User Guide of PlotToSat

Milto Miltiadou*[1], Stuart Grieve [2], Paloma Ruiz Benito[3], Verónica Cruz-Alonso[3], Julen Astigarraga[3], Julián Tijerín Triviño[3], and Emily Lines[1]

[1]Department of Geography, University of Cambridge, Cambridge, United Kingdom

[2]School of Geography, Queen Mary University of London, London, United Kingdom

[3]Universidad de Alcalá, Department of Life Sciences, Forest Ecology and Restoration Group (FORECO), Alcalá de Henares, Spain

30/04/2024

*Corresponding author and main programmer.

License

PlotToSat is released under the GNU General Public Licence, Version 3. The full description of the usage licence is available here:<https://github.com/Art-n-MathS/PlotToSat/blob/main/License.txt>

The following paper should be cited in every publication using PlotToSat: Miltiadou, M., Grieve, S., Ruiz-Benito, P., Astigarraga, J., Cruz-Alonso, V., Triviño, J.T., and Lines, E. (2024) PlotToSat: A Tool for Generating Annual Time-Series from Sentinel-1 and Sentinel-2 at Each Plot Within a Plot Network for Machine Learning Applications *Computers & Geosciences*

Link to paper: <url>

The sample data are randomly created; they are well-distributed and lie within Peninsular Spain.

Installation

This user guide assumes basic knowledge of Google Earth Engine (GEE) and Python. A GEE account is also required: <https://code.earthengine.google.com/>

PlotToSat is implemented using the Python API of Google Earth Engine in IPython 3 (Jupyter). While Visual Studio Code (VS Code) is recommended due to its status as a free, open-source, and cross-platform editor, other IDEs should also work well.

The code is compatible with both Linux and Windows machines and is available at: <https://github.com/Art-n-MathS/PlotToSat>

Depending on your environment, you can install the dependencies as follows:

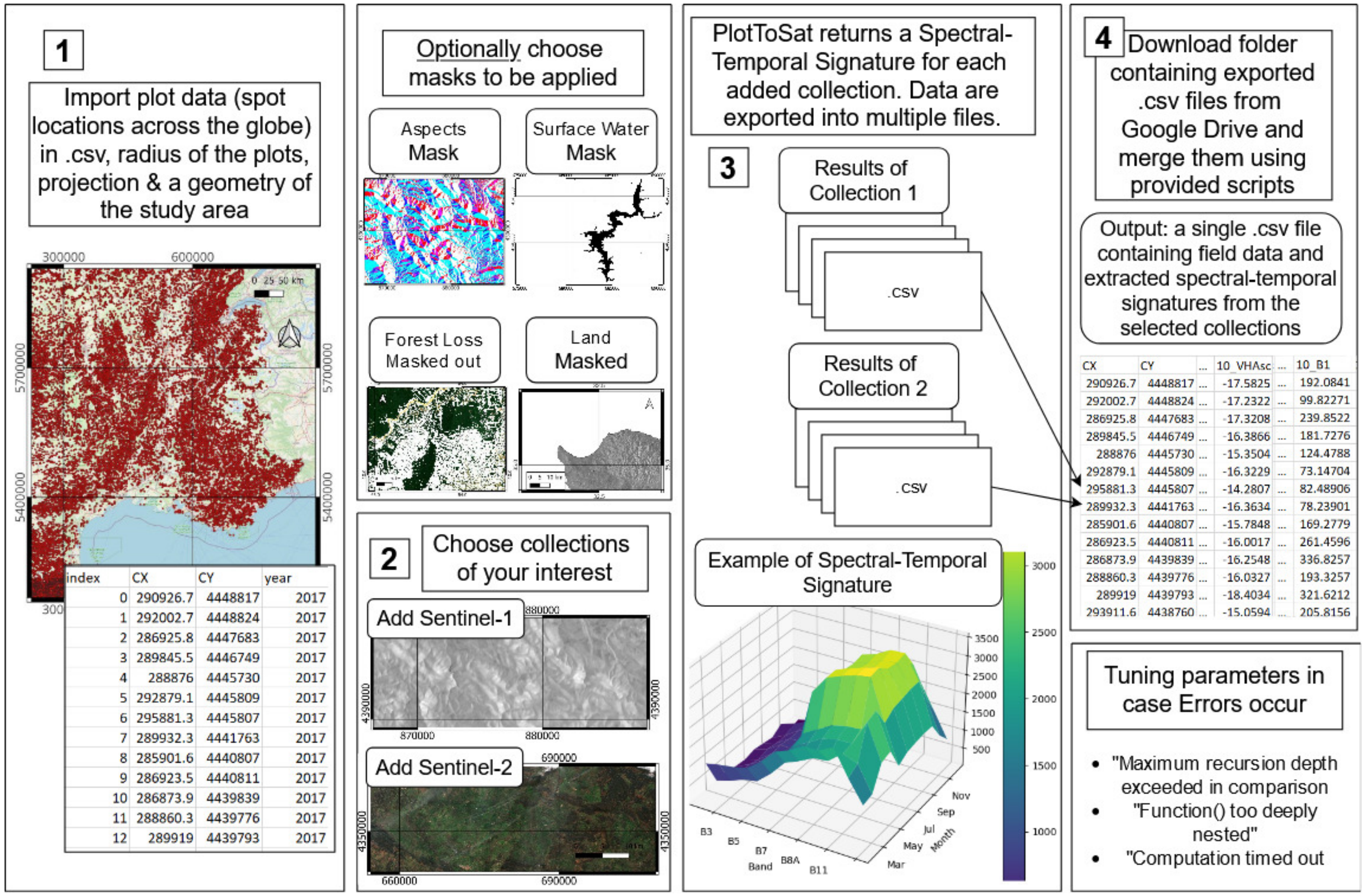
```
    pip install ipython pandas numpy earthengine-api
    conda install -c conda-forge earthengine-api ipython pandas numpy
```

1 Introduction

Forest ecologists gather data from predetermined sites known as plots. Plots are usually circular; defined by a their centre (latitude, longitude) and a radius. A plot network consists from a few to hundreds of thousands of plots, which are systematically placed to represent the forests of region e.g., a country. PlotToSat takes as input a plot network, and for each circular plot it extracts time-series of Sentinel-1 and/or Sentinel-2 collections for a given year –collections typically refer to organised sets of satellite data. For every plot, PlotTosat exports twelve values per band per collection corresponding to the twelve calendar months. Standard deviation is also provided for quality control.

Figure 1 shows the user-interface. The user-interface is divided into four main steps, along with two additional optional steps:

- Step 1: Define the essential input parameters and create an instance of PlotToSat. The compulsory input parameters required for creating the instance are: (1) a CSV file that lists the plot locations of a plot network, (2) the radius of the plots, (3) the projection of the plot data, (4) a geometry defining the study area, and (5) the year of interest.
- Optional Step 1: After Step 1, users can decide whether to apply any of the available masks (aspects, surface water, forest disturbance, and land mask) to the selected collections.
- Step 2: PlotToSat currently supports two EO collections (Sentinel-1 and Sentinel-2). Users can choose to add/use one or both of these collections.
- Step 3: Define the outputs (folder name and start of filenames); initially PlotToSat exports the time-series into multiple CSV files, which are stored on the user's Google Drive.
- Step 4: Download the folder containing the exported CSV files from Google Drive and merge them using the provided script.
- Optional Step 2: Tuning provided parameters to tackle potential errors that predominantly occur because Google Earth Engine (GEE) distributes processing power among its users by implementing processing limitations. While users may face GEE errors, we recommend solutions e.g., defining how many plots will be exported in each file in Step 3.


1
Import plot data (spot locations across the globe) in .csv, radius of the plots, projection & a geometry of the study area
300000
600000
5700000
5400000
0 25 50 km
Optionally choose masks to be applied
Aspects Mask
Surface Water Mask
Forest Loss Masked out
Land Masked
2
Choose collections of your interest
Add Sentinel-1
870000
880000
4390000
Add Sentinel-2
660000
690000
4350000
PlotToSat returns a Spectral-Temporal Signature for each added collection. Data are exported into multiple files.
3
Results of Collection 1
.csv
Results of Collection 2
.csv
Example of Spectral-Temporal Signature
B3
B5
B7
B8A
B11
Band
Mar
May
Jul
Sep
Nov
Month
500
1000
1500
2000
2500
3000
3500
4
Download folder containing exported .csv files from Google Drive and merge them using provided scripts
Output: a single .csv file containing field data and extracted spectral-temporal signatures from the selected collections
Tuning parameters in case Errors occur

index	CX	CY	year
0	290926.7	4448817	2017
1	292002.7	4448824	2017
2	286925.8	4447683	2017
3	289845.5	4446749	2017
4	288876	4445730	2017
5	292879.1	4445809	2017
6	295881.3	4445807	2017
7	289932.3	4441763	2017
8	285901.6	4440807	2017
9	286923.5	4440811	2017
10	286873.9	4439839	2017
11	288860.3	4439776	2017
12	289919	4439793	2017

CX	CY	...	10_VHAsc	...	10_B1
290926.7	4448817	...	-17.5825	...	192.0841
292002.7	4448824	...	-17.2322	...	99.82271
286925.8	4447683	...	-17.3208	...	239.8522
289845.5	4446749	...	-16.3866	...	181.7276
288876	4445730	...	-15.3504	...	124.4788
292879.1	4445809	...	-16.3229	...	73.14704
295881.3	4445807	...	-14.2807	...	82.48906
289932.3	4441763	...	-16.3634	...	78.23901
285901.6	4440807	...	-15.7848	...	169.2779
286923.5	4440811	...	-16.0017	...	261.4596
286873.9	4439839	...	-16.2548	...	336.8257
288860.3	4439776	...	-16.0327	...	193.3257
289919	4439793	...	-18.4034	...	321.6212
293911.6	4438760	...	-15.0594	...	205.8156

- "Maximum recursion depth exceeded in comparison
- "Function() too deeply nested"
- "Computation timed out

Figure 1: User's workflow of the system

The system is modular, consisting of multiple classes, yet the user's interaction is primarily with the PlotToSat class. Each class (e.g., *Sentinel-1* class) can function autonomously for other applications and in the Github repository (<`https://github.com/Art-n-MathS/PlotToSat`>) there is a test-case for each class.

The guide explains the User Interface of PlotToSat. For back-end information and image pre-processing steps, please refer to the associated paper.

2 Instructions: How to extract time-series at plot locations

Two test cases are provided for using the *PlotToSat* class. Test case 1 (Section 2.1) contains the minimal information that needs to be provided, while test case 2 (Section 2.3) includes the optional commands. The data exported on Google Drive are often divided into multiple subgroups. Section 2.4 explains how to merge the files exported in Google Drive.

2.1 Test Case 1: compulsory commands

Listing 1 provides the code for test case 1, demonstrating the simplicity of PlotToSat. The code is broken down and explained in Sections 2.1.1, 2.2 and 2.2.1.

```
# import all the necessary libraries
%run PlotToSat.ipynb

# Define study region
polygon = ee.Geometry.Polygon(
[[[-6.7820312500000135, 37.744682241748094],
[-1.1570312500000135, 37.32651387565316],
[0.1613281249999865, 42.25294129803316],
[-7.3093750000000135, 43.349172765639516]]])

# Create a dictionary that holds the relevant plot information
fieldData = {
    "csvfilename"          : "./samplePlots.csv",
    "proj"                 : "EPSG:3042",
    "radius"               : 30,
16          "xcol"                  : "CX",
17          "ycol"                  : "CY",
18          "outPlotFileWithIDs"    : r"plotsWithIDs\SpainIDs_1.csv"
19      }
20
21      # Specify the year of interest
22      year = 2019
23
24      # Create an instance of the Manager
25      myPlotToSat = PlotToSat(polygon,fieldData,year)
26
27      # Add Earth Observation Collections of interest
28      myPlotToSat.addCollection("sentinel-1", True)
29      myPlotToSat.addCollection("sentinel-2", 50  )
30
31      #Definition and exportation of outputs
32      myPlotToSat.exportFeatures("folderSpain1", "outfeaturevectors")
```

Listing 1: This is the "PlotToSat_test1.ipynb" file, which contains a complete example code for extracting time-series EO data at plot locations using PlotToSat.

2.1.1 Definition of input parameters and creation of a PlotToSat instance

To begin, generate a new IPython Notebook (.ipynb file) and **run "PlotToSat.ipynb"** to **import all the necessary libraries**. The new IPython Notebook should be in the same directory as the IPython Notebooks of PlotToSat.

```
1       %run PlotToSat.ipynb
```

Users interact with *PlotToSat* class. To establish an instance of the *PlotToSat* class (Line 25 in Listing 1), three inputs are required:

1. A polygon that defines the study area
2. A dictionary that contains all the relevant information about the field data
3. The specific year for exporting the spectral-temporal signatures

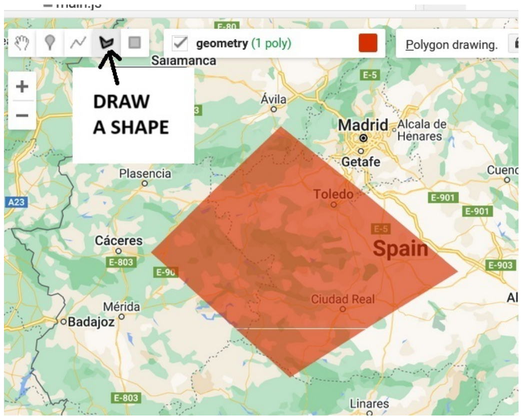
A polygon is required for defining the study region. Here two examples are given. The first example retrieves a database containing all countries and sets Spain as the study region:

```
countries = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017')
polygon = countries.filter(ee.Filter.eq('country_na', 'Spain'))
```

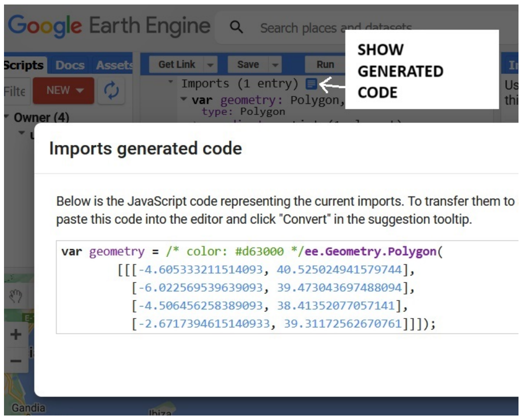
The second example employs a series of coordinates to define a polygon:

```
polygon = ee.Geometry.Polygon(
          [[[-4.605333211514093, 40.525024941579744],
          [-6.022569539639093, 39.473043697488094],
          [-4.506456258389093, 38.41352077057141],
          [-2.6717394615140933, 39.31172562670761]]])
```

The polygon of the 2nd example was generated using the "*Draw Shape*" feature in the Graphical User Interface (GUI) of GEE. The drawn polygon is saved in the Imports section, where the corresponding JavaScript code is provided (Figure 2). To translate the JavaScript code into Python API code, remove the 'var' declaration, the code enclosed between "/*" and "*/" (inclusive of "/*" and "*/"), and the semicolon ";" at the end.



(a) How to draw a polygon in GEE



```
Imports (1 entry)
var geometry: Polygon,
  type: Polygon
```

Below is the JavaScript code representing the current imports. To transfer them to paste this code into the editor and click "Convert" in the suggestion tooltip.

```
var geometry = /* color: #d63000 */ee.Geometry.Polygon(
        [[[-4.605333211514093, 40.525024941579744],
          [-6.022569539639093, 39.473043697488094],
          [-4.506456258389093, 38.41352077057141],
          [-2.6717394615140933, 39.31172562670761]]]);
```

(b) How to find the JavaScript code of polygon

Figure 2: Drawing a polygon and finding its coordinates to use them in PlotToSat.

The plot information are stored in a dictionary, which is a data structure that stores multiple data in pairs of key-value. The key "*csvfilename*" is associated with the file name and directory of the CSV file containing the plot data. The key "*proj*" refers to the geographic projection of the plot data. The key "*radius*" determines the radius of the plots; the unit of measurement is in meters. The keys "*xcol*" and "*ycol*" are associated with the names of the columns containing the x and y coordinates of the plots' centres. The key "*outPlotFileWithIDs*" is associated with the name and directory of a file to be exported. This exported file contains the plot information but it also attaches identifiers to them. This file with the identifiers is used for merging the time-series files into one CSV file, along with the plot information. At each run, please provide a different file name for key "outPlotFileWithIDs"; otherwise, PlotToSat will overwrite an existing file with the same name, and merging time-series with plot information will not be possible. All the keys of the dictionary are mandatory. Here is an example of defining them:

```
    fieldData = {
           "csvfilename"          : "./samplePlots.csv",
           "proj"                 : "EPSG:3042",
           "radius"               : 30,
           "xcol"                 : "CX",
           "ycol"                 : "CY",
           "outPlotFileWithIDs"   : r"plotsWithIDs\SpainIDs_1.csv"
    }
```

Specify the year of interest. PlotToSAT processes data for a single year in each run. The year of interest must be defined and imported into the constructor of the *PlotToSat* class. In the code snippet below, we define a variable that contains a potential year of interest.

```
    year = 2017
```

Once the three inputs are defined, you can **create an instance of PlotToSat** as follows:

```
    myPlotToSat = PlotToSat(polygon,fieldData,year)
```

2.2 Adding Earth Observation collections

Once the Manager is constructed, the user can add the collections of their interest. If the available collections are not known, the following command can be used to print the available collections, their call labels, and the GEE collections fetched:

```
    myPlotToSat.printAvailableCollections()
```

This should yield the following outcome since the system currently supports two collections:

```
  There are  2 collections available within the system:
  label                    collection
  0  sentinel-1             COPERNICUS/S1_GRD
  1  sentinel-2  COPERNICUS/S2_SR_HARMONIZED
```

The users select collections using the function *addCollection(<labelOfCollection>*, <parameter>). The system currently supports the Sentinel-1 and Sentinel-2 collections, with the respective labels *"sentinel-1"* and *"sentinel-2"*. The *<parameter>* argument serves a different purpose for each collection. For Sentinel-2, it's an integer defining the upper limit threshold of cloud coverage. Images exceeding this threshold are discarded. It's worth mentioning that additional cloud and shadow masking is applied to all images. For Sentinel-1, the input parameter is a Boolean value (True or False), which determines whether the aspect maps will be applied; if the elevation gradient is high, shaded areas usually appear on certain slopes. These shaded areas can be masked out using the aspect map. Detailed information about the available collections and pre-processing steps are provided at the relevant paper [1].

Users have the choice to use either one or both collections. Here is an example of how to add both collections to the *PlotToSat* class with aspect filters enabled for Sentinel-1 and an upper limit threshold of 50% cloud coverage for Sentinel-2:

```
    myPlotToSat.addCollection("sentinel-1", True)
    myPlotToSat.addCollection("sentinel-2", 50  )
```

2.2.1 Definition and exportation of outputs

Finally, we execute the following command to **fetch the data, interpret them, and export the time-series signatures as feature vectors.** The command "*myPlotToSat.exportFeatures(<a>,<b>)*" requires two inputs: (a) the folder name where the data will be exported on the user's Google Drive, and (b) the starting name for the exported feature vectors. If the specified folder does not exist, GEE will create it. It's important to note that GEE is unable to generate subfolders. Additionally, due to parallel processing, sometimes GEE creates two folders with the same name and shares the exported files between them.

```
    myPlotToSat.exportFeatures("gdrivefolder", "outfeaturevectors")
```

By executing the above command, a series of CSV files will be exported within the "*gdrivefolder*" in the user's Google Drive. By default PlotToSat exports time-series for 400 plots in each exported CSV file; changing this values is an optional command included in Test case 2 (Section 2.3.2). For instructions on merging the series of CSV files, please refer to Section 2.4.

2.3 Test Case 2: Optional commands

Listing 2 provides the code for test case 2, which includes the optional commands of PlotToSat. The optional commands added from test case 1 to test case 2 are explained in Sections 2.3.1 and 2.3.2.

```
    # Include these lines and comment out "ee.Authenticate()" after the
    # first use to avoid authentication at each run.
    import ee
    ee.Authenticate()
    ee.Initialize()
    import sys

    # import all the necessary libraries
    %run PlotToSat.ipynb

    # By default recursion is 1000. By increasing it PloToSat can handle
    # more plot data at once but you are doing it at your own risk as
    # raising it too much could cause the system to crash
    sys.setrecursionlimit(10000)

    # Definition of Study area
    countries = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017')
    polygon = countries.filter(ee.Filter.eq('country_na', 'Spain'))

    # Create a dictionary that holds the relevant plot information
    fieldData = {
        "csvfilename"   : "./samplePlots.csv",
        "proj"          :"EPSG:3042",
        "radius"        :25,
        "xcol"          :"CX",
        "ycol"          :"CY",
        "outPlotFileWithIDs"  : r"plotsWithIDs\SpainIDs_2.csv"
    }

    # Specify the year of interest
    year = 2020

    # Create an instance of the PlotToSat class
    myPlotToSat = PlotToSat(polygon,fieldData,year)

    # With this command you can see all the supported collections and the
    # associated labels needed for adding them into the PlotToSat instance
    myPlotToSat.printAvailableCollections()

    # Example of defining the optional masks
    masks = {
        "gsw": 30,
        "lmask": 30,
        "forestMask": {
            "buffer":30,
            "startDate":'2000-01-01',
            "endDate":'2019-12-31'
            }
        }
    myPlotToSat.setMasks(masks)

    # GEE limits processing of data, so PloToSat divides plots data into
    # groups. The default size of a group is 400 plots. A bigger number
    # produces less files to be merged and uses less GEE requests. But if
    # it is too big GEE returns an ERROR. So some testing is required here
    # to tune the sampling size.
    myPlotToSat.setSampling(300)

    # Adding Earth Observation Collections
    myPlotToSat.addCollection("sentinel-1", False)
    myPlotToSat.addCollection("sentinel-2", 50  )

    # Definition and exportation of outputs
    myPlotToSat.exportFeatures("folderSpain2", "r25_2020")

    # Command for re-running a subgroup of plots in case of time-out Errors
    myPlotToSat.exprtFeaturesMinMax("folderSpain2","r25_2020",300,600)
```

Listing 2: This is the "PlotToSat_test1.ipynb" file, which contains a complete example code for extracting time-series EO data at plot locations using PlotToSat.

2.3.1 Available optional masks

Masks can be used to reduce noise within the EO collections. The chosen optional masks are applied to all the EO collections added to the instance of *PlotToSat* (Section 2.2), and they remain constant across different dates. Before masking the EO data, a user-defined buffer is applied to each mask individually, with each mask having its own unique buffer. The related technical information about the available masks with examples are provided in the associated paper [1].

The user creates a dictionary containing the masks of interests. The dictionary may contain from zero to multiple masks. The masks' dictionary is added to an already created instance of the class *PlotToSat* (e.g., *myPlotToSat* instance, Section 2.1.1) using this command *myPlotToSat.setMasks(<masksDictionary>)*.

Table 1 provides a summary of the available masks, including their associated labels and the input parameters required for each mask to be added to the masks' dictionary. For each mask of interest you need (1) its corresponding label and (2) a buffer value. To define the forest loss mask though, a dictionary is required as input; the dictionary should contain the start and end date of the forest loss (note: only years are used), along with the buffer.

The buffer value defines how much buffer will be added around the edges of the area to be removed. Regarding the Descending and the Ascending Aspects the buffer value must always be zero as

Table 1: Available masks that can optionally be loaded to PlotToSat and applied.

Mask	Label	Input Parameters
Ground surface water	"gsw"	<buffer>
Land mask	"lmask"	<buffer>
Forest loss mask	"forestMask"	{ "buffer": <buffer> "startDate":<startDate> "endDate":<endDate>}
Descending Aspects (22.5-157.5)	"aspectDes"	0
Ascending Aspects (202.5-337.5)	"aspectAsc"	0

a median filter has already been applied and the segments of the aspects are small so adding buffers could results into very little to no data retained. Additionally, Descending and Ascending Masks should never be applied simultaneously, as they have no overlap. If the user wants to apply Ascending masks to the ascending Sentinel-1 data and Descending masks to the descending Sentinel-1 data then the option provided in Section 2.2 should be used while adding the Sentinel-1 collection to the Manager (i.e., <*myPlotToSat.addCollection("sentinel-1", True)*>, where "*True*" implies using the aspects maps).

Here is an example of how the user can define a surface mask with 30 meter buffer, a land surface mask with 30 meter buffer and a forest loss mask from 2000 till 2017 with a 30 meter buffer. The masks dictionary, named "*masks*" is then added to the instance of the instance of the *PlotToSat* class, named "*myPlotToSat*".

```
    masks = {
          "gsw": 30,
          "lmask": 30,
          "forestMask": {
                 "buffer":30,
                 "startDate":'2000-01-01',
                 "endDate":'2017-12-31'
                  }
            }
    myPlotToSat.setMasks(masks)
```

2.3.2 Dealing with potential errors

There are three known errors that you may encounter while running PlotToSat:

1. "Maximum recursion depth exceeded in comparison"
2. "Function() too deeply nested"
3. "Computation timed out"

The first two errors are closely associated, but the first will appear when executing the iPython script, while the second will arise when executing a GEE request on the Tasks tab. Recursion is a programming technique where a function calls itself to solve a problem. The "Maximum recursion depth exceeded" error happens when a recursive function calls itself excessively, surpassing the allocated memory for function calls within the program's call stack. This error is often addressed by rewriting the code in an iterative way. However, it's important to note that GEE is a functional programming language, intended to work with recursive functions. The primary solution implemented in PlotToSat is to divide the imported plots into subgroups and process those subgroups iteratively. The results of each subgroup though are export into a file, resulting into exporting multiple files but a script is provided for merging those files (Section 2.4).

There are two parameters the user may tune to reduce the possibility of Errors 1 and 2: (1) Increasing the maximum recursion depth of the system enables the system to process more data within a single subgroup, but it may result in slowdowns or even crashes. If you choose to use this option, it's important to understand its associated risks. The code for increasing the maximum depth as follows:

```
    import sys
    sys.setrecursionlimit(n)
```

where 'n' represents the new recursion depth/limit. The default value is 1000, which is retrievable using the function "*sys.getrecursionlimit()*". It further worth noting that this only changes the recursion parameters of a computer and will not resolved Error 2.

(2) By default, PlotToSat divides the plots into subgroups to mitigate Errors 1 and 2. The default number of plots per subgroup is 400. The user may define the number of plots per subgroup to either increase or decrease the number of GEE requests. Please note that if we increase the sampling number, the number of GEE requests and exported files decreases. Here is an example of defining the number of plots per subgroup:

```
    myPlotToSat.setSampling(n)
```

where "n" represents the number of plots that will be included in each subgroup. If the plots are spread across many EO tiles, the user may need to reduce the sampling number, while if the plots are closer together, a higher number of plots per subgroup should work. We recommend copying a subset of the plot data into a CSV file and conducting tests, starting with 1000 plots per subset and adjusting this number up or down to assess system and GEE capacity until optimal tuning is achieved. We advise against running this test on the entire dataset to avoid unnecessary consumption of cloud resources and GEE requests.

Regarding the final potential error, in [1] there were 2 cases of failed requests out of 174. If the request timeouts, the user can identify the range of plot data not processed (stated on the name of the failed request) and use the command "*myPlotToSat.exportFeaturesMinMax(<folder>, <startOfName>, <min>, <max>)*" to reprocess a subset of the plot network. The command "*myPlotToSat.exportFeaturesMinMax(<folder>, <startOfName>, <min>, <max>)*" requires four inputs: (a) the folder name where the data will be exported on the user's Google Drive, (b) the starting name for the exported feature vectors, (c) the index where processing of plots will stop. For example, if min=300 and max=600, plots at indices 300 through 599 will be processed. Index 300 is included, while index 600 is excluded. Please note that this command does not create subgroups. Therefore, if multiple commands fail you need to divide the data into subgroups manually and run the command multiple times. An example is given below:

```
    myPlotToSat.exprtFeaturesMinMax("folderSpain2", "r25_2020",300,600)
    myPlotToSat.exprtFeaturesMinMax("folderSpain2", "r25_2020",600,900)
```

Please do not rerun "*myPlotToSat.exportFeatures(<folder>, <startOfName>)*" as it will reprocess the EO data for all plots instead of just the failed requests, resulting in unnecessary consumption of computing resources, which is not energy efficient and consequently harmful to the environment.

2.4 Downloading and merging exported files

The "*MergingLib*" module provides the "*mergeAll(<gdriveFolderDir>, <fieldDataWithIdentifiers>)*" command, which merges the multiple exported CSV files from GEE. Initially, the user needs to download and extract the folder where the CSV files are saved on Google Drive (the folder's name is defined in PlotToSat, as shown in Section 2.2.1). Please note that sometimes GEE may create two folders with the same name and distribute the generated CSV files across both folders, due to GEE's parallel data processing. In such cases, download both folders and merge their contents before proceeding with the script for file merging. Then, create a new .ipynb file in the same directory as PlotToSat's .ipynb files and import the necessary libraries (as shown in line 1 of Listing 3). To execute the "*mergeAll(<gdriveFolderDir>, <fieldDataWithIdentifiers>)*" command (as shown in line 5 of Listing 3), you need to provide two input parameters:

1. The name and path directory of the local folder downloaded from your Google Drive, where the CSV files exported by PlotToSat and GEE are stored.

2. The value assigned to the key "*outPlotFileWithIDs*" when generating the field-related dictionary (Section 2.1.1). For example, in Listing 1, the value is "*plotsWithIDs/SpainIDs_1.csv*," and in Listing 2, it's "*plotsWithIDs/SpainIDs_2.csv*." These are file paths pointing to CSV files, each containing the related plot information with attached identifiers. As mentioned earlier, a different file path and CSV file name must be provided for each run of PlotToSat; otherwise, PlotToSat will override the previously created files.

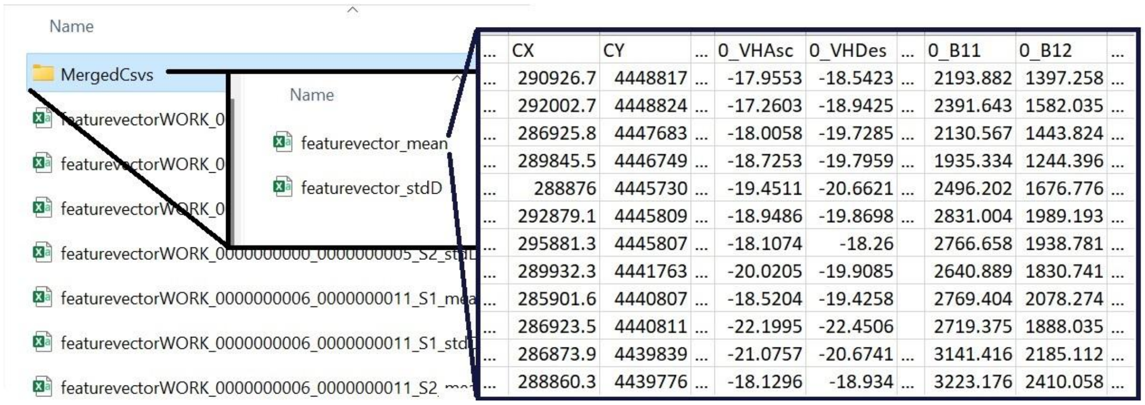
A full example code for merging the multiple exported CSV files is given here:

```
1  %run MergingLib.ipynb
2
3  gdriveFolderDir = r"C:\Documents\folderSpain2"
4  fieldDataWithIdentifiers = r"plotsWithIDs\SpainIDs_1.csv"
5  mergeAll(gdriveFolderDir,fieldDataWithIdentifiers)
```

Listing 3: An example script of how to merge the multiple exported files on GEE

3 Outputs: What do you get and what does it mean

Once the merge script is executed, the folder "MergedCsvs" is created inside the local folder downloaded from your Google Drive. The folder "MergedCsvs" contains two files: one for the mean and one for the standard deviation values of the pixels lying within the plots' circumferences (Figure 3). Please note that the mean and standard deviation values provided represent the statistics of the pixels within each plot's circumference. Furthermore, the pixel-wise mean of all images acquired during each month is calculated prior to calculating the exported statistics.


Name
MergedCsvs
featurevectorWORK_0
featurevectorWORK_0
featurevectorWORK_0
featurevectorWORK_0000000000_0000000005_S2_stdL
featurevectorWORK_0000000006_0000000011_S1_mea
featurevectorWORK_0000000006_0000000011_S1_std
featurevectorWORK_0000000006_0000000011_S2_mea
Name
featurevector_mean
featurevector_stdD

...	CX	CY	...	0_VHAsc	0_VHDes	...	0_B11	0_B12	...
...	290926.7	4448817	...	-17.9553	-18.5423	...	2193.882	1397.258	...
...	292002.7	4448824	...	-17.2603	-18.9425	...	2391.643	1582.035	...
...	286925.8	4447683	...	-18.0058	-19.7285	...	2130.567	1443.824	...
...	289845.5	4446749	...	-18.7253	-19.7959	...	1935.334	1244.396	...
...	288876	4445730	...	-19.4511	-20.6621	...	2496.202	1676.776	...
...	292879.1	4445809	...	-18.9486	-19.8698	...	2831.004	1989.193	...
...	295881.3	4445807	...	-18.1074	-18.26		2766.658	1938.781	...
...	289932.3	4441763	...	-20.0205	-19.9085		2640.889	1830.741	...
...	285901.6	4440807	...	-18.5204	-19.4258		2769.404	2078.274	...
...	286923.5	4440811	...	-22.1995	-22.4506		2719.375	1888.035	...
...	286873.9	4439839	...	-21.0757	-20.6741	...	3141.416	2185.112	...
...	288860.3	4439776	...	-18.1296	-18.934	...	3223.176	2410.058	...

Figure 3: An example of how the data are exported. Once the script for merging the data is run, the folder "MergedCsvs" is created.

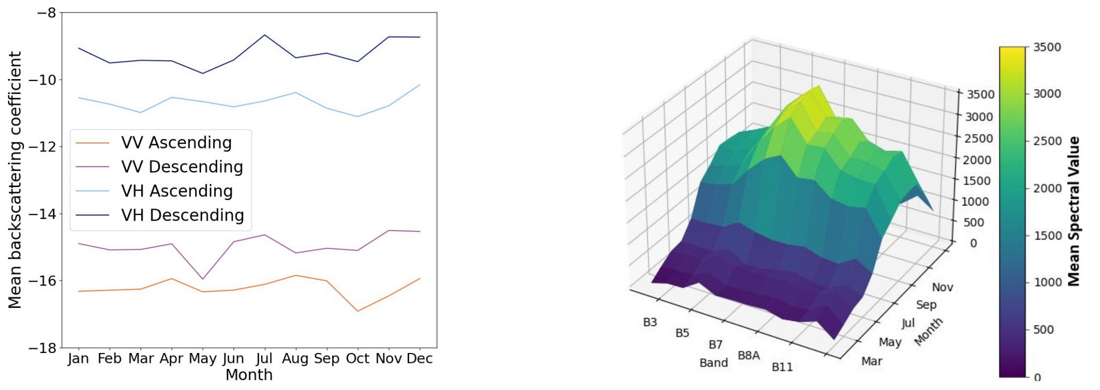
Each row in the merged CSV files contains plot information, including both plot and EO time-series data. The plot data are copied from the imported CSV file listing the plots of a plot network. Regarding the EO time-series data, twelve values are provided for each band, corresponding to each calendar month. Each column contains statistics for a specific band over the course of one month. A unique prefix is added to the existing band names, representing the months from 0 to 11, which correspond to January through December respectively. Each column name has the form A_B. Part A defines the temporal information of the time-series and part B the band information of the time-series. For instance, "0_B11" designates band B11 of Sentinel-2 for the month of January. Please note that the order of the columns is random!

Regarding Sentinel-2 the bands {B1, B2, B3, B4, B6, B7, B8, B8A, B9, B11, B12} correspond to {Aerosols, Blue, Green, Red, Red Edge 1, Red Edge 2, Red Edge 3, NIR, Red Edge 4, Water vapor, SWIR 1, SWIR 2}.

Regarding Sentinel-1, all bands {VHAsc, VHDes, VVAsc, VVDes} are collected in the microwave part of the spectrum. Polarisation defines the orientation of the electromagnetic waves. VV polarisation means that both transmitted and received signals have a vertical orientation. VH means that the transmitted signal has a vertical orientation and the received signal has a horizontal orientation. Asc and Des refer to the movement of the satellite; in an Ascending (Asc) orbit, the satellite is moving from south to north, while in a Descending (Des) orbit, the satellite is moving from north to south.

Please note that sometimes gaps exist within the exported merged files. If the entire EO time-series information for a plot is missing, then it is likely that the corresponding plot was located in a masked out area. If parts of the EO time-series are missing, it could be due to defects in data acquisition or due to cloud masking.

Figure 4 shows examples the Sentinel-1 and Sentinel-2 time-series of a single plot. There are four time-series per plot for Sentinel-1, corresponding to two polarisations (VV, VH) and two orbit direction (Ascending and Descending), resulting in four distinct options when combined. A Sentinel-2 time-series forms a spectral-temporal signature that captures both the spectral and temporal characteristics of a plot.



(a) Sentinel-1 (b) Sentinel-2

Figure 4: Examples of Sentinel-1 time-series and Sentinel-2 spectral-temporal signatures at a plot.

4 Acknowledgments

M. M., S. W. D. G. and E. R. L. were funded by a UKRI Future Leaders Fellowship (MR/T019832/1) awarded to E. R. L. and the University of Cambridge.

P. R-B. and J. A. acknowledge funding from the CLIMB-FOREST Horizon Europe Project (No 101059888) that was funded by the European Union

References

[1] M. Miltiadou, S. Grieve, P. Ruiz-Benito, J. Astigarraga, V. Cruz-Alonso, J. Tijerín Triviño, and E. Lines, “Plottosat: A tool for generating time-series signatures from sentinel-1 and sentinel-2 at each plot within a plot network for machine learning applications,” *Computers and Geosciences*, 2024.