```
from pprint import pprint
from PIL import Image
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import glob
import xml.etree.ElementTree as ET

# Distance betwen moon and sun in AUs
D_AU = 1

# Solar irradiance
F0 = np.array([
  136.1259307,
  129.8781929,
  125.1457188,
  120.4566749,
  115.2187742,
  110.7989129,
  105.971862,
  102.2853476,
  98.83159112,
  95.00990644,
  91.72241746,
  88.63043389,
  85.44216416,
  83.09659958,
  80.7461688,
  77.99745659,
  75.43755054,
  72.53298554,
  70.30310472,
  67.71506702,
  65.53063581,
  63.51647332,
  61.49193881,
  59.39769145,
  57.24811211,
  55.56974549,
  53.96628612,
  52.39858882,
  50.94286582,
  49.55873832,
  47.99340839,
  46.35543865,
  45.11640663,
  43.75374359,
  42.46741487,
  41.1950428,
```

```
  39.93375405,
  38.7480202,
  37.63257797,
  36.52968828,
  35.48372942,
  34.51571377,
  33.5041102,
  32.62925225,
  31.80035805,
  30.98128654,
  30.16775831,
  29.32709974,
  28.56074168,
  27.8298174,
  27.0453247,
  26.30808675,
  25.51810387,
  24.75010497,
  24.00573968,
  23.24760491,
  22.51761852,
  21.78398871,
  21.06792047,
  20.39822233,
  19.7458807,
  19.11661541,
  18.44061437,
  17.83250529,
  17.26068394,
  16.65126453,
  16.11545704,
  15.61912435,
  15.1210474,
  14.62910738,
  14.16359209,
  13.72237684,
  13.31430194,
  12.94713935,
  12.56233275,
  12.18239943,
  11.79722098,
  11.38810049,
  11.04636914,
  10.71621297,
  10.38904988,
  10.06620698,
  9.753295821,
  9.46418631,
  9.201075776,
```

```
  8.960974818,
  8.732115834,
  8.508712424,
  8.28861478,
  8.070068082,
  7.850866176,
  7.629585176,
  7.417896212,
  7.21399149,
  7.014245694,
  6.819995994,
  6.637200746,
  6.463212542,
  6.291676014,
  6.122400975,
  5.952327234,
  5.785907458,
  5.631916792,
  5.48221029,
  5.338864421,
  5.183886388,
  5.053359936,
  4.941756508,
  4.835098184,
  4.719922707,
  4.619729215,
  4.511137419,
  4.407240202,
  4.306184976,
  4.210413629,
  4.117013411,
  4.012368768,
  3.918726643,
  3.824014432,
  3.725826304,
  3.646586732,
  3.564719937,
  3.488199195,
  3.397463341,
  3.32250234,
  3.262984894,
  3.190955311,
  3.122692223,
  3.056477464,
  2.991274348,
  2.926566072,
  2.864612339,
  2.802940836,
  2.743157021,
```

```
  2.685370618,
  2.628641884,
  2.571929704,
  2.517226294,
  2.465127643,
  2.414375576,
  2.365285234,
  2.316701141,
  2.26923212,
  2.222564505,
  2.178496705,
  2.135290025,
  2.092826765,
  2.051565701,
  2.010893773,
  1.971470582,
  1.932492639,
  1.893925453,
  1.853239032,
  1.814419696,
  1.780829606,
  1.751599126,
  1.715922793,
  1.680125966,
  1.647791753,
  1.621454182,
  1.593640531,
  1.560460708,
  1.532378246,
  1.507178355,
  1.480349348,
  1.454525518,
  1.426003985,
  1.40026592,
  1.376814112,
  1.351395724,
  1.327241488,
  1.303320437,
  1.279240078,
  1.255715058,
  1.232621586,
  1.209534773,
  1.186777237,
  1.163774025,
  1.141839466,
  1.121354795,
  1.102697582,
  1.084984542,
  1.06779729,
```

```
  1.050654559,
  1.034116451,
  1.018239678,
  1.003106371,
  0.987228033,
  0.971082552,
  0.954532246,
  0.938549781,
  0.922761605,
  0.90746215,
  0.892772367,
  0.876952832,
  0.86169586,
  0.846904043,
  0.832961745,
  0.820193322,
  0.808495532,
  0.796418017,
  0.784036511,
  0.771772032,
  0.760169612,
  0.74902997,
  0.737997332,
  0.727055348,
  0.716477866,
  0.704633464,
  0.691770452,
  0.681177697,
  0.668685204,
  0.6563386,
  0.643784606,
  0.630929839,
  0.618670348,
  0.605670184,
  0.593191697,
  0.582320158,
  0.571630629,
  0.561438106,
  0.551831735,
  0.542986524,
  0.534529199,
  0.526707332,
  0.518722109,
  0.511109087,
  0.50373316,
  0.496221855,
  0.489530981,
  0.482582186,
  0.475974536,
```

```
  0.469794569,
  0.463575699,
  0.458286546,
  0.452850271,
  0.447197638,
  0.441572082,
  0.43580287,
  0.430755766,
  0.425717099,
  0.420589447,
  0.41588213,
  0.410468477,
  0.405233536,
  0.399887123,
  0.394668014,
  0.389642973,
  0.384580319,
  0.379611238,
  0.374544041,
  0.369613524,
  0.364863435,
  0.360132602,
  0.355533758,
  0.350967069,
]).reshape(256,1,1)

class utils:
    @staticmethod
    def _extract_sequence_numbers(file_path):
        """
        Extracts the text content of <sequence_number> elements from
an XML file.

        Args:
            file_path (str): The path to the XML file.

        Returns:
            list: A list of text contents from <sequence_number>
elements.
        """
        # Parse the XML file
        tree = ET.parse(file_path)
        root = tree.getroot()

        # Find all sequence_number elements
        sequence_numbers =
root.findall('.//{http://pds.nasa.gov/pds4/pds/v1}elements')

        # Extract the text content from each <sequence_number> element
        return [int(sequence_number.text) for sequence_number in
```

```
sequence_numbers]
    @staticmethod
    def _find_xml_files(base_path):
        """
        Finds all XML files matching the pattern
/data/calibrated/*/*.xml within the given base path.

        Args:
            base_path (str): The base directory path where the search
begins.

        Returns:
            list: A list of paths to the matching XML files.
        """
        # Define the search pattern
        pattern = os.path.join(base_path, 'data', 'calibrated', '*',
'*.xml')

        # Use glob to find all files matching the pattern
        matching_files = glob.glob(pattern)
        matching_files.sort()
        return matching_files

    @staticmethod
    def _find_qub_files(base_path):
        """
        Finds all .qub files matching the pattern
/data/calibrated/*/*.qub within the given base path.

        Args:
            base_path (str): The base directory path where the search
begins.

        Returns:
            list: A list of paths to the matching .qub files.
        """
        # Define the search pattern
        pattern = os.path.join(base_path, 'data', 'calibrated', '*',
'*.qub')

        # Use glob to find all files matching the pattern
        matching_files = glob.glob(pattern)
        matching_files.sort()
        return matching_files
    @staticmethod
    def _get_image_array(qub_path,shape):
        """
        Args:
            qub_path : path to the .qub file
            shape : shape of the image of form (channels,height,width)
```

```
        returns:
            a numpy array reshaped in the shape provided
        """
        with open(qub_path, 'rb') as f:
            img = np.frombuffer(f.read(),
dtype=np.float32).reshape(*shape)
        return img

    @staticmethod
    def get_image(base_path):
        """
        Args:
            base_path : Base directory path to the data. Which means
the path to directory which contains the browse, data geometyr and
miscellaneous folder
        returns:
            A list of images with the radiance resized and resahped.
To visualize it you need to do min max scaling and stuff
        """
        xml_files = utils._find_xml_files(base_path)
        image_files = utils._find_qub_files(base_path)
        shapes = [utils._extract_sequence_numbers(xml_file) for
xml_file in xml_files]
        images = [utils._get_image_array(qub_path,shape) for
qub_path,shape in zip(image_files,shapes)]
        return images
    @staticmethod
    def _read_misc_files_into_df(file_path):
        with open(file_path) as f:
            data = f.read()
        rows = data.split('\n')
        parsed_list = [' '.join(row.split()).split() for row in rows]
        return pd.DataFrame(parsed_list)

    @staticmethod
    def _get_misc_files(base_path):
        """
        Args:
            base_path : Base directory path to the data. Which means
the path to directory which contains the browse, data geometyr and
miscellaneous folder
        returns:
            A dictionary of miscellaneous files converted to the
dataframe
        """
        pattern = os.path.join(os.path.join(base_path,
'miscellaneous', 'calibrated', '*', '*.*'))
        matching_files = glob.glob(pattern)
        matching_files.sort()
```

```
        dfs = {}
        for file_path in matching_files:
            print(file_path)
            _, file_extension = os.path.splitext(file_path)
            dfs[file_extension] =
utils._read_misc_files_into_df(file_path).dropna()
        return dfs
    @staticmethod
    def get_misc_files(base_path):
        """
        Args:
            base_path : Base directory path to the data. Which means
the path to directory which contains the browse, data geometyr and
miscellaneous folder
        returns:
            A dictionary of miscellaneous files converted to the
dataframe
        """

        dfs = utils._get_misc_files(base_path)
        dfs['.spm'] = utils._process_spm_df(dfs['.spm'])
        dfs['.oat'] = utils._process_oat_df(dfs['.oat'])
        return dfs
    @staticmethod
    def convert_to_reflectance(data, solar_zenith_angle):
        return (np.pi * data)/(np.cos(solar_zenith_angle * np.pi /
180) * F0 * D_AU**2)

    @staticmethod
    def _process_spm_df(spm_df):
        column_names = [
            "Record type", "Physical record number", "Block length",
        #     "Year",
            "Month",
            "Date",
            "Hour",
            "Minute",
            "Second",
            "Millisec", "Satellite position X", "Satellite position
Y",
            "Satellite position Z", "Satellite velocity X-dot",
            "Satellite velocity Y-dot", "Satellite velocity Z-dot",
            "Phase angle", "Sun aspect", "Sun Azimuth",
            "Sun Elevation",
        ]

        # Assign the new column names
        spm_df.columns = column_names
```

```
        # Convert the columns to the appropriate data types
        spm_df["Record type"] = spm_df["Record type"].astype(str)
        spm_df["Physical record number"] = spm_df["Physical record
number"].astype(np.int32)
        spm_df["Block length"] = spm_df["Block
length"].astype(np.int32)
        # spm_df['Year'] = spm_df['Year'].astype(np.int32)
        spm_df['Month'] = spm_df['Month'].astype(np.int32)
        spm_df['Date'] = spm_df['Date'].astype(np.int32)
        spm_df['Hour'] = spm_df['Hour'].astype(np.int32)
        spm_df['Minute'] = spm_df['Minute'].astype(np.int32)
        spm_df['Second'] = spm_df['Second'].astype(np.int32)
        spm_df['Millisec'] = spm_df['Millisec'].astype(np.int32)
        # For "Time in UTC", assuming it is in the format
'YYYYMMDDHHMMSS', convert to datetime

        spm_df["Satellite position X"] = spm_df["Satellite position
X"].astype(np.float32)
        spm_df["Satellite position Y"] = spm_df["Satellite position
Y"].astype(np.float32)
        spm_df["Satellite position Z"] = spm_df["Satellite position
Z"].astype(np.float32)
        spm_df["Satellite velocity X-dot"] = spm_df["Satellite
velocity X-dot"].astype(np.float32)
        spm_df["Satellite velocity Y-dot"] = spm_df["Satellite
velocity Y-dot"].astype(np.float32)
        spm_df["Satellite velocity Z-dot"] = spm_df["Satellite
velocity Z-dot"].astype(np.float32)
        spm_df["Phase angle"] = spm_df["Phase
angle"].astype(np.float32)
        spm_df["Sun aspect"] = spm_df["Sun aspect"].astype(np.float32)
        spm_df["Sun Azimuth"] = spm_df["Sun
Azimuth"].astype(np.float32)
        spm_df["Sun Elevation"] = spm_df["Sun
Elevation"].astype(np.float32)
        # spm_df["Orbit Limb Direction"] = spm_df["Orbit Limb
Direction"].astype(int)
        return spm_df
    @staticmethod
    def _process_oat_df(oat_df):
        columns_names = [
            "Record type",
            "Physical record number in this file",
            "Block length in bytes",
            "Month",
            "Date",
            "Hour",
            "Minute",
```

```
            "Second",
            "Millisec",
            "Lunar Position X (kms) - J2000 Earth Centre Frame",
            "Lunar Position Y (kms) - J2000 Earth Centre Frame",
            "Lunar Position Z (kms) - J2000 Earth Centre Frame",
            "Satellite position X (kms) - Note-3",
            "Satellite position Y (kms) - Note-3",
            "Satellite position Z (kms) - Note-3",
            "Satellite velocity X-dot (kms/sec) - Note-3",
            "Satellite velocity Y-dot (kms/sec) - Note-3",
            "Satellite velocity Z-dot (kms/sec)  - Note-3",
            "Altitude Inertial Q1",
            "Altitude Inertial Q2",
            "Altitude Inertial Q3",
            "Altitude Inertial Q4",
            "Earth Fixed IAU frame Q1",
            "Earth Fixed IAU frame Q2",
            "Earth Fixed IAU frame Q3",
            "Earth Fixed IAU frame Q4"
            "Lunar Fixed IAU frame Q1",
            "Lunar Fixed IAU frame Q2",
            "Lunar Fixed IAU frame Q3",
            "Lunar Fixed IAU frame Q4",
            "Latitude of sub-satellite point (deg)",
            "Longitude of sub-satellite point (deg)",
            "Solar Azimuth",
            "Solar Elevation",
            "Latitude  (deg)",
            "Longitude (deg)",
            "Satellite altitude (kms)",
            "Angle between +Roll and Velocity Vector",
            "Eclipse Status - Note-4",
            "Emission Angle",
            "Sun Angle w.r.t -ve Yaw (Phase angle)",
            "Angle between +Yaw and Nadir",
            "Slant Range (Km)",
            "Orbit No",
            "Solar Zenith Angle",
            "Angle between Payload FoV axis and velocity vector",
            "X  (yaw) angle",
            "Y  (roll) angle",
            "Z(pitch) angle",
        ]
        oat_df.columns = columns_names
        oat_df.iloc[:,1:9] = oat_df.iloc[:,1:9].astype(np.int32)
        oat_df.iloc[:,9:42] = oat_df.iloc[:,9:42].astype(np.float32)
        oat_df.iloc[:,42] = oat_df.iloc[:,42].astype(np.int32)
        oat_df.iloc[:,43:] = oat_df.iloc[:,43:].astype(np.float32)
        return oat_df
```

```
# data path should be root directory of the bundle
data_path =
"/kaggle/input/isro-chandrayan-iirs/other/dataset-10/1/data/ch2_iir_nc
i_20200122T0404077687_d_img_d18"
images = utils.get_image(data_path)
image = images[0]
misc_dfs = utils.get_misc_files(data_path)
oat_df = misc_dfs['.oat']
mean_zenith_angle = oat_df.loc[:,'Solar Zenith Angle'].mean()
reflectance_image =
utils.convert_to_reflectance(image,mean_zenith_angle)
# reflectance image is the desired output
```

```
/kaggle/input/isro-chandrayan-iirs/other/dataset-10/1/data/
ch2_iir_nci_20200122T0404077687_d_img_d18/miscellaneous/calibrated/
20200122/ch2_iir_nci_20200122T0404077687_d_img_d18.lbr
/kaggle/input/isro-chandrayan-iirs/other/dataset-10/1/data/ch2_iir_nci
_20200122T0404077687_d_img_d18/miscellaneous/calibrated/20200122/
ch2_iir_nci_20200122T0404077687_d_img_d18.oat
/kaggle/input/isro-chandrayan-iirs/other/dataset-10/1/data/ch2_iir_nci
_20200122T0404077687_d_img_d18/miscellaneous/calibrated/20200122/
ch2_iir_nci_20200122T0404077687_d_img_d18.oath
/kaggle/input/isro-chandrayan-iirs/other/dataset-10/1/data/ch2_iir_nci
_20200122T0404077687_d_img_d18/miscellaneous/calibrated/20200122/
ch2_iir_nci_20200122T0404077687_d_img_d18.spm
```

```
reflectance_image[7:115, 0, 0]
```

```
array([0.07417825, 0.07434882, 0.07655642, 0.07588779, 0.07756479,
       0.07615556, 0.07812397, 0.07668536, 0.07923326, 0.08007594,
       0.07971056, 0.07915016, 0.08484272, 0.08539544, 0.08138908,
       0.08645678, 0.08365219, 0.08684789, 0.0886406 , 0.088315  ,
       0.08954253, 0.09601055, 0.11621486, 0.10631726, 0.1021927 ,
       0.09952087, 0.09837842, 0.09777242, 0.10442795, 0.10254341,
       0.09793654, 0.09864443, 0.09835146, 0.09896696, 0.10101151,
       0.10275173, 0.10479425, 0.10471216, 0.10665133, 0.10909155,
       0.11074782, 0.11163307, 0.11450666, 0.11262033, 0.11116292,
       0.11454606, 0.11422974, 0.11820742, 0.12185821, 0.1195468 ,
       0.12350811, 0.12070794, 0.12250887, 0.12223745, 0.12259433,
       0.12341735, 0.12578655, 0.12094436, 0.12243683, 0.13220589,
       0.13339657, 0.13646125, 0.13857756, 0.14308769, 0.1476499 ,
       0.14518736, 0.14318348, 0.13879346, 0.14311837, 0.13959993,
       0.14292387, 0.13932596, 0.13420626, 0.13305296, 0.13411494,
       0.13846667, 0.1429214 , 0.14431032, 0.14723379, 0.1465318 ,
       0.14609543, 0.14515233, 0.15055548, 0.14678895, 0.14876432,
```

```
       0.15241998, 0.15293012, 0.15740045, 0.16320721, 0.15779928,
       0.15296565, 0.15251741, 0.15778854, 0.15787022, 0.15970547,
       0.1593177 , 0.16153666, 0.16163294, 0.16603867, 0.16525269,
       0.16671426, 0.16684679, 0.1703375 , 0.17015271, 0.16995738,
       0.17130512, 0.17314632, 0.17530703])
```

```
reflectance_image[7:115, 0, 0].shape
```

```
(108,)
```

```
reflectance_image.shape
```

```
(256, 11012, 250)
```

```
# interpolated_df =
pd.read_csv('/kaggle/input/interpolated-data/bir1lm044_Silicate_(Ino)_
_Pyroxene__from_lunar_basalt.csv')
# interpolated_reflectance = interpolated_df['Reflectance'].values

import glob
import os

folder_path = '/kaggle/input/interpolated-data'

csv_files = glob.glob(os.path.join(folder_path, '*.csv'))

reflectance_vectors = {}

for file in csv_files:
    filename = os.path.splitext(os.path.basename(file))[0]

    df = pd.read_csv(file)

    reflectance_vectors[filename] = df['Reflectance'].values[1:]

reflectance_vectors['bir1lm044_Silicate_(Ino)__Pyroxene__from_lunar_ba
salt'].shape
```

```
(108,)
```

```
# interpolated_reflectance = interpolated_reflectance[1:]
# interpolated_reflectance.shape


def spectral_angle_mapper(pixel_vector, reference_vector):

    dot_product = np.dot(pixel_vector, reference_vector)

    norm_pixel = np.linalg.norm(pixel_vector)

    norm_reference = np.linalg.norm(reference_vector)
```

```
    cosine_angle = dot_product / (norm_pixel * norm_reference)

    angle = np.arccos(np.clip(cosine_angle, -1.0, 1.0))

    return angle

sam = []
for v in reflectance_vectors.values():
    sam.append(spectral_angle_mapper(reflectance_image[7:115, 0, 0],
v))

sam
```

```
[0.3107609242955783,
 0.22851939909589297,
 0.30629042396817785,
 0.29406854794868853,
 0.28544784889557456,
 0.456645027589915,
 0.3062103020444631,
 0.269084046622231,
 0.34843381858513317,
 0.20141974960151807,
 0.2499793161722711,
 0.3196507510015466,
 0.3107274312025724,
 0.12407684080424404,
 0.26231717357033746,
 0.24870050427341964,
 0.2647682020011628,
 0.12405848793740383,
 0.3244801237523986,
 0.26598495819039647,
 0.22884091820320074,
 0.18311291703538973,
 0.34388744262038784,
 0.27351992707536243,
 0.15337712590198668,
 0.2835599380265775,
 0.29647883995542496,
 0.21022801683554135,
 0.26487837577967793,
 0.30742119471690793,
 0.2964651837578382,
 0.23078094433685445,
 0.2921623790083003,
 0.32149752055707903,
 0.1910298604038281,
 0.1107813253884727,
 0.2995833272680778,
```

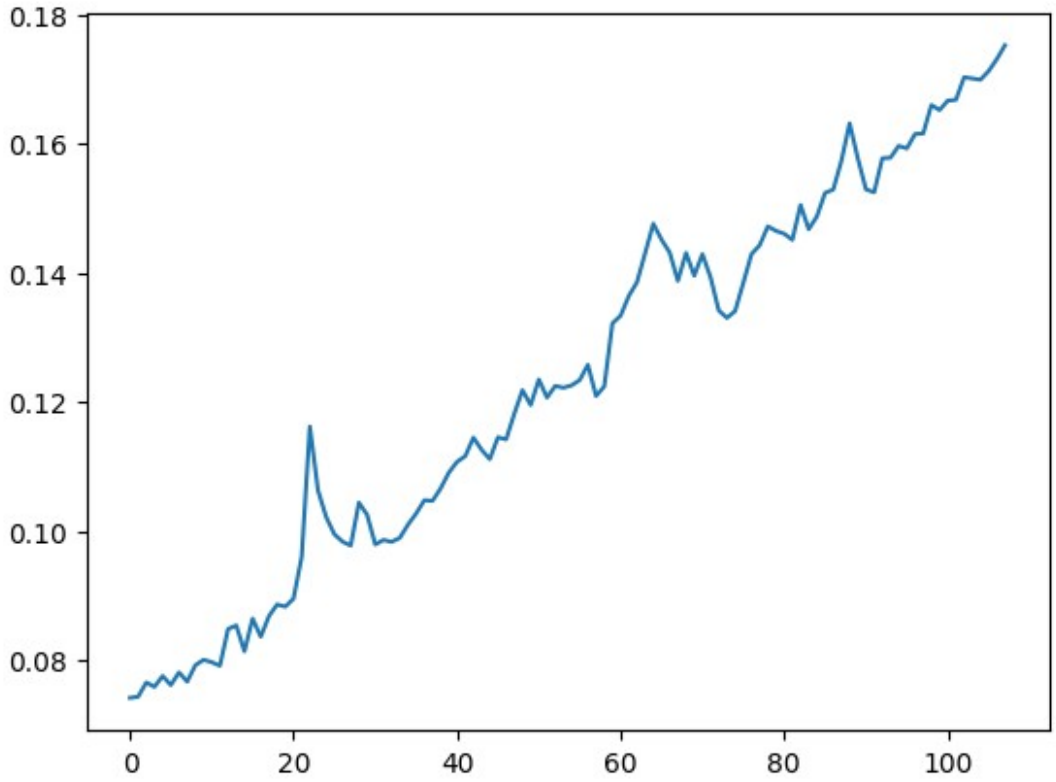
```
0.42378578651794063,
0.22220176574305767,
0.2700031835854677,
0.358811012051612,
0.28697664108493004,
0.16757475037780034,
0.21661642621355873,
0.26438213284746054,
0.13468175229574136,
0.2569136672631539,
0.23737639173082,
0.2992540750725525,
0.29249843787660046,
0.26435665006774395,
0.13474201661043014,
0.29738424369697997,
0.1909439974275937,
0.28711541798674356,
0.2936393841613467,
0.289379523311945,
0.1344816852704631,
0.2869443621244728,
0.29211508904095007,
0.1188212020052375,
0.43093121479602364,
0.3297032991481396,
0.30533350251508923,
0.2914199245356497,
0.182607756420661,
0.06270074272650773,
0.2927572333932568,
0.24354810701278462,
0.33346920545595893,
0.35110413569507004,
0.1588294077658691,
0.05513094382995546,
0.28217140180055583,
0.2917707991789591,
0.31346824933360995,
0.1353765702026935,
0.20498525412466312,
0.2643566500677444,
0.29532321015752705,
0.3070059622425305,
0.24630161156177388,
0.05837308108755834,
0.24479854918788102,
0.28705421295904704,
0.23674036384413885]
```

```
min_sam_key = min(reflectance_vectors.keys(), key=lambda k:
sam[list(reflectance_vectors.keys()).index(k)])
min_sam_key
```

```
'calmc2_Antarctic_Meteorite_of_Presumed_Lunar_Origin'
```

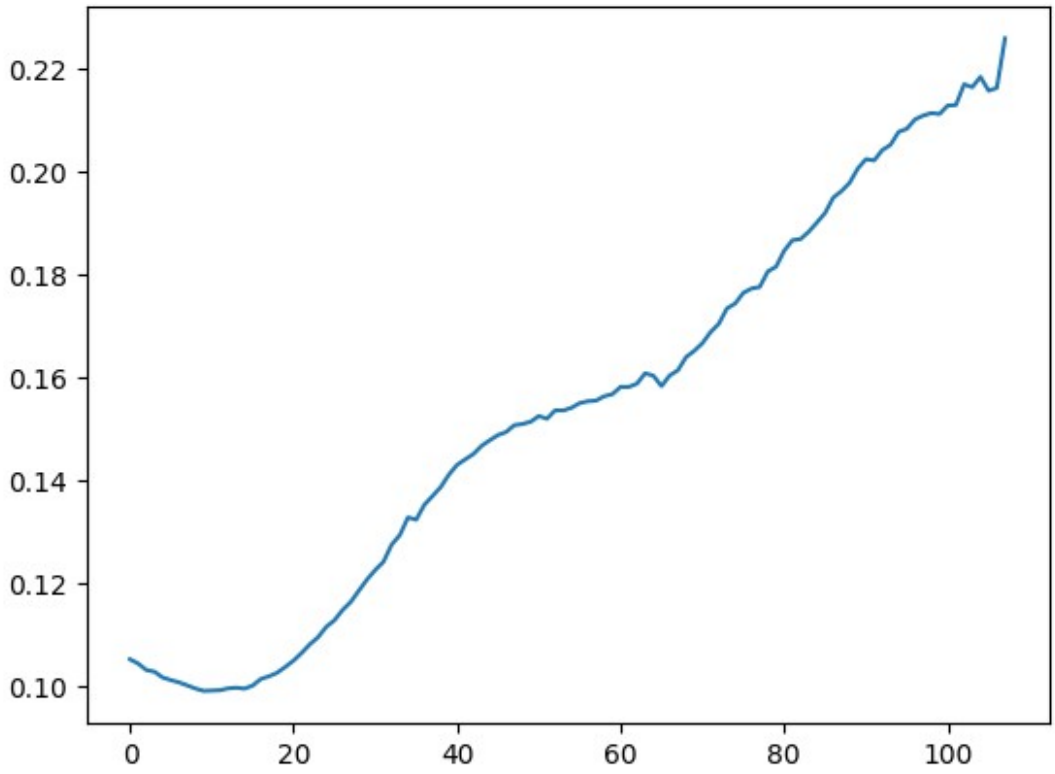
```
import matplotlib.pyplot as plt

plt.plot(reflectance_image[7:115, 0, 0])
```

```
[<matplotlib.lines.Line2D at 0x7af92c1be080>]
```




```
plt.plot(reflectance_vectors[min_sam_key])
```

```
[<matplotlib.lines.Line2D at 0x7af92c5d32b0>]
```

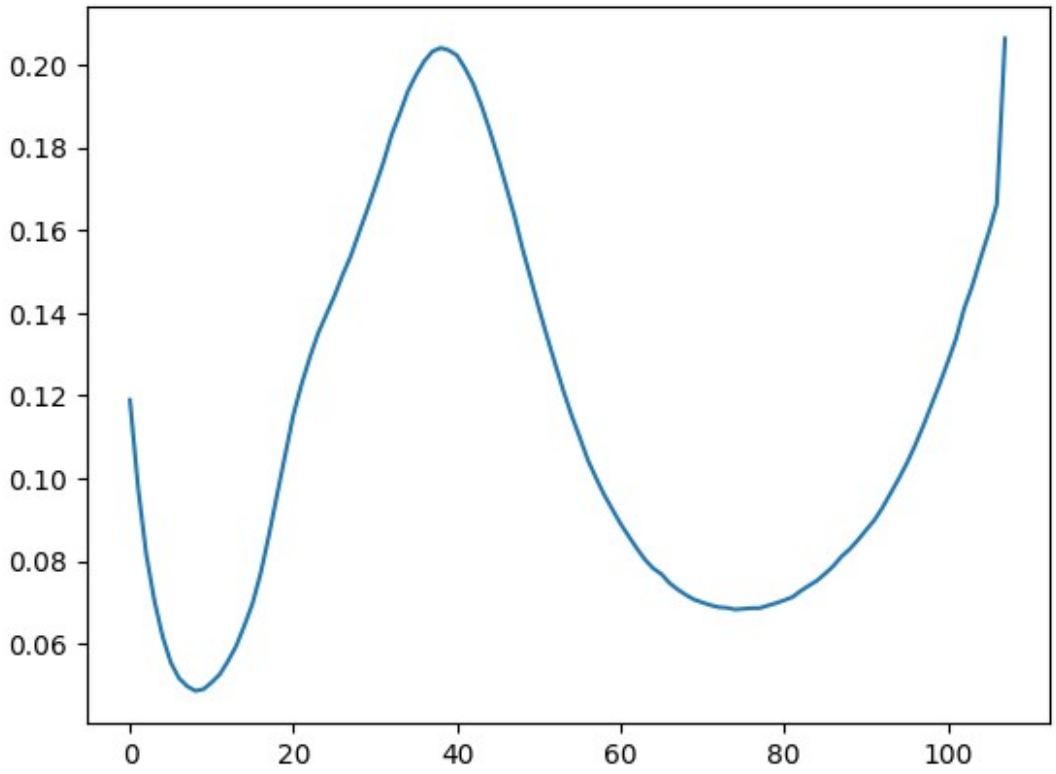



```
max_sam_key = max(reflectance_vectors.keys(), key=lambda k:
sam[list(reflectance_vectors.keys()).index(k)])
max_sam_key
```

```
'c3mb29_Silicate_(Ino)__Pyroxene__A-
881757_Lunar_Gabbrocontaining_Plag__Pyroxene_and_Ilmenite'
```

```
plt.plot(reflectance_vectors[max_sam_key])
```

```
[<matplotlib.lines.Line2D at 0x7af92ca94dc0>]
```




```
SAM = np.zeros((500, reflectance_image.shape[2],
len(reflectance_vectors)))

for i in range(100, 600):
    for j in range(0, reflectance_image.shape[2]):
        for k, v in enumerate(reflectance_vectors.values()):
            SAM[i, j, k] =
spectral_angle_mapper(reflectance_image[7:115, i, j], v)

    if(i % 1000 == 0):
        print(i)

SAM

min_sam = np.min(SAM, axis=2)
min_sam

min_value = np.min(SAM)

min_coords = np.unravel_index(np.argmin(SAM), SAM.shape)

min_value

min_coords

plt.plot(reflectance_image[7:115, 87, 3])
```

```
(list(reflectance_vectors.keys()).index()

plt.plot(reflectance_vectors[list(reflectance_vectors.keys())[35]])

import torch

# Convert the reflectance image and reflectance vectors to PyTorch
tensors
reflectance_image_tensor =
torch.tensor(reflectance_image[7:115, :, :],
dtype=torch.float32).cuda()
reflectance_vectors_tensor = torch.tensor([v for v in
reflectance_vectors.values()], dtype=torch.float32).cuda()

# Initialize a 3D tensor for SAM with the same spatial dimensions as
the reflectance_image
SAM = torch.zeros((reflectance_image.shape[1],
reflectance_image.shape[2], len(reflectance_vectors))).cuda()

# Define the spectral_angle_mapper function for GPU usage
def spectral_angle_mapper_gpu(pixel_vector, ref_vector):
    dot_product = torch.dot(pixel_vector, ref_vector)
    norm_pixel = torch.norm(pixel_vector)
    norm_ref = torch.norm(ref_vector)
    angle = torch.acos(dot_product / (norm_pixel * norm_ref))
    return angle

# Calculate the SAM value for each pixel and store it in the SAM
tensor
for i in range(reflectance_image.shape[1]):
    for j in range(reflectance_image.shape[2]):
        pixel_vector = reflectance_image_tensor[:, i, j]  # Get the
spectral vector for pixel (i, j)
        for k, ref_vector in enumerate(reflectance_vectors_tensor):
            SAM[i, j, k] = spectral_angle_mapper_gpu(pixel_vector,
ref_vector)

    if i % 1000 == 0:
        print(f'Processing row {i}/{reflectance_image.shape[1]}')

# If needed, move the SAM tensor back to CPU and convert it to a NumPy
array
SAM_cpu = SAM.cpu().numpy()

SAM.shape

SAM

min_sam_key = min(reflectance_vectors.keys(), key=lambda k:
SAM[list(reflectance_vectors.keys()).index(k)])
min_sam_key
```

```
'calmc2_Antarctic_Meteorite_of_Presumed_Lunar_Origin'

# SAM = spectral_angle_mapper(reflectance_image[7:115, 0, 0],
interpolated_reflectance)

# SAM

0.26598495819039564
```

CPRSM

```
import numpy as np

def CPRMS(pixel_vector, reference_vector):

    mean_pixel = np.mean(pixel_vector)
    mean_reference = np.mean(reference_vector)

    centered_pixel = pixel_vector - mean_pixel
    centered_reference = reference_vector - mean_reference

    squared_diff = (centered_pixel - centered_reference) ** 2

    val = np.sqrt(np.mean(squared_diff))

    return val

cprms = []
for v in reflectance_vectors.values():
    cprms.append(spectral_angle_mapper(reflectance_image[7:115, 0, 0],
v))

cprms

[0.3107609242955783,
 0.22851939909589297,
 0.30629042396817785,
 0.29406854794868853,
 0.28544784889557456,
 0.456645027589915,
 0.3062103020444631,
 0.269084046622231,
 0.34843381858513317,
 0.20141974960151807,
 0.2499793161722711,
 0.3196507510015466,
 0.3107274312025724,
 0.12407684080424404,
 0.26231717357033746,
```

0.24870050427341964,
0.2647682020011628,
0.12405848793740383,
0.3244801237523986,
0.26598495819039647,
0.22884091820320074,
0.18311291703538973,
0.34388744262038784,
0.27351992707536243,
0.15337712590198668,
0.2835599380265775,
0.29647883995542496,
0.21022801683554135,
0.26487837577967793,
0.30742119471690793,
0.2964651837578382,
0.23078094433685445,
0.2921623790083003,
0.32149752055707903,
0.1910298604038281,
0.1107813253884727,
0.2995833272680778,
0.42378578651794063,
0.22220176574305767,
0.2700031835854677,
0.358811012051612,
0.28697664108493004,
0.16757475037780034,
0.21661642621355873,
0.26438213284746054,
0.13468175229574136,
0.2569136672631539,
0.23737639173082,
0.2992540750725525,
0.29249843787660046,
0.26435665006774395,
0.13474201661043014,
0.29738424369697997,
0.1909439974275937,
0.28711541798674356,
0.2936393841613467,
0.289379523311945,
0.1344816852704631,
0.2869443621244728,
0.29211508904095007,
0.1188212020052375,
0.43093121479602364,
0.3297032991481396,
0.30533350251508923,

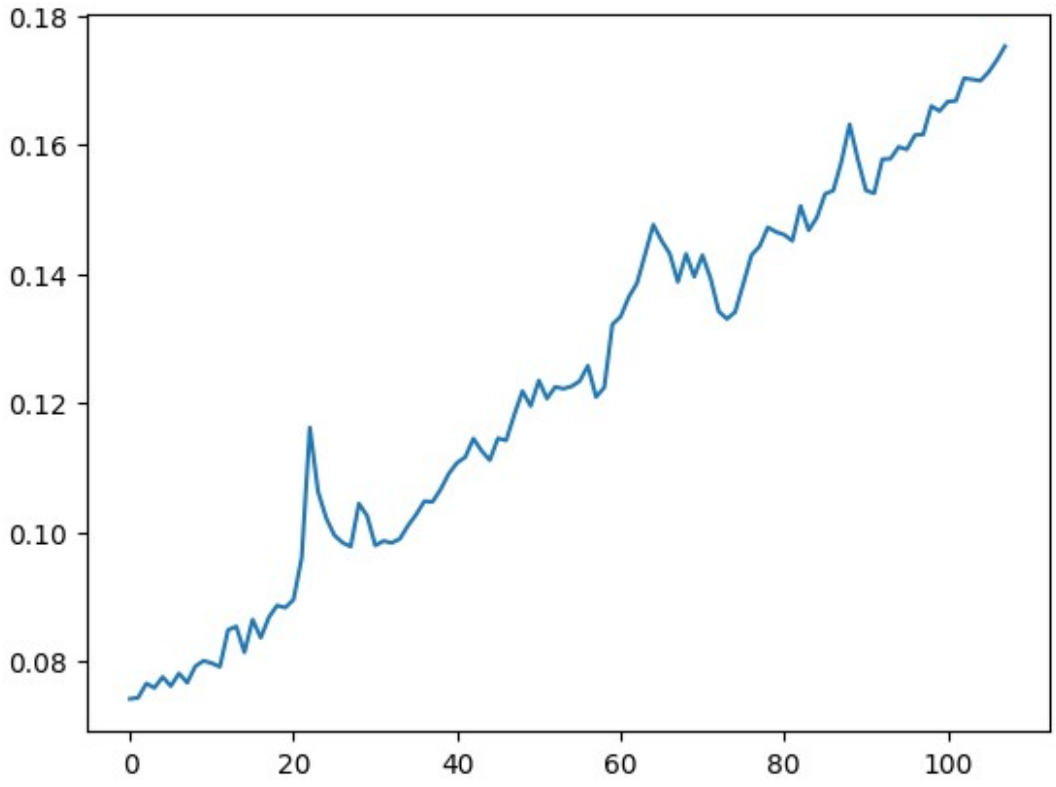
```
 0.2914199245356497,
 0.182607756420661,
 0.06270074272650773,
 0.2927572333932568,
 0.24354810701278462,
 0.33346920545595893,
 0.35110413569507004,
 0.1588294077658691,
 0.05513094382995546,
 0.28217140180055583,
 0.2917707991789591,
 0.31346824933360995,
 0.1353765702026935,
 0.20498525412466312,
 0.2643566500677444,
 0.29532321015752705,
 0.3070059622425305,
 0.24630161156177388,
 0.05837308108755834,
 0.24479854918788102,
 0.28705421295904704,
 0.23674036384413885]
```

```
min_cprms_key = min(reflectance_vectors.keys(), key=lambda k:
cprms[list(reflectance_vectors.keys()).index(k)])
min_cprms_key
```

```
'calmc2_Antarctic_Meteorite_of_Presumed_Lunar_Origin'
```

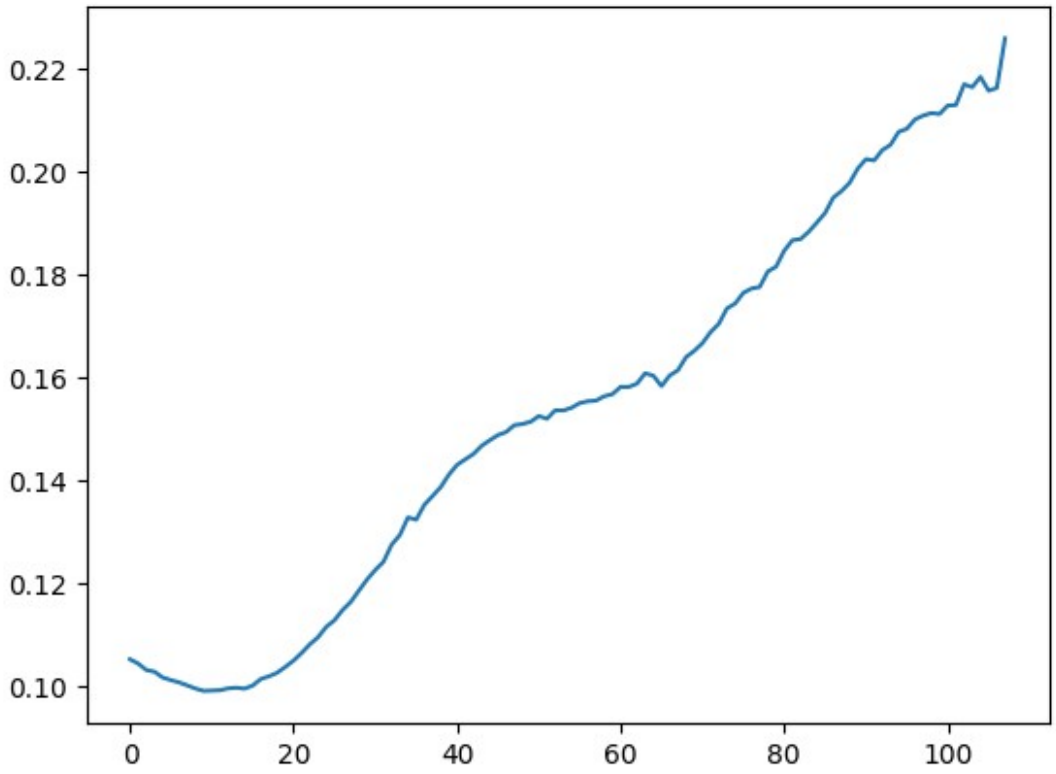
```
import matplotlib.pyplot as plt

plt.plot(reflectance_image[7:115, 0, 0])
```

```
[<matplotlib.lines.Line2D at 0x7af8b999f7c0>]
```




```
plt.plot(reflectance_vectors[min_cprms_key])
```

```
[<matplotlib.lines.Line2D at 0x7af92bf0c280>]
```




```
plt.plot(reflectance_vectors[max_cprms_key])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[47], line 1
----> 1 plt.plot(reflectance_vectors[max_cprms_key])

NameError: name 'max_cprms_key' is not defined
```

Torch Dataset

```
import os
import numpy as np
import torchdata.datapipes as dp
from torch.utils.data import DataLoader

class SlidingWindowDataPipe(dp.iter.IterDataPipe):
    def __init__(self, image_paths, window_size, step_size):
        super().__init__()
        self.image_paths = image_paths
        self.window_size = window_size
        self.step_size = step_size
```

```
    def __iter__(self):
        for image_path in self.image_paths:
            shapes =
SlidingWindowDataPipe.get_image_meta_data(image_path)
            for shape in shapes:
                _,img_height, img_width = shape
                # Sliding window
                for i in range(0, img_height - self.window_size + 1,
self.step_size):
                    if i + self.window_size > img_height:
                        continue
                    patch =
SlidingWindowDataPipe.get_partial_image_from_height(image_path,i,self.
window_size)
                    yield patch
    @staticmethod
    def _read_partial_data_of_given_height(qub_path,image_height,
image_width,row = 0, height = 250):
        image = []
        data_count = image_width * height
        channel_size = image_height * image_width
        with open(qub_path,'rb') as f:
            for channel_idx in range(256):
                offset = channel_idx * channel_size * 4 + row *
image_width  # float32 has 4 bytes
                f.seek(offset)
                channel_data = np.fromfile(f, dtype=np.float32,
count=data_count)
                image.append(channel_data.reshape((1,height,
image_width)))
        return np.vstack(image)
    @staticmethod
    def get_partial_image_from_height(base_path, row, height):
        xml_files = utils.find_xml_files(base_path)
        image_files = utils.find_qub_files(base_path)
        shapes = [utils.extract_sequence_numbers(xml_file) for
xml_file in xml_files]
        images =
[utils._read_partial_data_of_given_height(qub_path,shape[1], shape[2],
row, height) for qub_path,shape in zip(image_files,shapes)]
        return images
    @staticmethod
    def get_image_meta_data(base_path):
        xml_files = utils.find_xml_files(base_path)
        shapes = [utils.extract_sequence_numbers(xml_file) for
xml_file in xml_files]
        return shapes
```

```
data_paths = [
    "ENTER_LIST_OF_PATHS",
]

datapipe = SlidingWindowDataPipe(
    image_paths=data_paths,
    window_size=250,
    step_size=250
)
dataloader = DataLoader(datapipe, batch_size=12, shuffle=True)
dataloader_iter = iter(dataloader)
```

```
/opt/conda/lib/python3.10/site-packages/torch/utils/data/
graph_settings.py:103: UserWarning: `shuffle=True` was set, but the
datapipe does not contain a `Shuffler`. Adding one at the end. Be
aware that the default buffer size might not be sufficient for your
task.
  warnings.warn(
```

Example Usage of dataset

```
dataloader_iter.__next__()
```

```
---------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
last)
Cell In[9], line 1
----> 1 dataloader_iter.__next__()

File
/opt/conda/lib/python3.10/site-packages/torch/utils/data/dataloader.py
:630, in _BaseDataLoaderIter.__next__(self)
    627 if self._sampler_iter is None:
    628     # TODO(https://github.com/pytorch/pytorch/issues/76750)
    629     self._reset()  # type: ignore[call-arg]
--> 630 data = self._next_data()
    631 self._num_yielded += 1
    632 if self._dataset_kind == _DatasetKind.Iterable and \
    633         self._IterableDataset_len_called is not None and \
    634         self._num_yielded > self._IterableDataset_len_called:

File
/opt/conda/lib/python3.10/site-packages/torch/utils/data/dataloader.py
:674, in _SingleProcessDataLoaderIter._next_data(self)
    672 def _next_data(self):
    673     index = self._next_index()  # may raise StopIteration
--> 674     data = self._dataset_fetcher.fetch(index)  # may raise
StopIteration
```

```
    675     if self._pin_memory:
    676         data = _utils.pin_memory.pin_memory(data,
self._pin_memory_device)

File
/opt/conda/lib/python3.10/site-packages/torch/utils/data/_utils/fetch.
py:32, in _IterableDatasetFetcher.fetch(self, possibly_batched_index)
     30 for _ in possibly_batched_index:
     31     try:
---> 32         data.append(next(self.dataset_iter))
     33     except StopIteration:
     34         self.ended = True

File
/opt/conda/lib/python3.10/site-packages/torch/utils/data/datapipes/
_hook_iterator.py:154, in
hook_iterator.<locals>.IteratorDecorator.__next__(self)
    152         return self._get_next()
    153 else:  # Decided against using `contextlib.nullcontext` for
performance reasons
--> 154     return self._get_next()

File
/opt/conda/lib/python3.10/site-packages/torch/utils/data/datapipes/
_hook_iterator.py:142, in
hook_iterator.<locals>.IteratorDecorator._get_next(self)
    138 r"""
    139 Return next with logic related to iterator validity, profiler,
and incrementation of samples yielded.
    140 """
    141 _check_iterator_valid(self.datapipe, self.iterator_id)
--> 142 result = next(self.iterator)
    143 if not self.self_and_has_next_method:
    144     self.datapipe._number_of_samples_yielded += 1

File
/opt/conda/lib/python3.10/site-packages/torch/utils/data/datapipes/
_hook_iterator.py:226, in hook_iterator.<locals>.wrap_next(*args,
**kwargs)
    224         result = next_func(*args, **kwargs)
    225 else:
--> 226     result = next_func(*args, **kwargs)
    227 datapipe._number_of_samples_yielded += 1
    228 return result

File
/opt/conda/lib/python3.10/site-packages/torch/utils/data/datapipes/
datapipe.py:381, in _IterDataPipeSerializationWrapper.__next__(self)
    379 def __next__(self) -> T_co:  # type: ignore[type-var]
    380     assert self._datapipe_iter is not None
```

```
--> 381     return next(self._datapipe_iter)

File
/opt/conda/lib/python3.10/site-packages/torch/utils/data/datapipes/
_hook_iterator.py:183, in hook_iterator.<locals>.wrap_generator(*args,
**kwargs)
    181         response = gen.send(None)
    182 else:
--> 183     response = gen.send(None)
    185 while True:
    186     datapipe._number_of_samples_yielded += 1

Cell In[6], line 15, in SlidingWindowDataPipe.__iter__(self)
     13 def __iter__(self):
     14     for image_path in self.image_paths:
---> 15         shapes =
SlidingWindowDataPipe.get_image_meta_data(image_path)
     16         for shape in shapes:
     17             _,img_height, img_width = shape

Cell In[6], line 45, in
SlidingWindowDataPipe.get_image_meta_data(base_path)
     43 @staticmethod
     44 def get_image_meta_data(base_path):
---> 45     xml_files = utils.find_xml_files(base_path)
     46     shapes = [utils.extract_sequence_numbers(xml_file) for
xml_file in xml_files]
     47     return shapes

AttributeError: type object 'utils' has no attribute 'find_xml_files'
This exception is thrown by __iter__ of
SlidingWindowDataPipe(image_paths=['ENTER_LIST_OF_PATHS'],
step_size=250, window_size=250)
```