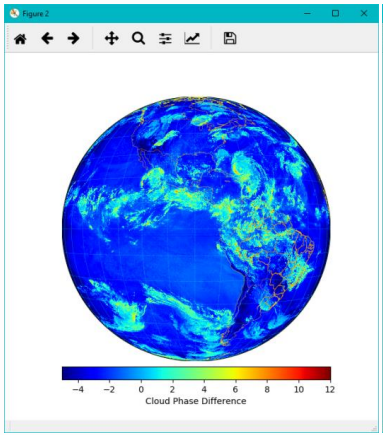
Script 19: GLM Heatmap

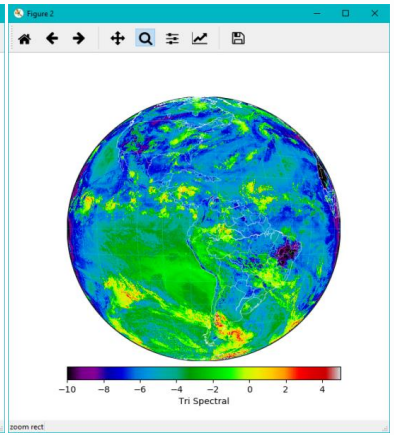


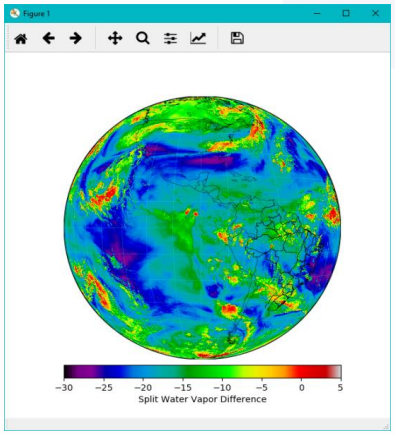
Extra Activities











Extra Activities: SPLIT WINDOW DIFFERENCE

- **Split Window Difference (SWD):**

Quick Guide:

http://cimss.ssec.wisc.edu/goes/OCLOFactSheetPDFs/ABIQuickGuide_SplitWindowDifference.pdf

Calculation: Band 13 - Band 15

Colormap: Spectral

Min. Value: -6

Max. Value: 6

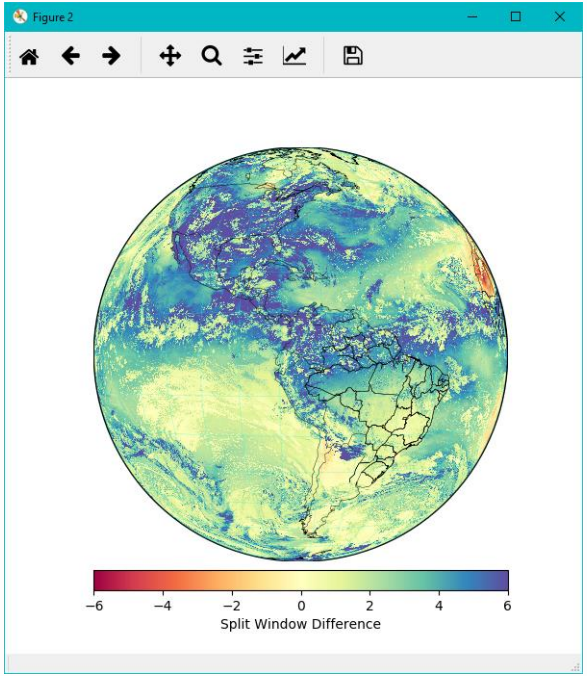
Legend: Split Window Difference

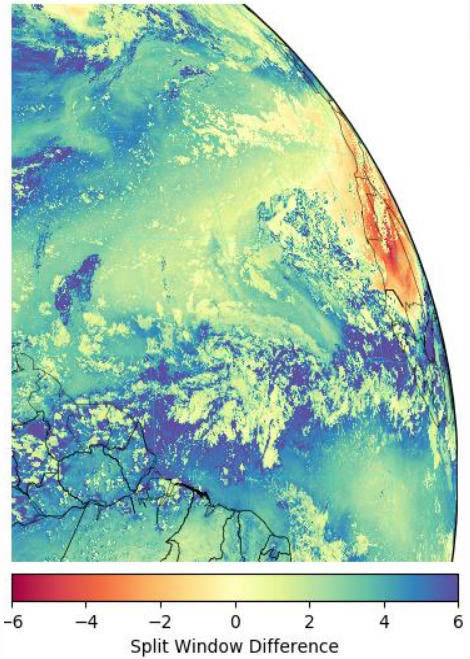
What can we see in the image?
A: Humidity at low levels, atmospheric dust, cirrus.

What do the negative values mean?
A: Atmospheric dust.

And the positives?
A: High humidity.

And the null values?
A: Dry regions.





Extra Activities: NIGHT FOG DIFFERENCE

- **Night Fog Difference:**

Quick Guide:

http://cimss.ssec.wisc.edu/goes/OCLOFactSheetPDFs/ABIQuickGuide_NightFogBTD.pdf

Calculation: Band 13 - Band 07

Colormap: gray

Min. Value: -5

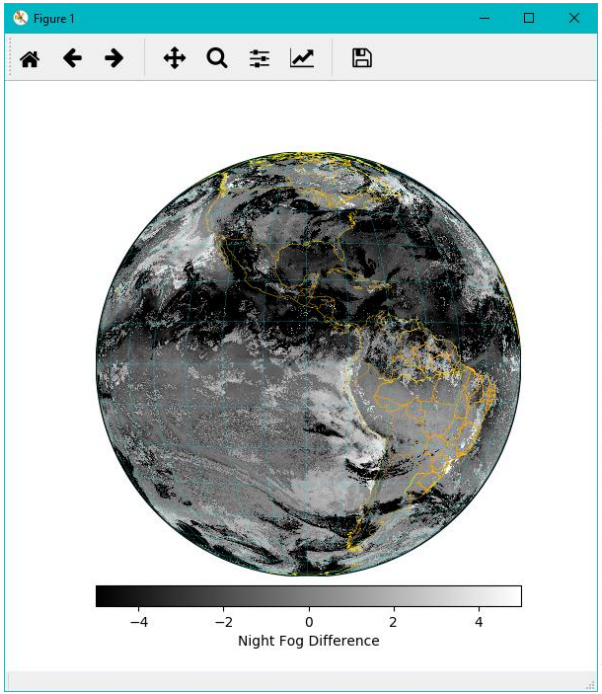
Max. Value: 5

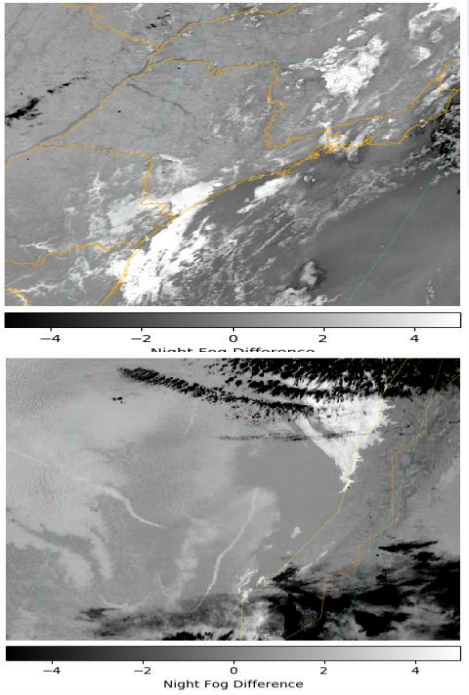
Legend: Night Fog Difference

Let's remember what are the main characteristics that can be observed in this product?

What do the positive values indicate?
A: Fog and stratus.

And the negative values?
A: cirrus clouds and ice tops.





Extra Activities: CLOUD PHASE DIFFERENCE

- **Cloud Phase Difference:**

Quick Guide:

http://cimss.ssec.wisc.edu/goes/OCLOFactSheetPDFs/ABIQuickGuide_G16_CloudPhaseBTD.pdf

Calculation: Band 14 - Band 11

Colormap: jet

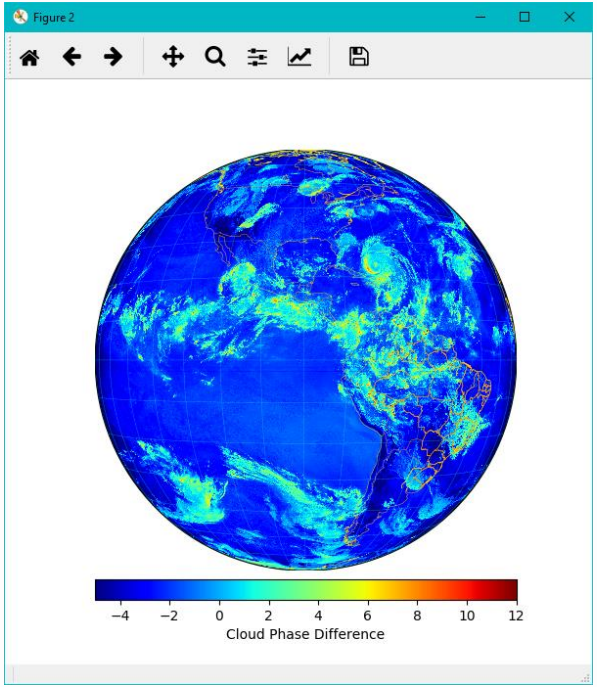
Min. Value: -10

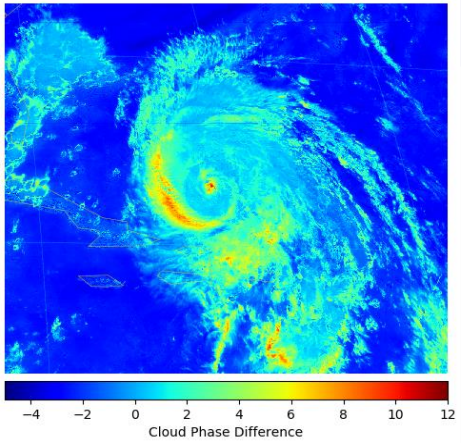
Max. Value: 25

Legend: Cloud Phase Difference

What do the largest positive values (in red) indicate?
A: Small ice crystals.

And the negative values?
A: Clouds with water tops and surface.





Extra Activities: SPLIT WATER VAPOR DIFFERENCE

- **Split Water Vapor Difference:**

Quick Guide:

http://cimss.ssec.wisc.edu/goes/OCLOFactSheetPDFs/ABIQuickGuide_SplitWV_BTDiffv2.pdf

Calculation: Band 08 - Band 10

Colormap: nipy_spectral

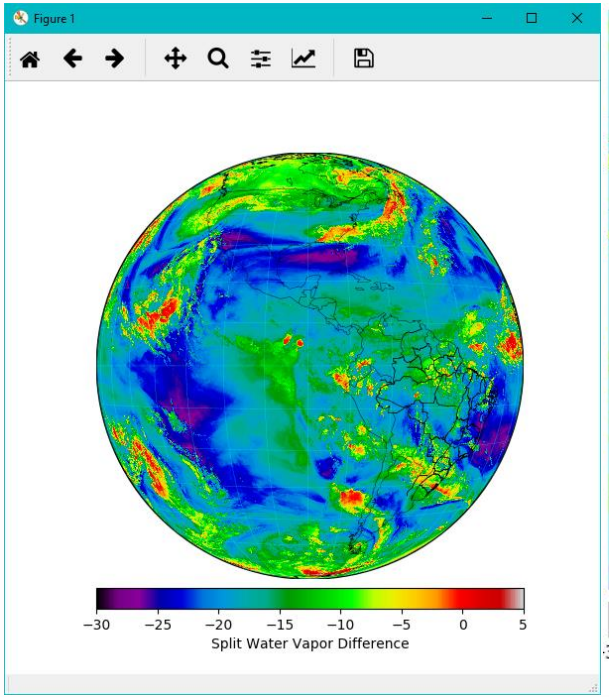
Min. Value: -30

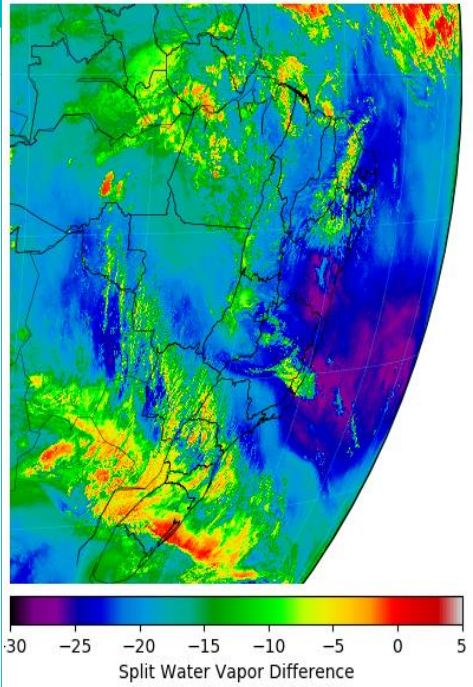
Max. Value: 5

Legend: Split Water Vapor Difference

In this color scale, what do the features in red represent?
A: Higher tops.

And in purple?
A: Humidity at high levels.





Extra Activities: TRI SPECTRAL DIFFERENCE

- **Tri Spectral Difference:**

Quick Guide:

http://cimss.ssec.wisc.edu/goes/OCLOFactSheetPDFs/ABIQuickGuide_SplitWV_BTDiffv2.pdf

Calculation: (11 - 14) - (14 - 15)

Colormap: nipy_spectral

Min. Value: -10

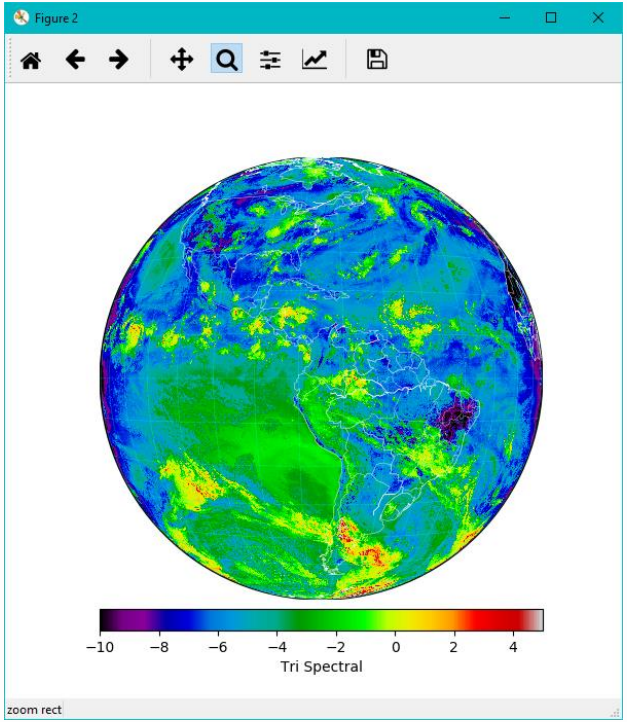
Max. Value: 5

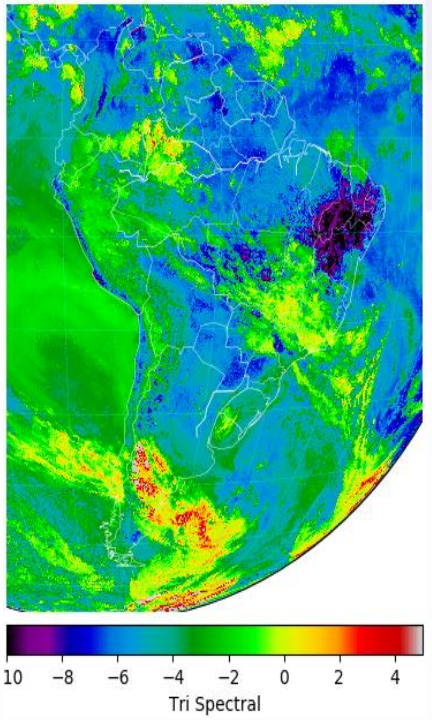
Legend: Tri Spectral Difference

What do the dark colors mean?
A: Difference in emissivity - exposed soil.

And what do the red tones represent?
A: Higher ice tops.

And in green?
A: Lower tops with water.



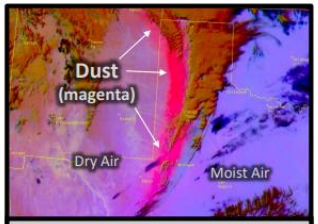


Extra Activities: DUST RGB

Dust RGB Quick Guide

Why is the Dust RGB Imagery Important?

Dust can be hard to see in VIS and IR imagery because it is optically thin, or because it appears similar to other cloud types such as cirrus. The RGB product is able to contrast airborne dust from clouds using band differencing and the IR thermal channel. The IR band differencing allows dust storms to be observed during both daytime and at night. Dust appears pink/magenta during the day and can vary in color at night depending on height. Dust is also distinguishable in the RGB from land surfaces like deserts as well as oceans, given sufficient thickness/density. For cloudy regions this RGB also allows users to infer relative height of the observed cloud top surfaces as well as cloud phase and thicknesses.



Dust RGB from GOES-16 ABI at 2302 UTC, 23 March 2017

Dust RGB Recipe

Color	Band / Band Diff. (μm)	Min – Max Gamma	Physically Relates to…	Small contribution to pixel indicates…	Large Contribution to pixel indicates…
Red	12.3-10.3	-6.7 to 2.6 C 1	Optical depth / cloud thickness	Thin clouds	Thick clouds or dust
Green	11.2-8.4	-0.5 to 20.0 C 2.5	Particle phase	Ice and particles of uniform shape (dust)	Water particles or thin cirrus over deserts
Blue	10.3	-11.95 to 15.55 C 1	Surface temperature	Cold surface	Warm surface

Impact on Operations

Primary Application

Identifying dust: Dust plumes are easily distinguished from surrounding clouds, both day and night.

Height of dust (night only): At night, the resulting dust color changes with height. Low-level dust plumes have a purple/plum color and changes to pink/magenta with height.

Secondary Applications: Cloud height/type analysis, inferring air mass/moisture boundaries (low vs high humidity), volcanic ash (orange/peach)

Limitations

High clouds obscure dust: High cloud cover can obscure dust plumes beneath them and make spatial analysis of the dust more difficult.

Dust thickness typically unknown: Magenta/pink variations in daytime are not indicators of thickness, but rather density. However, very thick dust plumes are purple in both day and night scenes.

Low clouds look like dust over oceans: Marine stratus over the ocean in the tropics appear light purple and can look similar in color to dust, particularly at night.

Contributor: Kevin Fuell NASA SPoRT https://weather.msfc.nasa.gov/sport/

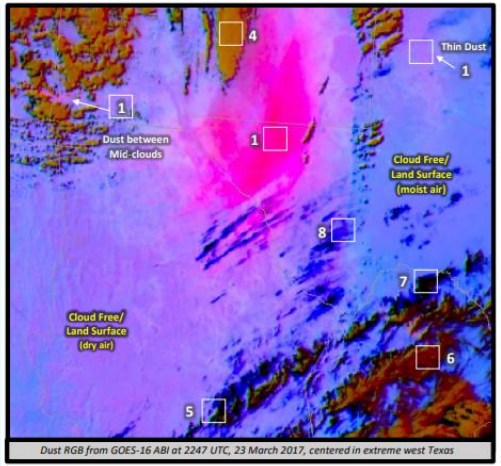
SPORT

Dust RGB Quick Guide

RGB Interpretation

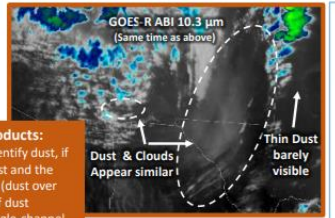
1. Dust plume (day) *(bright magenta, pink)*
   Note: Dust at night becomes purple shades below 3 km
2. Low, water cloud *(light purple, not shown)*
3. Desert surface (day) *(light blue, not shown)*
4. Mid, thick clouds *(tan shades)*
5. Mid, thin cloud *(green)*
6. Cold, thick clouds *(red)*
7. High, thin ice clouds *(black)*
8. Very thin cloud (Over warm surface) *(blue)*

Note: colors may vary diurnally, seasonally, and latitudinally



Dust RGB from GOES-16 ABI at 2247 UTC, 23 March 2017, centered in extreme west Texas

RGB Color Guide



Comparison to other products:
IR imagery can be used to identify dust, if the contrast between the dust and the background is bright enough (dust over the hot desert). Detection of dust becomes more difficult in single-channel IR imagery at night or over oceans.

Resources

UCAR/COMET
Multispectral Satellite Applications: RGB Products Explained.

NASA/SPoRT
Dust RGB micro-lesson:

EUMETrain
RGB Interpretation Guide

Hyperlinks not available when viewing material in AIR Tool

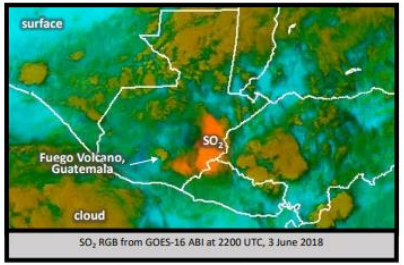
Extra Activities: SO2 RGB

SO_2 RGB Quick Guide

*interpretation still under investigation

Why is the SO_2 RGB Important?

Sulfur dioxide (SO_2) is a gas commonly released into the atmosphere during volcanic eruptions. In high concentrations it is toxic to humans and has considerable environmental effects, including volcanic smog, acid rain, and is harmful to vegetation downwind of the eruption. The SO_2 RGB product can be used to detect and monitor large sulfur dioxide emissions from volcanoes, as well industrial facilities such as power plants.



SO_2 RGB from GOES-16 ABI at 2200 UTC, 3 June 2018

SO_2 RGB Recipe

Color	Band / Band Diff. (μm)	Min to Max Gamma	Physically Relates to...	Small contribution to pixel indicates...	Large Contribution to pixel indicates...
Red	6.95 – 7.34 Ch 9 – Ch 10	-4.0 to 2.0 °C 1	Vertical water vapor difference, presence of SO_2	Low-levels, relatively drier atmosphere	SO_2 is present in mid- and high-levels of the atmosphere
Green	10.35 – 8.50 Ch 13 – Ch 11	-4.0 to 5.0 °C 1	Moisture, stability, particle size and phase, presence of ash and SO_2	Small crystal ice cloud	Low- and mid-level cloud, volcanic ash and/or SO_2
Blue	10.35 Ch 13	-30.1 to 29.8 °C 1	Cloud top or surface temperature	Mid- and high-levels in the atmosphere	Surface or low-levels in the atmosphere

Impact on Operations

Primary Application

Detection of sulfur dioxide: Stronger absorption of SO_2 is found in Band 10 (7.34 μm, a Water Vapor channel) and weaker SO_2 absorption is found in Band 11 (8.50 μm, an Infrared channel). These bands are differenced with similar water vapor and infrared channels in the red and green components respectively to highlight the presence of SO_2.

Limitations

Distinguishing SO_2 from ash and water vapor: Volcanic eruptions are often composed of ash and a mixture of gases, including water vapor and SO_2. Distinguishing the components can be a challenge, and water vapor can mask the ash and aerosol signals.

Low-level clouds: In the RGB, the light green color of low-level SO_2 is a similar color to that of low-level cloud.

Upper-level clouds: Thick opaque upper-level clouds can mask the SO_2 signal below.

Contributors: Bernie Connell, Erin Dagg CSU/CIRA https://www.cira.colostate.edu

CIRA

SO_2 RGB Quick Guide

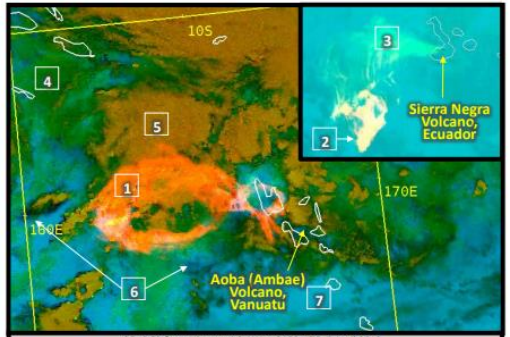
*interpretation still under investigation

RGB Interpretation

1. Upper-level SO_2, cold background *(orange)*
2. Upper-level SO_2, warm background *(light yellow)*
3. Low-level SO_2, *(light green)*
4. Low- and mid-level cloud *(green)*
5. Convective clouds *(tan)*
6. Thin, high level cloud *(dark blue)*
7. Ocean/land surface *(light blue)*

Note: colors may vary diurnally, seasonally, and latitudinally

This RGB composite was developed by the Japan Meteorological Agency (JMA) for Himawari-8. Interpretation is still under investigation.



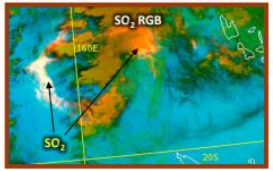
SO_2 RGB from Himawari-8 AHI at 2100 UTC, 5 April 2018
SO_2 RGB from GOES-16 ABI at 0130 UTC, 27 June 2018 (inset, upper right)

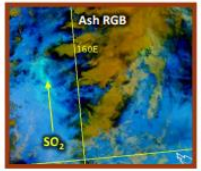
RGB Color Guide

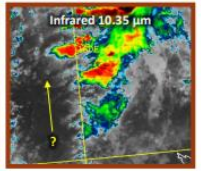
Comparison to Ash RGB and 10.35 μm Infrared:

The SO_2 RGB is a modified version of the Ash RGB recipe, tuned to better detect sulfur dioxide emissions. For the RED component, in place of the longwave difference in the Ash RGB, band 10 is differenced to take advantage of the strong SO_2 absorption region near 7.34 μm. For the GREEN component, similar channels are used in both RGBs but with different ranges, to take advantage of the lesser SO_2 absorption region near 8.50 μm. The BLUE component is the same for both RGBs.

The three images below are from the eruption of the Aoba Volcano shown above, but a few hours later at 0250 UTC, 6 April 2018. In the SO_2 RGB, SO_2 over (cold) cloud is orange, while SO_2 over (warm) ocean is white; in the Ash RGB, SO_2 is aqua green over ocean; in the IR 10.35 μm imagery, SO_2 is not discernable.





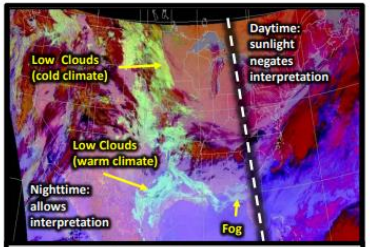


Extra Activities: NIGHT MICROPHYSICS RGB

Nighttime Microphysics RGB Quick Guide

Why is the Nighttime Microphysics RGB Imagery Important?

The distinction between low clouds and fog in satellite imagery is often a challenge. While the difference in the 10.4 and 3.9 µm channels has been a regularly applied product to meet aviation forecast needs, the Nighttime Microphysics (NtMicro) RGB adds another channel difference (12.4- 10.4 µm) as a proxy to cloud thickness and repeats the use of the 10.4 µm thermal channel to enhance areas of warm (i.e. low) clouds where fog is more likely. The NtMicro RGB is also an efficient tool to quickly identify other cloud types in the mid and upper atmosphere.



NtMicro RGB from GOES-16 on 22 March 2017 at 1207 UTC

NtMicro RGB Recipe

(Note: this applies best to opaque clouds. Semi-transparent clouds are influenced by underlying surface)

Color	Band / Band Diff. (µm)	Min – Max Gamma	Physically Relates to…	Small contribution to pixel indicates…	Large Contribution to pixel indicates…
Red	12.4 – 10.4	-6.7 – 2.6 C 1	Optical Depth	Thin clouds	Thick clouds
Green	10.4 – 3.9	-3.1 – 5.2 C 1	Particle Phase and Size	Ice particles; surface (cloud free)	Water clouds with small particles
Blue	10.4	-29.6 – 19.5 C 1	Temperature of surface	Cold Surface	Warm surface

Impact on Operations

Primary Application

Low clouds & fog analysis: Low clouds and fog are aqua in warm regimes, but become more yellow to light green in cold regimes (i.e. decrease in blue component).

Differentiate fog from low clouds: Fog tends to appear "washed out" compared to low clouds. So, look for fog to have a less bright or near gray coloring.

Efficient Cloud Analysis: The multi-channel approach of the RGB allows for easy and quick discrimination of cloud types across the imagery.

Secondary Applications: Cloud height and phase, fire hot spots, moisture boundaries

Limitations

Nighttime only application: The shortwave IR band is impacted by solar reflectance during the day which impacts the 10.4 – 3.9 difference relationship.

Thin fog blends with surface: Thin radiation fog is semi-transparent allowing surface emissions to impact pixel color. Fog often has less blue than low clouds.

Variable land/surface coloring: The color of cloud free regions will vary depending on their temperature, surface type, and the column moisture.

Shortwave IR noise in extreme cold: Speckled yellow pixels appear in very cold clouds (~<-30°C)

Contributor: Kevin Fuell NASA SPoRT https://weather.msfc.nasa.gov/sport/

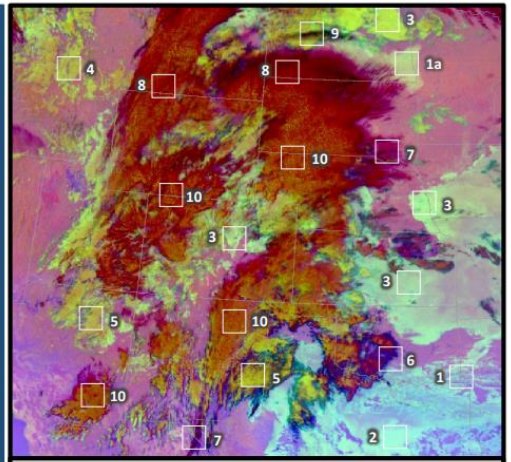
SPORT

Nighttime Microphysics RGB Quick Guide

RGB Interpretation

1	Fog *(dull aqua to gray)*
1a	Fog – cold regime *(dull yellow-green to gray)*
2	Very low, warm cloud *(aqua)*
3	Low, cool, cloud *(bright green)*
4	Mid water cloud *(light green)*
5	Mid, thick, water/ ice cloud *(tan)*
6	High, thin, ice cloud *(dark blue)*
7	High, very thin, ice cloud *(purple)*
8	High, thick cloud *(dark red)*
9	High, opaque cirrus cloud *(near black)*
10	High, thick, very cold cloud *(red/yellow, noisy)*

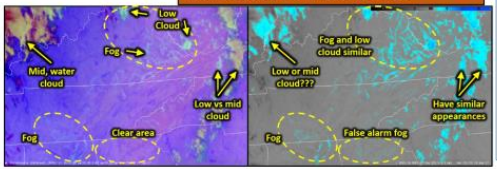
Note:, colors may vary diurnally, seasonally, and latitudinally



NtMicro RGB from GOES-16 ABI at 1127 UTC, 28 March 2017.

Comparison to Other Products (below)

The NtMicro RGB (left) helps to distinguish fog from clouds and "false alarm" features seen in the legacy "Fog" or 10.3-3.9 µm channel difference (right). Recall the 10.3-3.9 µm is also in the RGB.



Resources

UCAR/COMET
Multispectral Satellite Applications: RGB Products Explained

NASA/SPoRT
Nighttime Microphysics RGB Module

EUMETrain
RGB Interpretation Guide

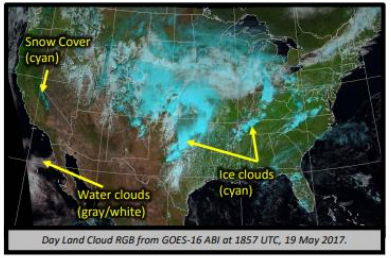
Hyperlinks not available when viewing material in AIR Tool

Extra Activities: DAY LAND CLOUD RGB

Day Land Cloud RGB Quick Guide

Why is the Day Land Cloud RGB Imagery Important?

The Day Land Cloud RGB is the same as the Natural Color RGB developed by EUMETSAT. This RGB is useful for discriminating water/ice clouds to identify low/high clouds. High ice clouds, snow, and sea ice appear cyan while low water clouds appear dull grey or white. Land/Ocean surfaces are in expected colors (but not true color). This imagery can also be used to assess vegetation and detect land surface changes where vegetation appears green and soil, inactive vegetation, and rock appear brown to dark gray.



Day Land Cloud RGB from GOES-16 ABI at 1857 UTC, 19 May 2017.

Day Land Cloud RGB Recipe

Color	Band / Band Diff. (µm)	Min – Max Gamma	Physically Relates to...	Small contribution to pixel indicates...	Large contribution to pixel indicates...
Red	1.6	0 – 97.5 % 1	Reflectance of clouds & surfaces	Ice or large particle clouds, water, snow/ice, sea ice	Water Clouds with small drops, and desert
Green	0.86	0 – 108.6 % 1		Water, inactive vegetation, bare soil	Clouds, vegetation, and snow/ice
Blue	0.64	0 – 100.0 % 1		Thin cloud, water, vegetation, bare soil	Thick clouds and snow/ice

Impact on Operations

Primary Application

Surface and atmospheric features: Discern high ice clouds from low water clouds, snow/ice cover, land surface features.

High ice clouds, snow, and sea ice are cyan: Ice strongly absorbs in the near-IR 1.6 µm band, leading to little red contribution (resulting in cyan) and notable contrast with water clouds (white/gray).

Low water clouds are gray to dull white: Water clouds with small droplets (i.e. fog) have a high reflectance in all three bands.

Land surface types are a 'Natural' color: Green vegetation, brown deserts and burn scars.

Limitations

Daytime only application: The RGB relies on solar reflectance from visible and near-IR channels.

Sun glint complicates water scenes: Water will appear grey to white as the sun moves overhead and reflects sunlight toward the satellite.

Distinguishing snow cover and high ice clouds: Snow and ice clouds are bright cyan in the RGB, but geographic features and/or cloud motion may help to differentiate between the two.

Thin cirrus/cirrostratus: These clouds are semi-transparent; hence, difficult to detect with the visible channels.

Dust appears similar color as bare land.

Contributor: Dr. Emily Berndt NASA SPoRT https://weather.msfc.nasa.gov/sport/

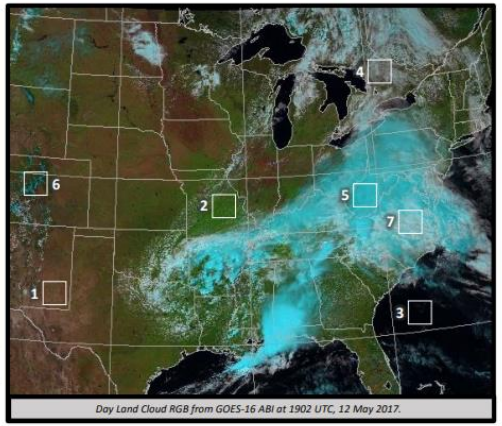
SPoRT

Day Land Cloud RGB Quick Guide

RGB Interpretation

1. Bare land or inactive vegetation *(shades of brown)*
2. Vegetation *(shades of green)*
3. Water bodies or flooded areas *(dark blue to black)*
4. Low water clouds *(shades of gray and white)*
5. High ice clouds *(bright cyan)*
6. Snow *(dark to bright cyan)*
7. Mid mixed phase clouds *(gray shades of cyan)*

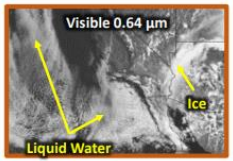
Note: colors may vary diurnally, seasonally, and latitudinally

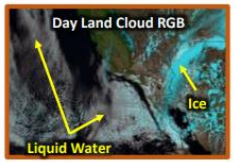


Day Land Cloud RGB from GOES-16 ABI at 1902 UTC, 12 May 2017.

RGB Color Guide

Comparison to other products: Cloud particle phase is not easy to discern in a single channel 0.64 µm visible satellite image. The Day Land Cloud RGB can distinguish between clouds which are primarily composed of ice crystals (bright cyan) and those primarily composed of liquid water (gray and dull white).





Resources

UCAR/COMET
Multispectral Satellite Applications: RGB Products Explained.

NASA/SPoRT
Applications Library

EUMETrain
RGB Interpretation Guide

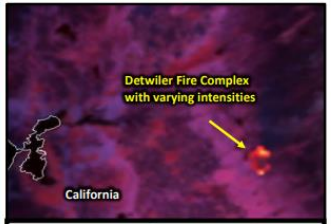
Hyperlinks not available when viewing material in AIR Tool

Extra Activities: FIRE TEMPERATURE RGB

Fire Temperature RGB Quick Guide

Why is the Fire Temperature RGB imagery Important?

This RGB allows the user to identify where the most intense fires are occurring and differentiate these from "cooler" fires. The RGB takes advantage of the fact that from 3.9 µm to shorter wavelengths, background solar radiation and surface reflectance increases. This means that fires need to be more intense in order to be detected by the 2.2 and 1.6 µm bands, as more intense fires emit more radiation at these wavelengths. Therefore, small/"cool" fires will only show up at 3.9 µm and appear red while increases in fire intensity cause greater contributions of the other channels resulting in white very intense fires.



Fire Temperature RGB from GOES-16 on 19 July 2017 at 1957 UTC

Fire Temperature RGB Recipe

Color	Band / Band Diff. (µm)	Min – Max Gamma	Physically Relates to…	Small contribution to pixel indicates…	Large Contribution to pixel indicates… (saturated)
Red	3.9	0 to 60 C 0.4	Cloud top phase and temperature	Cold land surfaces, water, snow, clouds	Hot land surface, (Low fire temperature)
Green	2.2	0 to 100 % 1	Particle size / land type	Large ice/water particles, snow, oceans	Small ice/water particles, (Medium fire temperature)
Blue	1.6	0 to 75 % 1	Particle size / land type	Ice clouds with large particles, snow, oceans	Water clouds, (High fire temperature)

Impact on Operations

Primary Application

Fire hotspot locations are detected: The saturation brightness temperature of the shortwave-IR 3.9 µm channel is low, around 500 K (i.e. relatively low intensity fire). Therefore, "hotspots" of wild fires look red in RGB.

Fire intensity can be analyzed: High intensity fires are near a maximum of 1400 K and this is near the peak emission detection (i.e. saturation) of the 1.6 µm channel. Therefore, active fires in the RGB transition from red to yellow to white as intensity increases and near-IR channels become saturated.

Limitations

Cloud cover blocks view of fire: The fires will only be visible in the RGB in clear sky areas.

Cloud features/type have less details: While water vs. ice clouds can be identified, other RGB products are better at displaying cloud features.

Daytime only application for clouds: The reflectance from clouds are not available at night in the near-IR bands used in the RGB.

False "red" fires due to land type: Some surfaces in arid, dry regions are highly emissive at 3.9 µm and will appear red but they are not on fire.

Contributor: NASA SPoRT https://weather.msfc.nasa.gov/sport/

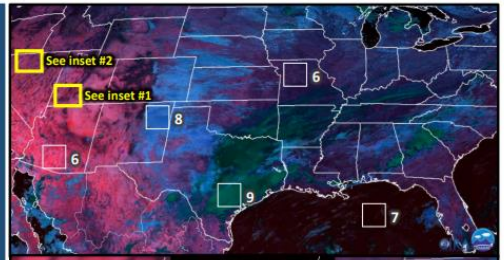
SPoRT

Fire Temperature RGB Quick Guide

RGB Interpretation

1	"Warm" Fire *(red)*
2	"Very Warm" Fire *(orange)*
3	"Hot" Fire *(yellow)*
4	"Very Hot" Fire *(near white)*
5	Burn Scars *(shades of maroon)*
6	Clear Sky: Land *(purples to pinks)*
7	Clear Sky: Water/Snow/Night *(near black)*
8	Water Clouds *(shades of blue)*
9	Ice Clouds *(shades of green)*

Note: colors may vary diurnally, seasonally, and latitudinally



Fire Temperature RGB from GOES-16 AIB at 1807 UTC, 24 June 2017 (above). Inset #1 provides zoomed-in view of active fire at 1952 UTC over southwest Utah for both day (left) and night (center); Inset #2 is zoomed-in view over northern Nevada for 25 July 2017 showing a burn scar.

RGB Color Guide

Comparison to Other Products

The True Color RGB shows smoke from the fires, but does not distinguish the intensity like the Fire Temp. RGB. While missing the smoke, the Fire Temp. RGB provides insight to active fire location and behavior.

Resources

UCAR/COMET
Multispectral Satellite Applications: RGB Products Explained.

VISIT Program
Satellite Chat: Ft. McMurray Fires

CIRA Blog
"Fire" related posts

Hyperlinks not available when viewing material in AIR Tool

GOES-16 Channels (ABI Sensor)

ABI Band	Central Wavelength (µm)	Wavelength Range (µm)	Resolution (km)	Spectral range	Descriptive Name
1	0.47	0.45 - 0.49	1	Visible	Blue
2	0.64	0.68 - 0.68	0.5	Visible	Red
3	0.86	0.847 - 0.882	1	Near Infrared	Vegetation
4	1.37	1.366 - 1.380	2	Near Infrared	Cirrus
5	1.6	1.59 - 1.63	1	Near Infrared	Snow/Ice
6	2.2	2.22 - 2.27	2	Near Infrared	Cloud Particle Size
7	3.9	3.80 - 3.99	2	Shortwave Infrared	Shortwave window
8	6.2	5.79 - 6.59	2	Midwave Infrared	Upper-level water vapor
9	6.9	6.72 - 7.14	2	Midwave Infrared	Midlevel water vapor
10	7.3	7.24 - 7.43	2	Midwave Infrared	Lower / midlevel watervapor
11	8.4	8.23 - 8.66	2	Longwave Infrared	Cloud-top phase
12	9.6	9.42 - 9.80	2	Longwave Infrared	Ozone
13	10.3	10.18 - 10.48	2	Longwave Infrared	Clean longwave window
14	11.2	10.82 - 11.60	2	Longwave Infrared	Longwave window
15	12.3	11.83 - 12.75	2	Longwave Infrared	Dirty longwave window
16	13.3	12.99-13.56	2	Longwave Infrared	CO2

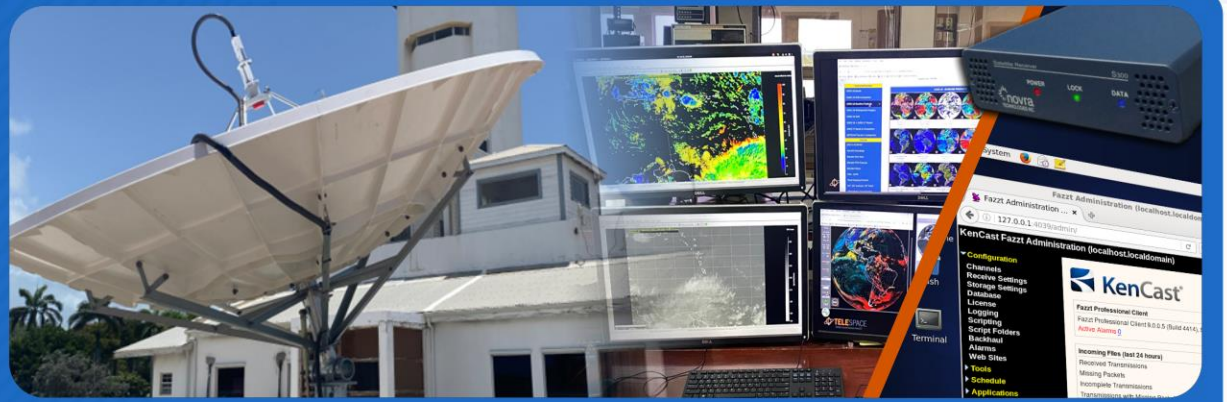
GEONETCast-Americas Training for the Eastern Caribbean States

Day 1 - May 3rd

Session 5:

Hands-on
GOES-R Data Processing

THANK YOU! QUESTIONS?



Diego Souza
diego.souza@inpe.br

DISSM - Meteorological Satellites and Sensors' Division
CGCT - General Coordination of Earth Sciences
INPE - National Institute for Space Research

This is made possible by the generous support of the American people through the United States Agency for International Development (USAID)