# CAL/VAL Prototype

## README File

**Change log**

May 29, 2025, V02.00.01

* Added all output variables into the table of “Output Variables”

April 08, 2025, V02.00.00

* Added a new python class to calculate several statistical metrics for FLEX and FLOX images.
* Installation & codes instructions improved.

January 07, 2025, V01.01.01

* Improved support for FLEX images.
* Installation & codes instructions improved.

November 29, 2024, V01.01.00

* Add support for FLEX images.

September 2, 2024, V01.00.01

* Add more detailed instructions on how to prepare the files for input
* Add an optional standalone python file (.py) to facilitate the subfolder creation.

August 28, 2024, V01.00.00

* First version of the CAL/VAL Prototype Guideline

**Objective and Scope**

The objective of this CAL/VAL prototype is to calculate coefficients of variation (CV) of regions of interest (ROI) centred at field sites, using Sentinel-2 (S2) L1C and L2A images and FLEX images.

First, this prototype processes all FLEX images and calculates the average and the standard deviation of the value “Sif Emission Spectrum“ inside the ROI. Besides, it will extract the date and the time of the FLEX images whose vegetation pixels inside the ROI are greater than a threshold. Then, if there is a FLOX data file available, the code will also calculate several statistical metrics comparing FLEX SIF (Sun-induced Fluorescence) data and FLOX SIF data.

After the process of FLEX images, this prototype will search for S2 images whose dates and time are the same to that of FLEX images; if there are not the same ones, it will find the S2 images with the nearest dates and time. Then, it will check the cloud coverage inside ROIs, and if it is below a specific threshold, it performs calculation of CV, and flags whether the CV is not greater than a given threshold, 1 if true and 0 if false. Otherwise, the S2 image will be skipped.

The final output files are in .csv format. For FLEX images, this prototype will generate a .csv file for each FLEX image containing the average and the standard deviation of the value “Sif Emission Spectrum“ inside the ROI. After processing all FLEX images, it will create a single .csv file to record the dates and the time of the FLEX images that have sufficient vegetation pixels inside the ROI. If a .csv file of FLOX data is provided, the code will also calculate several statistical metrics to compare several bands of FLEX and FLOX.

For the output of S2 images, it contains names of site, numbers of valid pixels, percentages of valid pixels, values of CV and the corresponding flags. For the output of FLEX images, if a site has sufficient vegetation pixels in its ROI, then this prototype will generate two .csv files, one containing vegetation pixels percentage and another containing the average and standard deviation values of all bands; otherwise, this prototype will generate only one .csv file, containing vegetation pixels percentage.

**Input variables:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable name** | **Descriptive Name** | **Type** | **Source** | **Remarks** |
| df\_Site | .csv file of sites | Pandas. Dataframe | Site.csv | -Mandatory |
| image\_L1C\_B08  image\_L2A\_B04  image\_L2A\_B08 | Raw S2 images | Rasterio.io.DatasetReader | Input S2 Images | -Mandatory |
| VegetationPixel.nc | Vegetation Pixel mask image file | netcdf | Input Images | -Mandatory |
| Image.nc | Raw FLEX images | netcdf | Input FLEX Images | -Mandatory |
| \_threshold\_cv | The threshold used to flag a CV value | Float | Optional Input.ini | -Optional; can’t be negative; default 0.2 |
| \_area | The side-length (meters) of squared ROI | Int | Optional Input.ini | -Optional; must be a multiple of 10; default 900 |
| \_cloud | The cloud coverage threshold to determine whether a S2 image is usable | Float | Optional Input.ini | -Optional; must be between 0 and 1; default 0.5 |
| bool\_delete\_cache | A Boolean value which determines whether the cache files will be deleted upon completion of the code | Boolean | Optional Input.ini | -Optional; default False |
| \_vegetationPixel | The vegetation pixel threshold to determine whether a FLEX image is usable | Float | Optional Input.ini | -Optional; must be between 0 and 1; default 0.5 |
| df\_flox | The FLOX SIF data of all sites | Pandas. Dataframe | flox\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv | -Optional |

**Output variables:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable name** | **Descriptive Name** | **Type** | **Remarks** |
| df\_sif | Dataframe of the output of calculation of SIF of FLEX images | Pandas.Dataframe |  |
| list\_site\_name | List of site names with which there are input FLEX images overlapping | List |  |
| list\_flex\_filename | List of filenames of input FLEX images which have enough vegetation pixels | List |  |
| list\_flex\_date | List of dates of input FLEX images which have enough vegetation pixels | List |  |
| list\_flex\_time | List of timestamps of input FLEX images which have enough vegetation pixels | List |  |
| temp\_avg\_max\_red | Average of 684 nm of the SIF emission spectrum inside the ROI of an input FLEX image | Float64 |  |
| temp\_avg\_max\_farred | Average of 740 nm of the SIF emission spectrum inside the ROI of an input FLEX image | Float64 |  |
| temp\_avg\_O2A | Average of 760 nm of the SIF emission spectrum inside the ROI of an input FLEX image | Float64 |  |
| temp\_avg\_O2B | Average of 686 nm of the SIF emission spectrum inside the ROI of an input FLEX image | Float64 |  |
| temp\_avg\_int | Average of the total integrated SIF inside the ROI of an input FLEX image | Float64 |  |
| temp\_avg\_O2A\_un | Average of the uncertainty of 760 nm of the SIF emission spectrum inside the ROI of an input FLEX image | Float64 |  |
| temp\_avg\_O2B\_un | Average of the uncertainty of 686 nm of the SIF emission spectrum inside the ROI of an input FLEX image | Float64 |  |
| list\_r\_2 | List which contains the coefficients of determination between several bands of FLEX and FLOX. | List |  |
| list\_rmse | List which contains the root mean squared errors between several bands of FLEX and FLOX. | List |  |
| list\_mean\_residual | List which contains the mean residuals between several bands of FLEX and FLOX. | List |  |
| list\_random\_uncertainty | List which contains the random uncertainties between several bands of FLEX and FLOX. | List |  |
| df\_output | Dataframe of the output of cloud filtering and cv flagging of Sentinel-2 images | Pandas.Dataframe |  |

**Applicable Version of ATBD**

**Applicable Version of Test Data Set**

* Version: V02.00.01
* Required Inputs:
  + Site.csv.
  + All S2 images inside “Input S2 Images” folder should be organized as follows: “Input S2 Images\$SiteName\$DateTTime\L1C\” and “Input S2 Images\$SiteName\$DateTTime\L2A\”. Change the part $SiteName to the actual names of sites, and $DateTTIme to the actual dates and times of your S2 images such as “20230821T100601”. The L1C and L2A images must cover the exact same region and have the same dates.
  + All FLEX images inside “Input FLEX Images” folder, organized as follows”Input FLEX Images\Site Name\”.
* Optional Inputs:
  + Optional Input.ini: Open it in a text editor and edit fields based on user requirements.
  + flox\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv: Optional FLOX data input csv file.
* Expected Outputs:
  + SiteName\FlexImageName – sif.csv
  + flex\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv
  + statistics.csv
  + Output\_S2.csv
  + Usable FLEX Images.csv

**Installation Instructions**

Programming language: Python3.11

Libraries: (Different versions of packages might work as well)

* Numpy >=1.26.4
* pandas >=2.2.2
* shapely >=2.0.5
* geopandas >=1.0.1
* matplotlib >=3.9.1
* lxml >=5.2.2
* beautifulsoup4 >=4.12.3
* GDAL ==3.9.1
* rasterio ==1.3.10
* configparser >=7.1.0
* xarray >=2024.10.0
* scipy >=1.14.1
* netcdf4 >=1.7.2
* gdal >= 3.9.1

Steps:

1. Make sure python3 is installed
2. Install the necessary python libraries using pip
   1. pip install -r /path/to/requirements.txt
3. Alternatively, if you are using Anaconda, you can directly create a virtual environment using “environment.yml”. Open cmd.exe via Anaconda and move to the location of the unzipped folder. Then type “conda env create -f environment.yml”.

**Execution Instructions:**

1. Create a virtual python environment using Anaconda or any other similar managers.
2. Ensure you have Python, and the necessary libraries installed in the virtual env. (see Installation Instructions)
3. If you don’t want to add more sites, skip this step and jump to the step 4. Otherwise, open “Sites.csv” and add their names, latitudes and longitudes. The “Site Name” must be unique.
4. For input FLEX images, firstly, you must create subfolders using all available site names inside “Input FLEX Images”, and then you can put your FLEX images in corresponding subfolders. Make sure the folder structure is identical to “Input Flex Images\IT-SR2\PRS\_TD\_20230616\_101431.nc”.
5. If you have available FLOX data, name it as “flox\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv” and put it in the same folder of “Main.py”.
6. To process S2 images, you also need to create subfolders using all available site names inside “Input S2 Images”.

The folder structure is as follows: “Input S2 Images\$SiteName\$DateTTime\L1C\” and “Input S2 Images\$SiteName\$DateTTime\L2A\”. The L1C and L2A images must cover the exact same region and have the same dates. Then put the S2 images into the corresponding subfolders.

1. Then put your S2 images into the corresponding subfolders.
2. If you don’t want to modify those optional variables, skip this step and jump to the step 8. If you want, open “Optional Input.ini” and write your own values inside it, after carefully reading the comments inside that .ini file.
3. Open the “Main.py” in an IDE (such as Spyder or Visual Studio) inside the virtual python environment you have just created, like Spyder or VS Code, and run it.
4. Upon successful execution, the output results can be found inside the folder “Output”.

**For FLEX images, expected output of an input image can be:**

|  |  |  |
| --- | --- | --- |
| SIF | Average | STD |
| Sif Emission Spectrum\_sif\_wavelength\_grid=670 | 0.275985 | 0.043109 |
| Sif Emission Spectrum\_sif\_wavelength\_grid=672 | 0.326899 | 0.051894 |
| Sif Emission Spectrum\_sif\_wavelength\_grid=674 | 0.384171 | 0.061756 |
| Sif … | … | … |

**flex\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv**

SIF\_FARRED\_max, SIF\_FARRED\_max\_wvl, SIF\_RED\_max, SIF\_RED\_max\_wvl, SIF\_O2B, SIF\_O2A, SIF\_int, SIF\_O2B\_un, SIF\_O2A\_un

**statistics.csv**

R Square (Coefficient of Determination), RMSE, Mean residual, Random uncertainty

**Usable FLEX Images.csv**

|  |  |  |  |
| --- | --- | --- | --- |
| Site | FLEX Filename | FLEX Date | FLEX Time |
| IT-JDS | **PRS\_TD\_20240726\_101410.nc** | **20240726** | **101410** |

**Output\_S2.csv**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Site | FLEX Filename | Valid Pixels L1C | Valid Pixels L2A | Valid Pixels Percentage L1C | Valid Pixels Percentage L2A | CV | Flag |
| SanRossore | PRS\_TD\_20230616\_101431.nc | 8100 | 8100 | 1 | 1 | 0.251332962 | 0 |

## Source Code

**Main.py**

# ---------------------------------------------------------------------------- #

#                            Import Python Packages                            #

# ---------------------------------------------------------------------------- #

import os

import time

import csv

from datetime import datetime

import shutil

import configparser

import numpy as np

import pandas as pd

import rasterio as rio

# Don't remove rasterio.mask this package otherwise it could cause errors

import rasterio.mask

# ---------------------------------------------------------------------------- #

#                                 Import Class                                 #

# ---------------------------------------------------------------------------- #

from class\_calval import FLEX, S2

from class\_sif\_calculation import SIF

# ---------------------------------------------------------------------------- #

#                                   Main Code                                  #

# ---------------------------------------------------------------------------- #

def main():

    time\_start = time.time()

    print("Code starts!")

    # Initiate classes

    flex = FLEX()

    sif = SIF()

    # -------------------------------- FLEX Images ------------------------------- #

    config = configparser.ConfigParser()

    # Read optional input

    config.read(flex.file\_optional)

    bool\_optional\_input = False

    if config["FLEX"]["area\_ROI"]:

        temp = config["FLEX"]["area\_ROI"]

        flex.\_area\_ROI = temp

        print(f"The area of the ROI has been modified to {temp} (m) x {temp} (m)!")

        bool\_optional\_input = True

    if config["FLEX"]["threshold\_vegetation\_pixel"]:

        temp = config["FLEX"]["threshold\_vegetation\_pixel"]

        flex.\_vegetation\_pixel = temp

        print(f"The threshold of vegetation pixel has been modified to {temp}!")

        bool\_optional\_input = True

    # Log if no optional input is provided

    if not bool\_optional\_input:

        print("No optional input found for FLEX images. The code will use the default values!")

    # Read Sites.csv

    df\_site = flex.get\_site\_info()

    print("Starting to proceed all FLEX images!")

    print("-"\*20)

    list\_site\_name = []

    list\_flex\_date = []

    list\_flex\_time = []

    list\_flex\_filename = []

    # Iterate each site in Sites.csv!

    for index, row in df\_site.iterrows():

        # Read site name, lat and lon

        temp\_site\_name = row['Sites']

        temp\_site\_lat = row['Latitude']

        temp\_site\_lon = row['Longitude']

        # Check if the current site has FLEX images!

        temp\_site\_path\_input = os.path.join(flex.path\_input,temp\_site\_name)

        if not os.path.exists(temp\_site\_path\_input):

            print(f"{temp\_site\_name} doesn't have any input FLEX images! This site has been skipped!")

            print("-"\*20)

            continue

        # Check if the current site has FLEX images!

        temp\_site\_flex\_images\_list = os.listdir(temp\_site\_path\_input)

        temp\_site\_flex\_images\_list\_nc = [i for i in temp\_site\_flex\_images\_list if i.endswith('.nc')]

        temp\_site\_flex\_images\_num = len(temp\_site\_flex\_images\_list\_nc)

        if temp\_site\_flex\_images\_num == 0:

            print(f"{temp\_site\_name} doesn't have any input FLEX images! This site has been skipped!")

            print("-"\*20)

            continue

        # Real work begins

        print(f"{temp\_site\_name} has {temp\_site\_flex\_images\_num} FLEX images!")

        # Processing all the FLEX images for the current site

        for i in range(temp\_site\_flex\_images\_num):

            temp\_flex\_filename = temp\_site\_flex\_images\_list\_nc[i]

            # Check filename

            flex.check\_filename(temp\_flex\_filename)

            ## Start to process the FLEX image

            print(f"Now starting with No.{i + 1} FLEX image '{temp\_flex\_filename}' of the site {temp\_site\_name}")

            # Veg pixel filter; pending!

            if flex.\_vegetation\_pixel:

                print("There are enough vegetation pixels inside the ROI in this image! The date and the time of this image will be recorded!")

                list\_site\_name.append(temp\_site\_name)

                list\_flex\_filename.append(temp\_flex\_filename)

                list\_flex\_date.append(temp\_flex\_filename.split('.')[0].split('\_')[-2])

                list\_flex\_time.append(temp\_flex\_filename.split('.')[0].split('\_')[-1])

            else:

                print("There are enough vegetation pixels inside the ROI in this image! The date and the time of this image will be recorded!")

            # Calculation of SIF

            flex.cal\_SIF(temp\_site\_name, temp\_flex\_filename, temp\_site\_lon, temp\_site\_lat, bool\_save = True)

            sif.SIF\_avg\_output(temp\_site\_name, temp\_flex\_filename, temp\_site\_lon, temp\_site\_lat, bool\_save = True)

            print(f"The FLEX image {temp\_flex\_filename} has been processed!")

        print(f"All the FLEX image(s) of the site {temp\_site\_name} has been processed!")

        print("-"\*20)

    print("All the input FLEX images have been processed!")

    print("-"\*20)

    print("Now starting to save the usable FLEX images to a .csv file!")

    with open(os.path.join(flex.path\_output,"Usable FLEX Images.csv"), 'w', newline='') as file:

        writer = csv.writer(file)

        writer.writerow(["Site","FLEX Filename","FLEX Date","FLEX Time"])

        for i in range(len(list\_site\_name)):

            writer.writerow([list\_site\_name[i],list\_flex\_filename[i],list\_flex\_date[i],list\_flex\_time[i]])

    print("The information of all usable FLEX images have been saved to a .csv file!")

    print("-"\*20)

    print("Now starting to merge the SIF results of all the usable FLEX images of all sites into one .csv file!")

    df\_flex\_merged = []

    df\_usable\_flex = pd.read\_csv(os.path.join(flex.\_path\_output,"Usable FLEX Images.csv"))

    for index, row in df\_usable\_flex.iterrows():

        temp\_site\_name = row['Site']

        temp\_filename = row['FLEX Filename']

        temp\_df = pd.read\_csv(os.path.join(flex.\_path\_output,temp\_site\_name,temp\_filename + " - sif avg.csv"))

        df\_flex\_merged.append(temp\_df)

    df\_flex\_merged = pd.concat(df\_flex\_merged, ignore\_index=True)

    df\_flex\_merged.to\_csv(os.path.join(flex.\_path\_output,"flex\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv"), index = False)

    print("The SIF results of all the usable FLEX images have been merged into one .csv file!")

    print("-"\*20)

    if os.path.exists(sif.file\_flox\_csv):

        print("The FLOX CSV file is found. The statistic calculation will be performed now!")

        sif.cal\_statistic(bool\_save = True)

    else:

        print("The FLOX CSV file is not found. The statistic calculation will be skipped!")

    print("-"\*20)

    # ----------------------------- Sentinel-2 Images ---------------------------- #

    print("Now starting to process Sentinel-2 images based on usable FLEX images found in the last section!")

    print("-"\*20)

    s2 = S2()

    if config["S2"]["threshold\_CV"]:

        temp = config["S2"]["threshold\_CV"]

        s2.\_threshold\_CV = temp

        print(f"The threshold of CV has been modified to {temp}!")

    if config["S2"]["threshold\_Cloud"]:

        s2.\_cloud = config["Optional Input"]["threshold\_Cloud"]

        print(f"The threshold of cloud coverage has been modified to {config['Optional Input']['threshold\_cloud']}!")

    if config["S2"]["area\_ROI"]:

        s2.\_area = config["Optional Input"]["area\_ROI"]

        print(f"The ROI has been modified to {config['Optional Input']['area\_ROI']} m by {config['Optional Input']['area\_ROI']} m!")

    if config["General"]["bool\_delete\_cache"] == "True":

        s2.\_bool\_delete\_cache = True

        # print("The cache foldeer will be deleted upon the completion of the code! ")

    # Read usable FLEX images .csv file

    df\_flex = pd.read\_csv(os.path.join(s2.\_path\_output,"Usable FLEX Images.csv"))

    # Initiate some empty lists to save the output values of each loop

    list\_valid\_site\_name = []

    list\_flex\_name = []

    list\_valid\_pixels\_l1c = []

    list\_valid\_percentage\_l1c = []

    list\_valid\_pixels\_l2a = []

    list\_valid\_percentage\_l2a = []

    list\_cv = []

    list\_flag = []

    if len(os.listdir(s2.path\_input)) !=0 :

        for index, row in df\_flex.iterrows():

            temp\_start\_time = time.time()

            temp\_site\_name = row["Site"]

            temp\_flex\_filename = row["FLEX Filename"]

            print(f"Now starting to process the site {temp\_site\_name} and its FLEX image {temp\_flex\_filename}!")

            temp\_flex\_date = row["FLEX Date"]

            temp\_flex\_time = row["FLEX Time"]

            temp\_flex\_datetime = datetime.strptime(str(temp\_flex\_date)+str(temp\_flex\_time), '%Y%m%d%H%M%S')

            # Get site name

            temp\_site\_index = df\_site.index[df\_site['Sites'] == temp\_site\_name].tolist()[0]

            temp\_site\_lat = df\_site["Latitude"][temp\_site\_index]

            temp\_site\_lon = df\_site["Longitude"][temp\_site\_index]

            temp\_path\_images\_site = os.path.join(s2.path\_input, temp\_site\_name)

            # Check usable FLEX images

            temp\_s2\_image\_final = os.listdir(temp\_path\_images\_site)

            if len(temp\_s2\_image\_final) == 0:

                print(f"No Sentinel-2 images found for the site {temp\_site\_name}. This site has been skipped!")

                print("-"\*20)

                continue

            else:

                temp\_s2\_image\_final = temp\_s2\_image\_final[0]

                for j in range(len(os.listdir(temp\_path\_images\_site))):

                    temp\_s2\_image = os.listdir(temp\_path\_images\_site)[j]

                    temp\_s2\_image\_datetime = temp\_s2\_image[0:8] + temp\_s2\_image[-6:]

                    temp\_s2\_image\_datetime = datetime.strptime(temp\_s2\_image\_datetime, '%Y%m%d%H%M%S')

                    if j == 0:

                        temp\_timediff\_final = temp\_s2\_image\_datetime - temp\_flex\_datetime

                    else:

                        temp\_timediff = temp\_s2\_image\_datetime - temp\_flex\_datetime

                        # print(f"Comparing S2 image {temp\_s2\_image} with FLEX image {temp\_flex\_filename}, the time difference is {temp\_timediff}")

                        if abs(temp\_timediff) < abs(temp\_timediff\_final):

                            temp\_timediff\_final = temp\_timediff

                            temp\_s2\_image\_final = temp\_s2\_image

                print(f"S2 image {temp\_s2\_image\_final} has the nearest date and time to the FLEX image {temp\_flex\_filename}")

                list\_valid\_site\_name.append(temp\_site\_name)

                list\_flex\_name.append(temp\_flex\_filename)

                print(f"The calculation and validation of the S2 image {temp\_s2\_image\_final} of the site {temp\_site\_name} has started! ")

                # Get paths to B8 of L1C and B4, B8 of L2A

                temp\_s2\_image\_path = os.path.join(temp\_site\_name,temp\_s2\_image\_final)

                s2.create\_cache\_subfolder(temp\_s2\_image\_path)

                path\_l1c\_b08\_raw, path\_l2a\_b04\_raw, path\_l2a\_b08\_raw, path\_l1c\_mask, path\_l2a\_mask, path\_l1c\_xml\_ds, path\_l1c\_xml\_tl, path\_l2a\_xml\_ds = s2.get\_path\_images(temp\_s2\_image\_path)

                # Read images

                image\_l1c\_b08 = rio.open(path\_l1c\_b08\_raw)

                image\_l2a\_b04 = rio.open(path\_l2a\_b04\_raw)

                image\_l2a\_b08 = rio.open(path\_l2a\_b08\_raw)

                # Get the values of the images

                values\_l1c\_b08 = image\_l1c\_b08.read(1).astype(np.int32)

                values\_l2a\_b04 = image\_l2a\_b04.read(1).astype(np.int32)

                values\_l2a\_b08 = image\_l2a\_b08.read(1).astype(np.int32)

                # Create a shapefile of our ROI and another one of the mask

                gdf\_roi = s2.create\_shapefile(image\_l1c\_b08, image\_l2a\_b04, temp\_site\_name, temp\_site\_lat, temp\_site\_lon)

                # Read masks of opaque clouds, cirrus clouds and snow ice areas

                mask\_l1c = rio.open(path\_l1c\_mask)

                mask\_l2a = rio.open(path\_l2a\_mask)

                mask\_l1c\_opaque\_clouds = mask\_l1c.read(1)

                mask\_l1c\_cirrus\_clouds = mask\_l1c.read(2)

                mask\_l1c\_snowice\_areas = mask\_l1c.read(3)

                mask\_l2a\_opaque\_clouds = mask\_l2a.read(1)

                mask\_l2a\_cirrus\_clouds = mask\_l2a.read(2)

                mask\_l2a\_snowice\_areas = mask\_l2a.read(3)

                # Check if all three masks are empty. If not empty, we should check if the masked

                mask\_l2a\_combined = mask\_l2a\_opaque\_clouds + mask\_l2a\_cirrus\_clouds + mask\_l2a\_snowice\_areas

                mask\_l1c\_combined = mask\_l1c\_opaque\_clouds + mask\_l1c\_cirrus\_clouds + mask\_l1c\_snowice\_areas

                temp\_pass\_l1c, temp\_valid\_pixels\_l1c, temp\_valid\_pixels\_percentage\_l1c = s2.cal\_valid\_pixels(temp\_site\_name, image\_l1c\_b08, mask\_l1c\_combined, gdf\_roi, note = "L1C")

                temp\_pass\_l2a, temp\_valid\_pixels\_l2a, temp\_valid\_pixels\_percentage\_l2a = s2.cal\_valid\_pixels(temp\_site\_name, image\_l2a\_b04, mask\_l2a\_combined, gdf\_roi, note = "L2A")

                list\_valid\_pixels\_l1c.append(temp\_valid\_pixels\_l1c)

                list\_valid\_pixels\_l2a.append(temp\_valid\_pixels\_l2a)

                list\_valid\_percentage\_l1c.append(temp\_valid\_pixels\_percentage\_l1c)

                list\_valid\_percentage\_l2a.append(temp\_valid\_pixels\_percentage\_l2a)

                if temp\_pass\_l1c and temp\_pass\_l2a:

                    # NDVI, Rad, NIRv

                    temp\_ndvi = s2.cal\_l2a\_ndvi(path\_l2a\_xml\_ds, values\_l2a\_b04, values\_l2a\_b08)

                    temp\_rad = s2.cal\_l1c\_rad(path\_l1c\_xml\_ds, path\_l1c\_xml\_tl, values\_l1c\_b08)

                    temp\_nirv = temp\_ndvi \* temp\_rad

                    print(f"The NIRv of {temp\_site\_name} has been calculated successfully!")

                    # Save NIRv.tif to cache folder

                    src = image\_l1c\_b08

                    out\_meta = src.meta

                    out\_meta.update({

                        "driver": "GTiff",

                        "dtype": "float64"

                    })

                    with rio.open(os.path.join(s2.\_path\_cache,temp\_site\_name,"NIRv.tif"), 'w', \*\*out\_meta) as dest:

                        dest.write(temp\_nirv, 1)

                    # Clip the NIRv.tif

                    image\_nirv = rio.open(os.path.join(s2.\_path\_cache,temp\_site\_name,"NIRv.tif"))

                    s2.clip\_raster\_by\_shapefile(temp\_s2\_image\_path, image\_nirv, gdf\_roi, suffix = "NIRv ROI")

                    # Read the clipped ROI NIRv.tif

                    image\_nirv\_roi = rio.open(os.path.join(s2.\_path\_cache,temp\_s2\_image\_path,"NIRv ROI.tif"))

                    values\_nirv\_roi = image\_nirv\_roi.read(1)

                    temp\_cv = s2.cal\_cv(values\_nirv\_roi)

                    temp\_flag = s2.cal\_flag(temp\_cv)

                    list\_cv.append(temp\_cv)

                    list\_flag.append(temp\_flag)

                    temp\_end\_time = time.time()

                    temp\_elapsed\_time = temp\_end\_time - temp\_start\_time

                    print(f"The calculation and validation of site {temp\_site\_name} and its S2 image {temp\_s2\_image\_final} has been finished successfully, which took {temp\_elapsed\_time:.2f} seconds! ")

                    print("-"\*20)

                else:

                    print(f"The calculation and validation of site {temp\_site\_name} and its S2 image {temp\_s2\_image\_final} has been skipped, due to exceeding invalid pixels!")

                    list\_cv.append(None)

                    list\_flag.append(None)

                    print("-"\*20)

                    continue

        # Loop finished, now we save the output to a new .csv file

        df\_output = pd.DataFrame({

            "Site": list\_valid\_site\_name,

            "FLEX Filename": list\_flex\_name,

            "Valid Pixels L1C": list\_valid\_pixels\_l1c,

            "Valid Pixels L2A": list\_valid\_percentage\_l2a,

            "Valid Pixels Percentage L1C": list\_valid\_percentage\_l1c,

            "Valid Pixels Percentage L2A": list\_valid\_percentage\_l2a,

            "CV": list\_cv,

            "Flag": list\_flag

        })

        # Delete cache folder?

        if s2.\_bool\_delete\_cache:

            del image\_l1c\_b08, image\_l2a\_b04, image\_l2a\_b08, image\_nirv, image\_nirv\_roi

            shutil.rmtree(s2.\_path\_cache)

            print("The cache folder and all its contents has been deleted permanently! ")

        print(f"Please find the final output.csv in the following folder: {s2.\_path\_output}")

        df\_output.to\_csv(os.path.join(s2.\_path\_output,"Output\_S2.csv"), index = False)

    else:

        print("WARNING: There are no Sentinel-2 images found inside the input folder. ")

    # ------------------------------ Code Terminates ----------------------------- #

    time\_end = time.time()

    time\_elapsed = time\_end - time\_start

    print(f"This python code has finished its work, and in totale it has taken {time\_elapsed:.2f} seconds!")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**class\_calval.py**

import os

from typing import Optional, Union

import re

import numpy as np

import pandas as pd

import xarray as xr

from bs4 import BeautifulSoup

import math

import shapely as shp

import geopandas as gpd

import rasterio as rio

import rasterio.mask

class CalVal:

    # Constuctor

    def \_\_init\_\_(self, cwd: Optional[str] = None):

        '''

        Initialize the class.

        Args:

            cwd (str, optional): Path to the working directory. If None, defaults to the script's location.

        Raises:

            TypeError: If 'cwd' is not a string.

        '''

        # ----------------------------------- Paths ---------------------------------- #

        # Current work directory (cwd)

        if cwd is not None:

            if isinstance(cwd, str):

                self.\_path\_main = cwd

            else:

                raise TypeError("The path to the working directory can only be a string!")

        else:

            # If

            self.\_path\_main = os.path.realpath(os.path.dirname(\_\_file\_\_))

        # Output folder

        self.\_path\_output = os.path.join(self.\_path\_main, "Output")

        # The absolute path to the input .csv file, where the info of all sites are saved.

        self.\_file\_site\_csv = os.path.join(self.\_path\_main, "Sites.csv")

        # The absolute path to the folder, where interim files are saved.

        self.\_path\_cache = os.path.join(self.\_path\_main, "Cache")

        # The absolute path to the folder of "Main - Optional Input.ini"

        self.\_file\_optional = os.path.join(self.\_path\_main, "Optional Input.ini")

        # A boolean variable which determines whether the cache files will be deleted unpon the completion of the code. False by default.

        self.\_bool\_delete\_cache = False

        ### Automatically check

        self.\_\_check\_site\_csv()

        self.\_\_check\_output()

        self.\_\_check\_cache()

    # ------------------------------ Private Methods ----------------------------- #

    # Create an output folder if not exists

    def \_\_check\_output(self):

        if not os.path.exists(self.path\_output):

            print("No output folder found! Creating a new output folder......")

            os.makedirs(self.path\_output)

            print(f"Output folder {self.path\_output} created successfully!")

    # Check if "Sites.csv" exists. Otherwise gives error.

    def \_\_check\_site\_csv(self):

        if not os.path.exists(self.file\_site\_csv):

            raise FileNotFoundError(f"The working directory {self.path\_main} doesn't contain the 'Sites.csv' file!")

        # else:

        #     print(f"'Sites.csv' file has been found in {self.\_path\_Main}!")

    # Create a cache folder if not exists

    def \_\_check\_cache(self):

        if not os.path.exists(self.\_path\_cache):

            os.makedirs(self.\_path\_cache)

            # print("Cache folder created successfully!")

        # if not self.\_bool\_delete\_cache:

        #     print("The cache files will be saved in the following folder: " + self.\_path\_cache)

        # else:

        #     print("Cache folder will be deleted upon the completion of the code.")

    # ------------------------------ Getter & Setter ----------------------------- #

    @property

    def path\_main(self):

        return self.\_path\_main

    @property

    def path\_output(self):

        return self.\_path\_output

    @property

    def file\_site\_csv(self):

        return self.\_file\_site\_csv

    @file\_site\_csv.setter

    def file\_siteCSV(self, value):

        if not isinstance(value, str):

            raise TypeError('The path to sites.csv must be a string!')

        else:

            self.\_file\_site\_csv = value

        self.\_\_check\_site\_csv()

    @property

    def file\_optional(self):

        return self.\_file\_optional

    @file\_optional.setter

    def file\_optional(self, value):

        if not isinstance(value, str):

            raise TypeError('The path to the optional input file must be a string!')

        elif not os.path.exists(value):

            raise FileNotFoundError(f"The file {self.\_file\_optional} is not found!")

        else:

            self.\_file\_optional = value

    # ------------------------------ Public Methods ------------------------------ #

    # Create a pandas dataframe using Sites.csv

    def get\_site\_info(self):

        df\_sites = pd.read\_csv(self.file\_site\_csv)

        # Site names

        site\_name = df\_sites["site\_code"]

        if not site\_name.notna().all():

            raise ValueError("Please make sure there is no missing site name in the .csv file!")

        # Convert site names to string

        site\_name\_str = [str(element) for element in site\_name]

        # Site lat

        site\_lat = df\_sites["latitude"]

        if not site\_lat.notna().all():

            raise ValueError("Please make sure there is no missing latitude in the .csv file!")

        if not pd.to\_numeric(site\_lat, errors='coerce').notna().all():

            raise ValueError("Please make sure latitudes are numeric values in the .csv file!")

        if not site\_lat.between(-90, 90).all():

            raise ValueError("Please make sure latitudes are within the range of -90 to 90!")

        # Site lon

        site\_lon = df\_sites["longitude"]

        if not site\_lon.notna().all():

            raise ValueError("Please make sure there is no missing latitude in the .csv file!")

        if not pd.to\_numeric(site\_lon, errors='coerce').notna().all():

            raise ValueError("Please make sure longtitudes are numeric values in the .csv file!")

        if not site\_lon.between(-180, 180).all():

            raise ValueError("Please make sure longtitudes are within the range of -180 to 180!")

        # Create final CSV

        df\_sites = pd.DataFrame({

            "Sites": site\_name\_str,

            "Latitude": site\_lat,

            "Longitude": site\_lon

        })

        print("'Sites.csv' read successfully!")

        return df\_sites

class FLEX(CalVal):

    # FLEX image resolution

    \_FLEX\_RESOLUTION = 300

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Input FLEX Images path

        self.\_path\_input = os.path.join(self.\_path\_main,"Input FLEX Images")

        # ROI size

        self.\_area\_roi = 900

        # Vegetation pixel percentage!

        self.\_vegetation\_pixel = 0.5

        # Check input flex images folder

        self.\_\_check\_input()

    # ------------------------------ Private Methods ----------------------------- #

    def \_\_check\_input(self):

        if not os.path.exists(self.path\_input):

            raise FileNotFoundError(f"The working directory {self.\_path\_main} doesn't contain the 'Input FLEX Images' Folder!")

        if not bool(os.listdir(self.path\_input)):

            raise FileNotFoundError(f"There is no FLEX image found inside the 'Input FLEX Images' folder!")

    # ----------------------------- Getter and Setter ---------------------------- #

    # Getter and setter for FLEX input path

    @property

    def path\_input(self):

        return self.\_path\_input

    @path\_input.setter

    def path\_input(self, value):

        self.\_path\_input = value

    # Getter and setter for ROI

    @property

    def area\_roi(self):

        return self.\_area\_roi

    @area\_roi.setter

    def area\_roi(self, value):

        if value < self.FLEX\_RESOLUTION:

            raise ValueError("The ROI must be greater than 300m x 300m!")

        if value % self.FLEX\_RESOLUTION != 0:

            raise ValueError("The ROI must contain complete FLEX pixel(s)!")

        self.\_area\_ROI = value

    # Getter for FLEX image resolution; no setter since it is a constant!

    @property

    def FLEX\_RESOLUTION(self):

        return self.\_FLEX\_RESOLUTION

    # Getter for vegetation pixel

    @property

    def vegetation\_pixel(self):

        return self.\_vegetation\_pixel

    @vegetation\_pixel.setter

    def vegetation\_pixel(self, value):

        if value < 0 or value > 1:

            raise ValueError("The valid vegetation pixel percentage must be between 0 and 1!")

        self.\_vegetation\_pixel = value

    # ------------------------------ Public Methods ------------------------------ #

    ## Check file name convention

    # PRS\_TD\_20230616\_101431.nc

    def check\_filename(self, filename: str) -> None:

        '''

        A function used to check whether the filename of a FLEX image corresponds to the correct format. If not, it will raise an error.

        '''

        if not re.fullmatch(r"^PRS\_TD\_\d{8}\_\d{6}\.nc$", filename):

            raise ValueError(f"The filename '{filename}' is not correct! It should have the same format as 'PRS\_TD\_20230616\_101431.nc'!")

    ## SIF Calculation

    def cal\_SIF(self, site\_name: str, filename: str, site\_lon: Union[int, float], site\_lat: Union[int, float], bool\_save: Optional[bool] = False) -> None:

        temp\_ds = rio.open(f'netcdf:{os.path.join(self.path\_input,site\_name,filename)}:Leaf Area Index')

        # Get the pixel where there is the site

        temp\_index\_x, temp\_index\_y = temp\_ds.index(site\_lon,site\_lat)

        # print(f"FLEX image '{temp\_FLEX\_filename}' opened succesfully!")

        temp\_list\_sif\_name = []

        temp\_list\_sif\_avg = []

        temp\_list\_sif\_std = []

        temp\_ds = xr.open\_dataset(os.path.join(self.path\_input, site\_name, filename))

        temp\_name\_rar = list(temp\_ds.data\_vars)

        for var\_name in temp\_name\_rar:

            if "Sif Emission Spectrum\_sif\_wavelength\_grid" in var\_name:

                temp\_array = temp\_ds[var\_name][(temp\_index\_x-1):(temp\_index\_x+2),(temp\_index\_y-1):(temp\_index\_y+2)].values

                temp\_list\_sif\_name.append(var\_name)

                temp\_AVG = np.average(temp\_array).item()

                temp\_list\_sif\_avg.append(temp\_AVG)

                temp\_STD = np.std(temp\_array).item()

                temp\_list\_sif\_std.append(temp\_STD)

        temp\_df\_sif = pd.DataFrame({

            "SIF": temp\_list\_sif\_name,

            "Average": temp\_list\_sif\_avg,

            "STD": temp\_list\_sif\_std

        })

        if bool\_save:

            if not os.path.exists(os.path.join(self.\_path\_output,site\_name)):

                os.makedirs(os.path.join(self.\_path\_output,site\_name))

            temp\_df\_sif.to\_csv(os.path.join(self.\_path\_output,site\_name,filename + " - sif.csv"), index = False)

class S2(CalVal):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.\_\_S2\_RESOLUTION = 10

        # Set a threshold of CV. You can change this value according to your own need, but note that there is no sense if the threshold is not greater than 0. The threshold of CV is set to 0.2 by default.

        self.\_threshold\_cv = 0.2

        # Set the side-length of the ROI. The ROI must be a squared area. This value must be a multiple of 10. The side-length of the ROI is 900 meters by default.

        self.\_area = 900

        # Cloud coverage

        self.\_cloud = 0.5

        # The absolute path of the S2 images

        self.\_path\_input = os.path.join(self.\_path\_main,"Input S2 Images")

        self.\_\_check\_input()

        self.\_\_check\_FLEX()

    # ------------------------------ Private Methods ----------------------------- #

    def \_\_check\_input(self):

        if not os.path.exists(self.path\_input):

            raise FileNotFoundError(f"The working directory {self.\_path\_main} doesn't contain the 'Input S2 Images' Folder!")

        if not bool(os.listdir(self.path\_input)):

            raise FileNotFoundError(f"There is no S2 image found inside the 'Input FLEX Images' folder!")

    def \_\_check\_FLEX(self):

        if not os.path.exists(os.path.join(self.\_path\_output,"Usable FLEX Images.csv")):

            raise FileNotFoundError("There is no available FLEX image found ")

    # ------------------------------ Private Members ----------------------------- #

    # Getter

    @property

    def s2\_resolution(self):

        return self.\_\_S2\_RESOLUTION

    # Setter

    @s2\_resolution.setter

    def s2\_resolution(self, value):

        if value:

            raise AttributeError("Cannot modify the spatial resolution of Sentinel-2 images!")

    # Getter

    @property

    def path\_input(self):

        return self.\_path\_input

    # ----------------------------- Protected Members ---------------------------- #

    # Getter

    @property

    def threshold\_cv(self):

        return self.\_threshold\_cv

    # Setter

    @threshold\_cv.setter

    def threshold\_cv(self, value):

        if value <= 0:

            raise ValueError("The threshold of CV should be greater than 0!!!")

        self.\_threshold\_cv = value

    # Getter

    @property

    def area(self):

        return self.\_area

    # Setter

    @area.setter

    def area(self, value):

        if value < self.\_\_S2\_RESOLUTION:

            raise ValueError()

        if value % self.\_\_S2\_RESOLUTION != 0:

            raise ValueError("The size of the ROI must be a multiple of 100 squared meters!!!")

        self.\_area = value

    # Getter

    @property

    def cloud(self):

        return self.\_cloud

    # Setter

    @cloud.setter

    def cloud(self, value):

        if value < 0 or value > 1:

            raise ValueError("The cloud coverage must be between 0 and 1!!!")

        self.\_cloud = value

    # ------------------------------ Public Methods ------------------------------ #

    def create\_cache\_subfolder(self, path):

        temp = os.path.join(self.\_path\_cache,path)

        if not os.path.exists(temp):

            os.makedirs(temp)

    # Get paths to S2 images and mask images. In this case we need B8 image of L1C and B4, B8 images of L2A

    def get\_path\_images(self, path\_s2\_image):

        # L1C

        temp\_path\_l1c = os.path.join(self.path\_input,path\_s2\_image,"L1C")

        for path, subdirs ,files in os.walk(temp\_path\_l1c):

            for name in files:

                temp = os.path.join(path, name)

                if "IMG\_DATA" in temp and temp[-3:] == 'jp2' and "B08" in temp:

                    path\_l1c\_b08\_raw = temp

                # Path to the mask file

                if "MSK\_CLASSI\_B00" in temp and temp[-3:] == 'jp2':

                    path\_l1c\_mask = temp

                # Get the path to the XML file "MTD\_DS" of L1C raster, where there are values of "U", "Solar Irradiance", "Quantification Value" and "Radiometric Offset"

                if "MTD\_DS.xml" in temp:

                    path\_l1c\_xml\_ds = temp

                # Get the path to the XML file "MTD\_TL" of L1C raster, where there are values (matrices) of "Solar Zenith Angle".

                if "MTD\_TL.xml" in temp:

                    path\_l1c\_xml\_tl = temp

        # L2A

        temp\_path\_l2a = os.path.join(self.path\_input,path\_s2\_image,"L2A")

        for path, subdirs, files in os.walk(temp\_path\_l2a):

            for name in files:

                temp = os.path.join(path, name)

                if temp[-3:] == 'jp2'in temp and "10m" in temp and "B04" in temp :

                    path\_l2a\_b04\_raw = temp

                if temp[-3:] == 'jp2'in temp and "10m" in temp and "B08" in temp :

                    path\_l2a\_b08\_raw = temp

                # Path to the mask file

                if "MSK\_CLASSI\_B00" in temp and temp[-3:] == 'jp2':

                    path\_l2a\_mask = temp

                # Get the path to the XML file "MTD\_DS" of L1C raster, where there are values of "Quantification Value" and "Radiometric Offset"

                if "MTD\_DS.xml" in temp:

                    path\_l2a\_xml\_ds = temp

        if not os.path.exists(temp\_path\_l1c) or not os.path.exists(temp\_path\_l2a):

            print("User Error: Please organise the input S2 images in correct folder structure. ")

            raise FileNotFoundError(f"The input S2 images folder {path\_s2\_image} doesn't contain the correct folder structure or doesn't contain S2 images! Please check the input S2 images folder!")

        else:

            return path\_l1c\_b08\_raw, path\_l2a\_b04\_raw, path\_l2a\_b08\_raw, path\_l1c\_mask, path\_l2a\_mask, path\_l1c\_xml\_ds, path\_l1c\_xml\_tl, path\_l2a\_xml\_ds

    def create\_shapefile(self, img\_l1c, img\_l2a, site\_name, site\_lat, site\_lon, bool\_toSave = False):

        # Get the crs of input L1C image and L2A image

        crs\_l1c = img\_l1c.crs.data["init"].split(":")[1]

        crs\_l2a = img\_l2a.crs.data["init"].split(":")[1]

        # In the case that L1C and L2A have different crs, give an error. But this won't happen if the input images are correct.

        if crs\_l2a != crs\_l1c:

            raise SystemExit("Stop right there!")

        crs\_final = 'EPSG:' + crs\_l1c

        # Create a point shapefile based on the site, using Lon-Lat

        df\_4326 = pd.DataFrame({

            "Site": [site\_name],

            "Latitude": [site\_lat],

            "Longitude": [site\_lon]

        })

        gdf\_4326 = gpd.GeoDataFrame(

            df\_4326,

            geometry = gpd.points\_from\_xy(df\_4326['Longitude'], df\_4326['Latitude']),

            crs = "EPSG:4326"

        )

        gdf\_new = gdf\_4326.copy()

        gdf\_new = gdf\_new.to\_crs(crs\_final)

        # First we retrieve the x, y coordinate of our site

        site\_x = gdf\_new.geometry.x.values[0]

        site\_y = gdf\_new.geometry.y.values[0]

        site\_row, site\_col = img\_l2a.index(site\_x, site\_y)

        site\_pixel\_x, site\_pixel\_y = img\_l2a.xy(site\_row, site\_col)

        # Calculate the "cardinal" distance

        side\_length\_half = self.\_area / 2

        if side\_length\_half % 2 == 0:

            # If the half of the side length is even, we need to add another 5 meters to make sure the pixels on the borders will not be omitted when we clip the raster images.

            length\_cardinal = side\_length\_half + 5

        else:

            length\_cardinal = side\_length\_half

        site\_x\_left\_new = site\_pixel\_x - length\_cardinal

        site\_x\_right\_new = site\_pixel\_x + length\_cardinal

        site\_y\_top\_new = site\_pixel\_y + length\_cardinal

        site\_y\_bottom\_new = site\_pixel\_y - length\_cardinal

        # Create a bounding box

        shp\_new = shp.box(site\_x\_left\_new, site\_y\_bottom\_new, site\_x\_right\_new, site\_y\_top\_new)

        # Create shapefile!

        gdf\_new = gpd.GeoDataFrame(

            pd.DataFrame({"0": ["0"]}),

            geometry=[shp\_new],

            crs = crs\_final

        )

        return gdf\_new

    def cal\_l2a\_ndvi(self, path\_l2a\_xml\_ds, values\_l2a\_b04, values\_l2a\_b08):

        # Read the DS xml file of L2A

        with open(path\_l2a\_xml\_ds, 'r') as f:

            data = f.read()

        bs\_l2a\_ds = BeautifulSoup(data, "xml")

        # Get the quantification value!

        quantification\_l2a = int(bs\_l2a\_ds.find("BOA\_QUANTIFICATION\_VALUE").text)

        # Get the radiometric offset!

        offset\_l2a\_b04 = int(bs\_l2a\_ds.find("BOA\_ADD\_OFFSET", {"band\_id": "3"}).text)

        offset\_l2a\_b08 = int(bs\_l2a\_ds.find("BOA\_ADD\_OFFSET", {"band\_id": "7"}).text)

        # Calculate NDVI of L2A!

        temp\_ndvi = ((values\_l2a\_b08 + offset\_l2a\_b08).astype(float) / quantification\_l2a - (values\_l2a\_b04 + offset\_l2a\_b04).astype(float) / quantification\_l2a) / ((values\_l2a\_b08 + offset\_l2a\_b08).astype(float) / quantification\_l2a + (values\_l2a\_b04 + offset\_l2a\_b04).astype(float) / quantification\_l2a )

        return temp\_ndvi

    def cal\_l1c\_rad(self, path\_l1c\_xml\_ds, path\_l1c\_xml\_tl, values\_l1c\_b08):

        with open(path\_l1c\_xml\_ds, 'r') as f:

            data = f.read()

        bs\_l1c\_ds = BeautifulSoup(data, "xml")

        # Get the quantification value!

        quantification\_l1c = int(bs\_l1c\_ds.find("QUANTIFICATION\_VALUE").text)

        # Get the radiometric offset!

        offset\_l1c = int(bs\_l1c\_ds.find("RADIO\_ADD\_OFFSET", {"band\_id": "7"}).text)

        # Get the U

        u\_l1c = float(bs\_l1c\_ds.find("U").text)

        # Get the solar irradiance

        solar\_irradiance\_l1c = float(bs\_l1c\_ds.find("SOLAR\_IRRADIANCE", {"bandId": "7"}).text)

        # Read the TL xml file of L1C

        with open(path\_l1c\_xml\_tl, 'r') as f:

            data = f.read()

        bs\_l1c\_ds = BeautifulSoup(data, "xml")

        # Get the sun zenith angle! There should be a 23 x 23 arrays in the xml. Now we save each row as an array and keep all these arrays into a list

        list\_sun\_zenith = []

        for row in bs\_l1c\_ds.find("Sun\_Angles\_Grid").find("Zenith").find\_all("VALUES"):

            temp\_list = row.text.split(" ")

            temp\_arr = np.array(temp\_list)

            temp\_arr = temp\_arr.astype(float)

            list\_sun\_zenith.append(temp\_arr)

        # Now we stack these nested-in-list arrays into a 2d array

        index = 0

        for arr in list\_sun\_zenith:

            if index == 0:

                arr\_sun\_zenith = arr

            else:

                arr\_sun\_zenith = np.vstack((arr\_sun\_zenith, arr))

            index = index + 1

        # Get the shape of L1C image, which should be (10980, 10980)

        shape\_l1c = values\_l1c\_b08.shape

        # Repeat each element of sun zenith angle array, in both axies. The final array should have a shape of (11500, 11500)

        arr\_sun\_zenith\_repeat = np.repeat(arr\_sun\_zenith, 500, axis = 1)

        arr\_sun\_zenith\_repeat = np.repeat(arr\_sun\_zenith\_repeat, 500, axis = 0)

        # Index only the first 10980 of each dimension

        arr\_sun\_zenith\_assigned = arr\_sun\_zenith\_repeat[0:shape\_l1c[0], 0:shape\_l1c[1]]

        # radiance = reflectance \* cos(radians(SunZenithAngle)) \* solarIrradiance \* U / pi

        temp\_Radiance = (values\_l1c\_b08 + offset\_l1c).astype(float)  \* np.cos(np.radians(arr\_sun\_zenith\_assigned)) \* solar\_irradiance\_l1c / quantification\_l1c / (math.pi \* (1 / u\_l1c))

        return temp\_Radiance

    def clip\_raster\_by\_shapefile(self, path, raster, shp, suffix = None):

        out\_image, out\_transform = rio.mask.mask(raster, shp.geometry, crop=True)

        out\_meta = raster.meta

        out\_meta.update({"driver": "GTiff",

                        "height": out\_image.shape[1],

                        "width": out\_image.shape[2],

                        "transform": out\_transform})

        with rio.open(os.path.join(self.\_path\_cache,path,suffix + ".tif"), "w", \*\*out\_meta) as dest:

            dest.write(out\_image)

    def cal\_valid\_pixels(self, path, raster, mask\_Combined, shp, note = None):

        if np.max(mask\_Combined) >= 1:

            # Upscale 60mx60m mask to 10mx10m without modifying any pixel values

            mask\_Combined\_Upscale = np.repeat(mask\_Combined, 6, axis = 0)

            mask\_Combined\_Upscale = np.repeat(mask\_Combined\_Upscale, 6, axis = 1)

            # Save this mask

            mask\_meta = raster.meta

            with rio.open(os.path.join(self.\_path\_cache,path,"Mask.tif"), "w", \*\*mask\_meta) as dest:

                dest.write(mask\_Combined\_Upscale, indexes = 1)

            # clip and save

            temp\_Mask = rio.open(os.path.join(self.\_path\_cache,path,"Mask.tif"))

            self.clip\_RasterbySHP(path, temp\_Mask, shp, suffix = "Mask ROI")

            temp\_MaskClipped = rio.open(os.path.join(self.\_path\_cache,path,"Mask ROI.tif"))

            temp\_MaskCombined = temp\_MaskClipped.read(1)

            if np.max(temp\_MaskCombined) >= 1:

                temp\_ValidPixels = np.count\_nonzero(temp\_MaskCombined == 0)

                temp\_InvalidPixels = np.count\_nonzero(temp\_MaskCombined != 0)

                temp\_TotalPixels = temp\_ValidPixels + temp\_InvalidPixels

                temp\_ValidPixelsRatio = temp\_ValidPixels / temp\_TotalPixels

                print(f"There are {temp\_ValidPixels} valid pixels in the S2 {note} image of {path}!")

                if temp\_ValidPixelsRatio >= self.\_cloud:

                    print(f"But the ratio of valid pixels is {temp\_ValidPixelsRatio:.2%}, equal to or greater than {self.\_cloud:.2%}, so we can use these S2 images. ")

                    temp\_Pass = True

                    return temp\_Pass, temp\_ValidPixels, temp\_ValidPixelsRatio

                else:

                    print(f"And the ratio of valid pixels is {temp\_ValidPixelsRatio:.2%}, lower than {self.\_cloud:.2%}, so we can't use these S2 images and hence we can't proceed. ")

                    temp\_Pass = False

                    return temp\_Pass, temp\_ValidPixels, temp\_ValidPixelsRatio

            else:

                print(f"All pixels in the S2 {note} image of {path} are valid! ")

                temp\_Pass = True

                return temp\_Pass, (self.\_area / 10) \*\* 2, 1

        else:

            print(f"All pixels in the S2 {note} image of {path} are valid! ")

            temp\_Pass = True

            return temp\_Pass, (self.\_area / 10) \*\* 2, 1

    def cal\_cv(self, value):

        return np.std(value) / np.mean(value)

    def cal\_flag(self, value):

        if value <= self.threshold\_cv:

            return 1

        else:

            return 0

**class\_sif\_calculation.py**

import os

import csv

from typing import Optional, Union

import numpy as np

import pandas as pd

import xarray as xr

import rasterio as rio

from class\_calval import FLEX

class SIF(FLEX):

    def \_\_init\_\_(self):

         super().\_\_init\_\_()

         self.\_band\_O2A = '760'

         self.\_band\_O2B = '686'

         self.\_band\_max\_red = '684'

         self.\_band\_max\_farred = '740'

         self.\_list\_header = ['site','filename','date','time','SIF\_FARRED\_max','SIF\_FARRED\_max\_wvl','SIF\_RED\_max','SIF\_RED\_max\_wvl','SIF\_O2B','SIF\_O2A','SIF\_int','SIF\_O2B\_un','SIF\_O2A\_un']

         self.\_file\_flox\_csv = os.path.join(self.\_path\_main,'flox\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv')

    # ------------------------------ Getter & Setter ----------------------------- #

    @property

    def band\_O2A(self):

        return self.\_band\_O2A

    @band\_O2A.setter

    def band\_O2A(self, value):

        self.\_band\_O2A = self.\_\_check\_input\_wavelength\_\_(value)

    @property

    def band\_O2B(self):

        return self.\_band\_O2B

    @band\_O2B.setter

    def band\_O2B(self, value):

        self.\_band\_O2B = self.\_\_check\_input\_wavelength\_\_(value)

    @property

    def band\_max\_red(self):

        return self.\_band\_max\_red

    @band\_max\_red.setter

    def band\_max\_red(self, value):

        self.\_band\_max\_red = self.\_\_check\_input\_wavelength\_\_(value)

    @property

    def band\_max\_farred(self):

        return self.\_band\_max\_farred

    @band\_max\_farred.setter

    def band\_max\_farred(self, value):

        self.\_band\_max\_farred = self.\_\_check\_input\_wavelength\_\_(value)

    @property

    def list\_header(self):

        return self.\_list\_header

    @list\_header.setter

    def list\_header(self, value):

        if value:

            raise ValueError('User error: Cannot modify variables to calculate!!!')

    @property

    def file\_flox\_csv(self):

        return self.\_file\_flox\_csv

    @file\_flox\_csv.setter

    def file\_flox\_csv(self, value):

        self.\_file\_flox\_csv = value

    # ------------------------------ Private Methods ------------------------------ #

    def \_\_check\_input\_wavelength\_\_(self, value):

        if isinstance(value, str):

            # The string must only contain numbers

            if value.isdigit():

                return value

            else:

                raise ValueError('Please enter the correct wavelength in nm (without the unit)!')

        elif isinstance(value, int):

            temp = str(value)

            return temp

        else:

            raise ValueError('Please enter the correct wavelength in nm (without the unit)!')

    # ------------------------------ Public Methods ------------------------------ #

    def SIF\_avg\_output(self, site\_name: str, filename: str, site\_lon: Union[int, float], site\_lat: Union[int, float], bool\_save: Optional[bool] = False) -> list:

        '''

        This function is used to calculate average values of a series of SIF metrics in a 3x3 pixel ROI of a FLEX image of a site.

        Parameters:

        - site\_name: str, the name of the site

        - filename: str, the name of the FLEX image

        - site\_lon: Union[int, float], the longitude of the site

        - site\_lat: Union[int, float], the latitude of the site

        Returns:

            list\_value: list, a list of average values of SIF metrics in a 3x3 pixel ROI of a FLEX image of a site: ['site','filename','date','time','SIF\_FARRED\_max','SIF\_FARRED\_max\_wvl','SIF\_RED\_max','SIF\_RED\_max\_wvl','SIF\_O2B','SIF\_O2A','SIF\_int','SIF\_O2B\_un','SIF\_O2A\_un']

        '''

        temp\_ds = rio.open(f'netcdf:{os.path.join(self.path\_input,site\_name,filename)}:Leaf Area Index')

        # Get the pixel where there is the site

        temp\_index\_x, temp\_index\_y = temp\_ds.index(site\_lon,site\_lat)

        print(f"FLEX image '{filename}' opened succesfully!")

        # Prepare for the output

        # Date + Time

        temp\_date = filename.split('.')[0].split('\_')[-2]

        temp\_time = filename.split('.')[0].split('\_')[-1]

        # Read dataset

        temp\_ds = xr.open\_dataset(os.path.join(self.path\_input, site\_name, filename))

        # Calculate metrics

        temp\_avg\_max\_red = np.average(temp\_ds[f"Sif Emission Spectrum\_sif\_wavelength\_grid={self.band\_max\_red}"][(temp\_index\_x-1):(temp\_index\_x+2),(temp\_index\_y-1):(temp\_index\_y+2)].values).item()

        temp\_avg\_max\_farred = np.average(temp\_ds[f"Sif Emission Spectrum\_sif\_wavelength\_grid={self.band\_max\_farred}"][(temp\_index\_x-1):(temp\_index\_x+2),(temp\_index\_y-1):(temp\_index\_y+2)].values).item()

        temp\_avg\_O2A = np.average(temp\_ds[f"Sif Emission Spectrum\_sif\_wavelength\_grid={self.band\_O2A}"][(temp\_index\_x-1):(temp\_index\_x+2),(temp\_index\_y-1):(temp\_index\_y+2)].values).item()

        temp\_avg\_O2B = np.average(temp\_ds[f"Sif Emission Spectrum\_sif\_wavelength\_grid={self.band\_O2B}"][(temp\_index\_x-1):(temp\_index\_x+2),(temp\_index\_y-1):(temp\_index\_y+2)].values).item()

        temp\_avg\_int = np.average(temp\_ds[f"Total Integrated SIF"][(temp\_index\_x-1):(temp\_index\_x+2),(temp\_index\_y-1):(temp\_index\_y+2)].values).item()

        temp\_avg\_O2A\_un = np.average(temp\_ds[f"Sif Emission Spectrum Uncertainty\_sif\_wavelength\_grid={self.band\_O2A}"][(temp\_index\_x-1):(temp\_index\_x+2),(temp\_index\_y-1):(temp\_index\_y+2)].values).item()

        temp\_avg\_O2B\_un = np.average(temp\_ds[f"Sif Emission Spectrum Uncertainty\_sif\_wavelength\_grid={self.band\_O2B}"][(temp\_index\_x-1):(temp\_index\_x+2),(temp\_index\_y-1):(temp\_index\_y+2)].values).item()

        # Output as a list

        list\_value = [site\_name,filename,temp\_date,temp\_time,temp\_avg\_max\_farred,self.band\_max\_farred,temp\_avg\_max\_red,self.band\_max\_red,temp\_avg\_O2B,temp\_avg\_O2A,temp\_avg\_int,temp\_avg\_O2A\_un,temp\_avg\_O2B\_un]

        # The header line

        if bool\_save:

            if not os.path.join(self.\_path\_output,site\_name):

                os.makedirs(os.path.join(self.\_path\_output,site\_name))

            with open(os.path.join(self.\_path\_output,site\_name,filename + " - sif avg.csv"), 'w', newline='') as file:

                writer = csv.writer(file)

                writer.writerow(self.list\_header)

                writer.writerow(list\_value)

        return list\_value

    def cal\_statistic(self, bool\_save: Optional[bool] = False) -> list:

        df\_flox = pd.read\_csv(self.file\_flox\_csv)

        df\_flox = df\_flox[['FLOX\_site\_code','SIF\_FARRED\_max','SIF\_RED\_max','SIF\_O2B','SIF\_O2A','SIF\_int','SIF\_O2B\_un','SIF\_O2A\_un']]

        df\_flox.rename(columns={'FLOX\_site\_code': 'site'}, inplace=True)

        df\_flex = pd.read\_csv(os.path.join(self.\_path\_output,"flex\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv"))

        df\_flex = df\_flex[['site','SIF\_FARRED\_max','SIF\_RED\_max','SIF\_O2B','SIF\_O2A','SIF\_int','SIF\_O2B\_un','SIF\_O2A\_un']]

        df\_merge = pd.merge(df\_flox,df\_flex,how='inner',on=['site'], suffixes=('\_flox','\_flex'))

        column\_pairs = [('SIF\_FARRED\_max\_flox','SIF\_FARRED\_max\_flex'),('SIF\_RED\_max\_flox','SIF\_RED\_max\_flex'),('SIF\_O2B\_flox','SIF\_O2B\_flex'),('SIF\_O2A\_flox','SIF\_O2A\_flex'),('SIF\_int\_flox','SIF\_int\_flex'),('SIF\_O2B\_un\_flox','SIF\_O2B\_un\_flex'),('SIF\_O2A\_un\_flox','SIF\_O2A\_un\_flex')]

        list\_r\_2 = []

        list\_rmse = []

        list\_mean\_residual = []

        list\_random\_uncertainty = []

        for pair in column\_pairs:

            temp\_r\_2 = self.cal\_r\_2(df\_merge[pair[1]],df\_merge[pair[0]])

            temp\_rmse = self.cal\_rmse(df\_merge[pair[1]],df\_merge[pair[0]])

            temp\_mean\_residual = self.cal\_mean\_residual(df\_merge[pair[1]],df\_merge[pair[0]])

            temp\_random\_uncertainty = self.cal\_random\_uncertainty(df\_merge[pair[1]],df\_merge[pair[0]],temp\_mean\_residual)

            list\_r\_2.append(temp\_r\_2)

            list\_rmse.append(temp\_rmse)

            list\_mean\_residual.append(temp\_mean\_residual)

            list\_random\_uncertainty.append(temp\_random\_uncertainty)

        if bool\_save:

            with open(os.path.join(self.\_path\_output,"statistics.csv"), 'w', newline='') as file:

                writer = csv.writer(file)

                writer.writerow(['SIF metrics','R^2','RMSE','Mean residual','Random uncertainty'])

                writer.writerow(['SIF\_FARRED\_max',list\_r\_2[0],list\_rmse[0],list\_mean\_residual[0],list\_random\_uncertainty[0]])

                writer.writerow(['SIF\_RED\_max',list\_r\_2[1],list\_rmse[1],list\_mean\_residual[1],list\_random\_uncertainty[1]])

                writer.writerow(['SIF\_O2B',list\_r\_2[2],list\_rmse[2],list\_mean\_residual[2],list\_random\_uncertainty[2]])

                writer.writerow(['SIF\_O2A',list\_r\_2[3],list\_rmse[3],list\_mean\_residual[3],list\_random\_uncertainty[3]])

                writer.writerow(['SIF\_int',list\_r\_2[4],list\_rmse[4],list\_mean\_residual[4],list\_random\_uncertainty[4]])

                writer.writerow(['SIF\_O2B\_un',list\_r\_2[5],list\_rmse[5],list\_mean\_residual[5],list\_random\_uncertainty[5]])

                writer.writerow(['SIF\_O2A\_un',list\_r\_2[6],list\_rmse[6],list\_mean\_residual[6],list\_random\_uncertainty[6]])

        return list\_r\_2, list\_rmse, list\_mean\_residual, list\_random\_uncertainty

    def cal\_r\_2(self, x: np.array, y: np.array) -> float:

        return 1 - ((x - y) \*\* 2).sum() / ((x - x.mean()) \*\* 2).sum()

    def cal\_rmse(self, x: np.array, y: np.array) -> float:

        return np.sqrt(((x - y) \*\* 2).mean())

    def cal\_mean\_residual(self, x: np.array, y: np.array) -> float:

        return (x - y).mean()

    def cal\_random\_uncertainty(self, x: np.array, y: np.array, mean\_residual: float) -> float:

        return ((x - y - mean\_residual) \*\* 2).mean()

## TDS and Auxiliar Files

**Example of installation file:**

**requirements.txt:**

Numpy >=1.26.4

pandas >=2.2.2

shapely >=2.0.5

geopandas >=1.0.1

matplotlib >=3.9.1

lxml >=5.2.2

beautifulsoup4 >=4.12.3

GDAL ==3.9.1

rasterio ==1.3.10

configparser >=7.1.0

xarray >=2024.10.0

scipy >=1.14.1

netcdf4 >=1.7.2

gdal >=3.9.1

**environment.yml**

name: cal-val-prototype

channels:

- conda-forge

- defaults

dependencies:

- python =3.11

- numpy >=1.26.4

- pandas >=2.2.2

- shapely >=2.0.5

- geopandas >=1.0.1

- lxml >=5.2.2

- beautifulsoup4 >=4.12.3

- gdal ==3.9.1

- rasterio ==1.3.10

- configparser >=7.1.0

- xarray >=2024.10.0

- scipy >=1.14.1

- netcdf4 >=1.7.2

- gdal >=3.9.1

**Input files:**

**Site.csv:**

|  |  |  |
| --- | --- | --- |
| **Site** | **Latitude** | **Longitude** |
| IT-CP2 | 41.7043 | 12.3573 |
| IT-JDS | 44.87431 | 11.9792 |
| IT-LIN | 41.1797 | -96.4404 |
| IT-SR2 | 43.732 | 10.291 |

**Input S2 Images\:**

…

**Input FLEX Images\**

**…**

**Optional Input.ini**

[OptionalInput]

; if left blank, these variables will use their default values.

; set the threshold of vegetation pixel for FLEX images. If the vegetation pixels inside the ROI of an image is lower than this threshold, the code will not record the date and the time of this FLEX image.

; It must be a value between 0 and 1 (can be equal to either 0 or 1).

; Default 0.5

threshold\_Vegetation =

; set the threshold of coefficient of variation (CV). If the CV is lower than this threshold, the code flags it as 1. Otherwise it will be flagged as 0.

; It must be a positive value.

; Default 0.2

threshold\_CV =

; set the threshold of cloud coverage. If the cloud coverage of an image is lower than this threshold, the code will pass the calculation of CV on that image.

; It must be a value between 0 and 1 (can be equal to either 0 or 1).

; Default 0.5

threshold\_Cloud =

; set the side-length (meters) of the squared ROI.

; It must be a multiple of 10m since we are using Sentinel-2 images whose spatial resolution is 10m.

; Default 900 (m)

area\_ROI =

; set this to True so that the "Cache" folder will be deleted when the code completes its work.

; It must be either True or False.

; Default False

bool\_DeleteCache = True

**Output file:**

For FLEX images, expected output of a input image can be:

|  |  |  |
| --- | --- | --- |
| SIF | Average | STD |
| Sif Emission Spectrum\_sif\_wavelength\_grid=670 | 0.275985 | 0.043109 |
| Sif Emission Spectrum\_sif\_wavelength\_grid=672 | 0.326899 | 0.051894 |
| Sif Emission Spectrum\_sif\_wavelength\_grid=674 | 0.384171 | 0.061756 |
| Sif … | … | … |

**flex\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv**

SIF\_FARRED\_max, SIF\_FARRED\_max\_wvl, SIF\_RED\_max, SIF\_RED\_max\_wvl, SIF\_O2B, SIF\_O2A, SIF\_int, SIF\_O2B\_un, SIF\_O2A\_un

**statistics.csv**

R Square (Coefficient of Determination), RMSE, Mean residual, Random uncertainty

**Usable FLEX Images.csv**

|  |  |  |  |
| --- | --- | --- | --- |
| Site | FLEX Filename | FLEX Date | FLEX Time |
| IT-JDS | **PRS\_TD\_20240726\_101410.nc** | **20240726** | **101410** |

**Output\_S2.csv**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Site | FLEX Filename | Valid Pixels L1C | Valid Pixels L2A | Valid Pixels Percentage L1C | Valid Pixels Percentage L2A | CV | Flag |
| IT-SR2 | PRS\_TD\_20230616\_101431.nc | 8100 | 8100 | 1 | 1 | 0.251332962 | 0 |