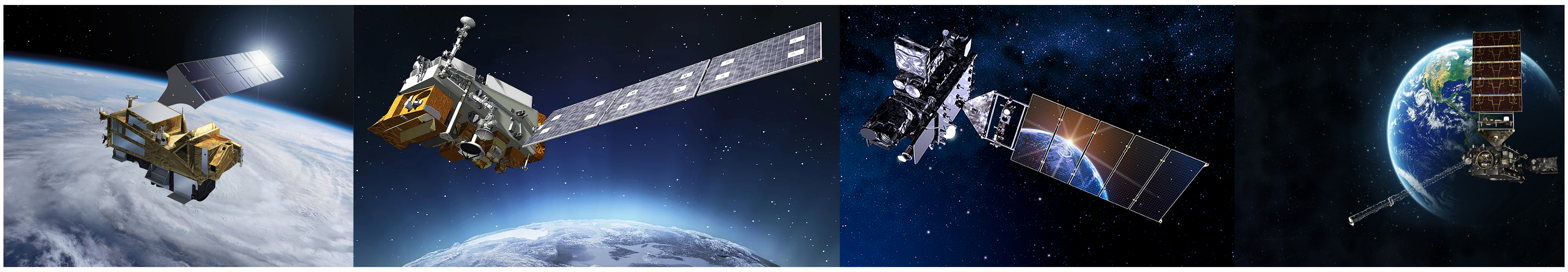
STAR Atmospheric Composition Product Training

Featuring Aerosol, Fire, and Trace Gas Satellite Products from ABI, VIIRS, TROPOMI & TEMPO



Python Short Demo: Calculate Latitude and Longitude from GOES Imager Projection (ABI Fixed Grid) Information

This Python function can be used to calculate latitude and longitude from the GOES Imager Projection information in any ABI L1b or L2 data file.

Please acknowledge the NOAA/NESDIS/STAR Aerosols and Atmospheric Composition Science Team if using any of this code in your work/research!

```
# Calculate latitude and longitude from GOES ABI fixed grid projection data
# GOES ABI fixed grid projection is a map projection relative to the GOES satellite
# Units: latitude in °N (°S < 0), longitude in °E (°W < 0)
# See GOES-R Product User Guide (PUG) Volume 5 (L2 products) Section 4.2.8 for details & example of calculations
# "file_id" is an ABI L1b or L2 .nc file opened using the netCDF4 library

def calculate_degrees(file_id):

    # Read in GOES ABI fixed grid projection variables and constants
    x_coordinate_1d = file_id.variables['x'][:]  # E/W scanning angle in radians
    y_coordinate_1d = file_id.variables['y'][:]  # N/S elevation angle in radians
    projection_info = file_id.variables['goes_imager_projection']
    lon_origin = projection_info.longitude_of_projection_origin
    H = projection_info.perspective_point_height+projection_info.semi_major_axis
    r_eq = projection_info.semi_major_axis
    r_pol = projection_info.semi_minor_axis

    # Create 2D coordinate matrices from 1D coordinate vectors
    x_coordinate_2d, y_coordinate_2d = np.meshgrid(x_coordinate_1d, y_coordinate_1d)

    # Equations to calculate latitude and longitude
    lambda_0 = (lon_origin*np.pi)/180.0
    a_var = np.power(np.sin(x_coordinate_2d),2.0) +
(np.power(np.cos(x_coordinate_2d),2.0)*(np.power(np.cos(y_coordinate_2d),2.0)+(((r_eq*r_eq)/
(r_pol*r_pol))*np.power(np.sin(y_coordinate_2d),2.0))))
    b_var = -2.0*H*np.cos(x_coordinate_2d)*np.cos(y_coordinate_2d)
    c_var = (H**2.0)-(r_eq**2.0)
    r_s = (-1.0*b_var - np.sqrt((b_var**2)-(4.0*a_var*c_var)))/(2.0*a_var)
    s_x = r_s*np.cos(x_coordinate_2d)*np.cos(y_coordinate_2d)
    s_y = - r_s*np.sin(x_coordinate_2d)
    s_z = r_s*np.cos(x_coordinate_2d)*np.sin(y_coordinate_2d)

    # Ignore numpy errors for sqrt of negative number; occurs for GOES-16 ABI CONUS sector data
    np.seterr(all='ignore')

    abi_lat = (180.0/np.pi)*(np.arctan(((r_eq*r_eq)/(r_pol*r_pol))*((s_z/np.sqrt(((H-s_x)*(H-s_x))+(s_y*s_y))))))
    abi_lon = (lambda_0 - np.arctan(s_y/(H-s_x)))*(180.0/np.pi)

    return abi_lat, abi_lon
```

Demonstration of how to use the function: The user enters the "directory_path" (directory where the ABI file is located) and "file_name" (name of ABI file) parameter variables. In this example, the directory is assumed to be the current working directory (cwd), set using the *pathlib* module, and the ABI file is a GOES-17 ABI Mesoscale 1 sector FDC (fire/hot spot characterization) file for Aug 17, 2021 (Julian day 229) at 14:00 UTC.

```
# Import Python packages

# Library to work with netCDF files
from netCDF4 import Dataset

# Library to perform array operations
import numpy as np

# Module to set filesystem paths appropriate for user's operating system
from pathlib import Path
```

```
# Open an ABI netCDF4 data file

# Enter directory and file name for ABI data file
directory_path = Path.cwd()  # Current working directory
file_name = 'OR_ABI-L2-FDCM1-M6_G17_s20212291400255_e20212291400312_c20212291400457.nc'
file_path = directory_path / file_name

# Open the file using the netCDF4 library
file_id = Dataset(file_path)
```

```
# Print arrays of calculated latitude and longitude

# Call function to calculate latitude and longitude from GOES ABI fixed grid projection data
abi_lat, abi_lon = calculate_degrees(file_id)

# Print latitude array
print(abi_lat)

# Print max and min of latitude data to check data range
print('The maximum latitude value is', np.max(abi_lat), 'degrees')
print('The minimum latitude value is', np.min(abi_lat), 'degrees')

# Print longitude array
print(abi_lon)

# Print max and min of longitude data to check data range
print('The maximum longitude value is', np.max(abi_lon), 'degrees')
print('The minimum longitude value is', np.min(abi_lon), 'degrees')
```

```
[[49.4893684387207 49.49054718017578 49.491451263427734 ...
  50.358699798583984 50.36152267456055 50.363651275634766]
 [49.45414352416992 49.45476531982422 49.455787658691406 ...
  50.32088088989258 50.32374572753906 50.325984954833984]
 [49.418357849121094 49.419395446777344 49.42030715942383 ...
  50.28329849243164 50.28595733642578 50.28821563720703]
 ...
 [35.06407928466797 35.06440734863281 35.064842224121094 ...
  35.404842376708984 35.40587615966797 35.40659713745117]
 [35.03932189941406 35.039669036865234 35.04008483886719 ...
  35.379573822021484 35.380611419677734 35.381465911865234]
 [35.01442337036133 35.014862060546875 35.01518630981445 ...
  35.354244232177734 35.355262756347656 35.356136322021484]]
The maximum latitude value is 50.36365 degrees
The minimum latitude value is 35.014423 degrees


[[-124.71633  -124.6854   -124.65459  ... -107.962685 -107.92437
  -107.88682 ]
 [-124.72633  -124.69568  -124.66485  ... -107.991325 -107.95301
  -107.915375]
 [-124.73658  -124.70576  -124.67501  ... -108.019646 -107.98162
  -107.94401 ]
 ...
 [-127.58433  -127.561066 -127.53775  ... -115.499535 -115.47376
  -115.44823 ]
 [-127.58766  -127.564384 -127.54109  ... -115.507774 -115.48201
  -115.456375]
 [-127.591034 -127.56774  -127.54448  ... -115.51607  -115.49032
  -115.464676]]
The maximum longitude value is -107.88682 degrees
The minimum longitude value is -127.591034 degrees
```