Help File for Sentinel1 Peak Alignment

This is the code for the time series alignment method which applied in our paper:

> S. Maleki et al. "*Sentinel-1 (S1) Time Series Alignment Method for Rapeseed Fields Mapping*", under review at Frontiers in Remote Sensing journal, 2024.

In fact, the position of the highest peak in the Sentinel1 time series across the rapeseed growth cycle differs between two years or between two study sites as shown in Figure1, which may likely induce lower accuracies when transferring a classifier from one site-year to another. The alignment of the highest peaks in the S1 time series of the training and test datasets was achieved through a process consisting of three steps outlined in our paper.
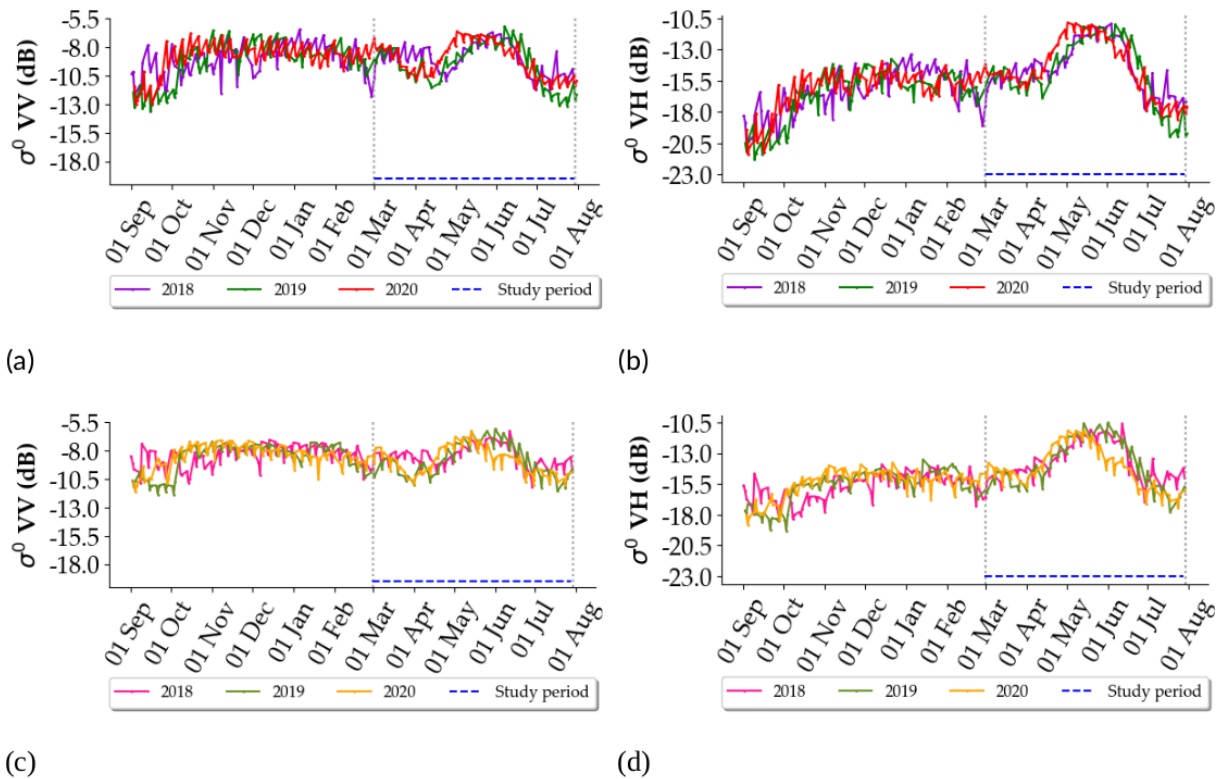

Figure 1. Time series of S1 average backscattering coefficients (σ°) in VV and VH polarizations for rapeseed fields in each year (2018, 2019 and 2020). The temporal period used in this study is represented by dashed lines. (a) La Rochelle (Fr): VV, (b) La Rochelle: VH, (c)Tarbes (Fr): VV, (d)Tarbes: VH.

==**Step 1: Use the script Step1_Peak_Detection.py to identify the dataset on which the alignment method should be applied.**==

Overview

This Python script processes Sentinel-1 SAR data, with a focus on the VV polarization, to smooth the data, detect peaks, and identify the highest peak within a specific time range. Additionally, it determines the mean number of timesteps before the highest peak. The script can be applied to VH polarization as well. Visualizations of the original and smoothed data, along with detected peaks, are also generated for verification.

Dependencies

Ensure the following Python libraries are installed:

- **Pandas**, **matplotlib**, **numpy**, **scipy**, **datetime**

Processes:

1. **Smoothing**:

   o   Applies a Gaussian filter to smooth SAR data.

   o   Iteratively adjusts the smoothing factor to ensure appropriate peak detection.

2. **Peak Detection**:

   o   Detects local maxima (peaks) and minima (bottoms).

   o   Identifies the highest peak within a user-defined date range.

3. **DOY Adjustment**:

   o   Adjusts DOYs to align the highest peak to a reference DOY (e.g., 100).

4. **Visualization**:

   o   Plots original and smoothed data with detected peaks for verification.

5. **Statistical Analysis**:

   o   Determines the mean number of timesteps before the highest peak.

Input Files

Train Dataset

A .npz file for the training study area and year, containing:

- X_SAR_train: A 3D numpy array of shape (n_samples, n_timepoints, n_polarizations) containing the time series of SAR polarizations, where each row represents a time series of for example VV polarization. In the **X_SAR** array, VV polarization values are stored at the first index, and VH polarization values are stored at the second index along the third dimension. For example:

```
vv_df_train = pd.DataFrame(X_SAR_train[:, :, 0])
```

- y_multi_train: Labels corresponding to each data row.

- id_parcels_train: IDs of the data parcels.

- dates_SAR_train: List of dates corresponding to the SAR data.

Example: Colza_data_C_W5_2019_matched.npz

Test Dataset

B .npz file for the testing study area and year, containing:

- X_SAR_test: 3D array of SAR data with the same structure as training dataset.

- y_multi_ test: Labels corresponding to each data row.

- id_parcels_test: IDs of the data parcels.

- dates_SAR_test: List of dates corresponding to the SAR data.

Example: Colza_data_U_new_5_2020_matched.npz

How to Use

1. Configure Input Data

- Replace dataset_train and dataset_test paths with your file paths for .npz datasets.

```python
# Load the numpy data
year_train=2019
dataset_train = np.load(
        f'D:/Colza_DB/One_year/W5/CA_US/CA2019_US2020/Colza_data_C_W5_{year_train}_matched.npz', allow_pickle=True)
X_SAR_train, y_multi_train = dataset_train["X_SAR"], dataset_train["y"]
id_parcels_train, dates_SAR_train = dataset_train["id_parcels"], dataset_train["dates_SAR"]

# Convert dates_SAR_train to DataFrame
dates_SAR_train = pd.DataFrame(dates_SAR_train, columns=['Date'])

# Create vv_df from X_SAR_train[:, :, 0] ### vv is the first feature.
#Since the number of vv and vh is equal we do the calculation before alignement using one of them

vv_df_train = pd.DataFrame(X_SAR_train[:, :, 0])
#vv_df = to_dB(vv_df)
vv_df_train['primary_id'] = id_parcels_train   # Add a new column "primary_id" to store the original IDs
vv_df_train['y_multi_train'] = y_multi_train   # Add a new column "y_multi_train" to store y_multi_train
colza_SAR_vv_train = vv_df_train[y_multi_train == "CZH"]
print('colza_SAR_vv_train.shape', colza_SAR_vv_train.shape)
# Call the function for 'vv'
peak_df_vv_train, vv_smoothgauss_df_vv_train = smooth_and_plot_train(colza_SAR_vv_train, 'vv',year_train,dates_SAR_train)
```

Replace the training year → year_train=2019
Replace the training dataset → dataset_train = np.load(...)

```python
# Load the numpy data
year_test=2020
dataset_test = np.load(
        f'D:/Colza/One_year/W5/CA_US/CA2019_US2020/Colza_data_U_new_5_{year_test}_matched.npz', allow_pickle=True)
X_SAR_test, y_multi_test = dataset_test["X_SAR"], dataset_test["y"]
id_parcels_test, dates_SAR_test = dataset_test["id_parcels"], dataset_test["dates_SAR"]

# Convert dates_SAR_test to DataFrame
dates_SAR_test = pd.DataFrame(dates_SAR_test, columns=['Date'])

vv_df_test = pd.DataFrame(X_SAR_test[:, :, 0])
#vv_df = to_dB(vv_df)
vv_df_test['primary_id'] = id_parcels_test   # Add a new column "primary_id" to store the original IDs
vv_df_test['y_multi_test'] = y_multi_test   # Add a new column "y_multi_test" to store y_multi_test
SAR_vv_test = vv_df_test
print('SAR_vv_test.shape', SAR_vv_test.shape)

# Call the function for 'vv'
peak_df_vv_test, vv_smoothgauss_df_vv_test = smooth_and_plot_test(SAR_vv_test, 'vv',year_test,dates_SAR_test)
```

Replace the testing year → year_test=2020
Replace the testing dataset → dataset_test = np.load(...)

2. Run the Script

- Execute the script. It processes training data first, followed by test data.

- Visualizations are generated for up to 10 rows from each dataset.

Notes

A. Ensure the date range for peak detection matches the phenological cycle of rapeseed in your study area. For example:

   - France: Peaks between April 1st and July 1st.

   - North America: Peaks between June 1st and November 1st.

```python
# Find the highest peak only between April 1st and July 1st for France
## Find the highest peak only between June 1st and November 1st for North America

peaks_within_range = []
for j in range(len(peak_pos)):
    if pd.Timestamp(year=year_train, month=6, day=1) <= dates_SAR_train.iloc[peak_pos[j]]['Date'] < pd.Timestamp(year=year_
        peaks_within_range.append(j)
```

B. For large datasets, consider limiting the number of plots to avoid excessive computation time.

```python
############################################# plot
if fig_count < 10:     # Limit the number of plots to the first 10 rows with no detected peaks
    plt.figure(fig_count)
    plt.plot(doys_adjusted, data, color='blue', label='Original Data')
    plt.plot(doys_adjusted, data_smoothgauss, label='Smoothed Data')
    plt.plot(doys_adjusted[highest_doy], highest_peak, 'o', color='r', label='Detected Peaks')
    plt.title(f"Visualization for Primary ID: {current_primary_id}")  # Set the title with primary_id
    plt.ylabel(f"{pol_label}dB")
    plt.legend()
    plt.grid()
    plt.show()
    fig_count += 1

if fig_count < 10:  # Limit the number of plots
    plt.figure(fig_count)
    plt.plot(dates_SAR_train, data, color='black', label='Original Data')
    plt.plot(dates_SAR_train, data_smoothgauss, label='Smoothed Data')
    plt.plot(highest_date, highest_peak, 'o', color='r', label='Detected Peaks')
    # Get the primary_id for the current row
    current_primary_id = df['primary_id'].iloc[i]
    plt.title(f"Visualization for Primary ID: {current_primary_id}")  # Set the title with primary_id
    plt.ylabel(f"{pol_label}dB")
    # Format x-axis for cleaner date labels
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))  # Year-Month format
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=1))  # Tick every month
    plt.legend()
    plt.grid()
    plt.show()
    fig_count += 1
```

---

3. Outputs

A. Peak Data

   - Description: A DataFrame containing information about detected peaks, including peak values, adjusted Days of Year (DOYs), and other relevant metrics for each parcel. (peak_df_vv_train and peak_df_vv_test).

   - Use Case: Helps in verifying the correctness of calculation.

B. Smoothed Data

   - Description: A DataFrame of Gaussian smoothed SAR backscatter values for each parcel (vv_smoothgauss_df_vv_train and vv_smoothgauss_df_vv_test).

   - Use Case: Helps in verifying the correctness of calculation.

C. Visualizations

- o Description:

    - Plots showing both the original and smoothed time series data.

    - Detected peaks are marked on the plots for up to 10 parcels.

- o Use Case: Helps in verifying the correctness of peak detection and smoothing visually.

D. The most important output for alignment: Mean Timestep Analysis

- o The script calculates the mean number of timesteps before the detected peak and compares it between the train and test datasets, then returns the largest mean number of timesteps as Set_number. In the alignment process, the dataset with the smaller mean number of timesteps will be adjusted so that its mean number of timesteps equals Set_number. It also defines the dataset (train or test) to which timestamps should be added at the beginning and removed from the end in the second step (Use the script Step2_Peak_Alignment.py to perform the alignment).

- o **Use Case:**

    - **The Set_number value (e.g., Set_number = 10) is used as an input in Step 2 (Step2_Peak_Alignment script) to perform temporal alignment of the time series. It also determines which dataset (train or test) should be used in the second step (Use the script Step2_Peak_Alignment.py for the alignment process).**

Step 2: Use the script Step2_Peak_Alignment.py to perform the alignment.

This script processes Sentinel-1 SAR data (VV and VH polarizations) and applies a peak detection and alignment method within train and test dataset. **The input file is the dataset (either train or test) defined in Step 1. To align the train and test datasets, the other dataset (train or test) remains unchanged.** The script smooths the time series data, detects the highest peak within a specified date range (June 1st to November 1st for North America or April 1st to July 1st for France), and aligns the time series accordingly. It also handles data visualization and stores the aligned data for further use.

Dependencies

Ensure the following Python libraries are installed:

- **Pandas, matplotlib, numpy, scipy, datetime**

Input Files

The input file is the dataset (either train or test) defined in Step 1. To align the train and test datasets, the other dataset (train or test) remains unchanged. npz file containing:

- X_SAR_train: 3D array of SAR data containing the time series of SAR polarizations, where each row represents a time series of for example VV polarization. In the **X_SAR** array, VV polarization values are stored at the first index, and VH polarization values are stored at the second index along the third dimension. For example:

```python
vv_df = pd.DataFrame(X_SAR[:, :, 0])

vh_df = pd.DataFrame(X_SAR[:, :, 1])
```

- y_multi: Labels corresponding to each data row.

- id_parcels: IDs of the data parcels.

- dates_SAR: List of dates corresponding to the SAR data.

Example: Colza_data_C_W5_2019_matched.npz

Processes:

1. **Smoothing**: The data is smoothed using a Gaussian filter.

2. **Peak Detection**: It identifies peaks and bottom points in the smoothed data using a simple peak detection algorithm.

3. **Peak Alignment**: For each time series, the peak is aligned to the year-based day of year (DOY).

4. **Visualization**: The original and smoothed data, along with detected peaks, are plotted for visual inspection.

5. **Data Adjustment**: The time series is adjusted based on the peak location and aligned. The Set_number parameter specifies the number of timestamps that should precede the highest peak in the dataset, ensuring alignment between the train and test datasets. If the number of timestamps before the peak is less than Set_number, additional timestamps need to be added at the beginning of the data to make the number of timestamps before the peak equal to Set_number. Finally, the same number of added timestamps will be removed from the end of the time series. More details are provided in our paper, currently under review in *Frontiers in Remote Sensing* journal.

How to Use

1. **Set Parameters:** Adjust the Set_number variable achieved from setep1.

```
######## Created by Saeideh Maleki
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import ndimage

#######Part2. allignement using the Set_number value from the script Step2_Peak_Alignement and replace by Set_number value in curr
Set_number=10
def smooth_and_plot(df, pol_label, year,dates_SAR):
    peak_data = []
    doys_adjusted_list = []
    data_list = []
    valid_id_parcels = []
    valid_y_multi = []
    columns_added_list = []
    fig_count = 0
```

2. Configure Input Data

- Replace input dataset with your file paths for .npz datasets.

Replace the year

Replace the dataset

```
# Load the numpy data
year = 2020
name= 'Colza_data_U_new_5_'
dataset = np.load(
        f'D:/Colza_DB/One_year/W5/CA_US/CA2019_US2020/Colza_data_U_new_5_{year}_matched.npz', allow_pickle=True)
X_SAR, y_multi = dataset["X_SAR"], dataset["y"]
id_parcels, dates_SAR = dataset["id_parcels"], dataset["dates_SAR"]
print('input X_SAR before alignement:', X_SAR.shape)
```

3. Run the Script

- Execute the script.

- Visualizations are generated for up to 5 rows from the dataset.

Notes

B. Ensure the date range for peak detection matches the phenological cycle of rapeseed in your study area. For example:

  o France: Peaks between April 1st and July 1st.

  o North America: Peaks between June 1st and November 1st.

```
# Find the highest peak only between April 1st and July 1st for France
## Find the highest peak only between June 1st and November 1st for North America

peaks_within_range = []
for j in range(len(peak_pos)):
    if pd.Timestamp(year=year_train, month=6, day=1) <= dates_SAR_train.iloc[peak_pos[j]]['Date'] < pd.Timestamp(year=year
        peaks_within_range.append(j)
```

C. For large datasets, consider limiting the number of plots to avoid excessive computation time.

```
if fig_count < 5:  # Limit the number of plots to the first 5 rows with no detected peaks
    plt.figure(fig_count)
    plt.plot(doys_adjusted, data, color='red', label='Original Data')
    plt.plot(doys_adjusted, data_smoothgauss, color='black', label='Smoothed Data')
    plt.plot(doys_adjusted[highest_doy], highest_peak, 'o', color='r', label='Detected Peaks')

    # Get the primary_id for the current row
    plt.title(f"Visualization for Primary ID: {current_primary_id}")  # Set the title with primary_id
    plt.ylabel(f"{pol_label}dB")
    plt.xlabel(f"Standardized timestamps")
    plt.legend()

    plt.show()
    fig_count += 1

if fig_count < 5:  # Limit the number of plots to the first 5 rows with no detected peaks
    plt.figure(fig_count)
    plt.plot(dates_SAR, data, color='red', label='Original Data')
    plt.plot(dates_SAR, data_smoothgauss, color='black', label='Smoothed Data')
    plt.plot(highest_date, highest_peak, 'o', color='r', label='Detected Peaks')
    # Get the primary_id for the current row
    plt.title(f"Visualization for Primary ID: {current_primary_id}")  # Set the title with primary_id
    plt.ylabel(f"{pol_label}dB")
    plt.xlabel(f"Dates")
    plt.legend()
    plt.show()
    fig_count += 1
```

Outputs:

- columns_added_vv_w5_oneyear.csv: CSV file listing the number of columns added to each VV time series.

  - Use Case: Helps in verifying the correctness of calculation.

- columns_added_vh_w5_oneyear.csv: CSV file listing the number of columns added to each VH time series.

  - Use Case: Helps in verifying the correctness of calculation.

- **The most important output is <name><year>_aligned.npz: The processed SAR data, saved as a .npz file containing the aligned X_SAR, y, id_parcels, and dates_SAR. This dataset will be used in classification.**

Notes

To verify the alignment, and compare before and after alignment you can use Plot_alignment script.

For the code associated to the classification task, please refer to the following repository: [Colza_Classif](https://github.com/cassiofragadantas/Colza_Classif)