School of Mathematical and Physical Sciences

Python Based Data Analysis Tools for Raman LiDAR Models

Visible Infrared Imaging Radiometer Suite (VIIRS) onboard the Suomi National Polar-orbiting Partnership [Credit: NASA]

Adem Ozer

Supervisor:

Prof. Helen Pask

School of Mathematical and Physical Sciences,
Macquarie University, North Ryde NSW 2109, Australia

Date: January, 2024

Contents

Satellite Spectrometer Data

1.1 Data Processing Levels

NASA's Earth Observing System Data and Information System (EOSDIS) data products are processed at various levels ranging from Level 0 to Level 4. Level 0 products are raw data at full instrument resolution. At higher levels, the data are converted into more useful parameters and formats. All EOS instruments must have Level 1 Standard Data Products (SDPs); most have SDPs at Level 2 and Level 3; and many have Level 4 SDPs. Some EOS Interdisciplinary Science Investigations also have generated Level 4 SDPs.

Data Level	Description
Level 0	Reconstructed, unprocessed instrument and payload data at full resolution, with any and all communications artifacts (e.g., synchronization frames communications headers, duplicate data) removed. (In most cases, NASA's EOS Data and Operations System [EDOS] provides these data to the DAACs as production data sets for processing by the Science Data Processing Segment [SDPS] or by one of the SIPS to produce higher-level products.)
Level 1A	Level 1 A(L1 A) data are reconstructed, unprocessed instrument data at full resolution, time-referenced, and annotated with ancillary information, including radiometric and geometric calibration coefficients and georeferencing parameters (e.g., platform ephemeris) computed and appended but not applied to L0 data.
Level 1B	L1B data are L1A data that have been processed to sensor units (not all instruments have L1B source data).
Level 1C	L1C data are L1B data that include new variables to describe the spectra. These variables allow the user to identify which L1C channels have been copied directly from the L1B and which have been synthesized from L1B and why.
Level 2	Derived geophysical variables at the same resolution and location as L1 source data.
Level 2A	L2A data contains information derived from the geolocated sensor data, such as ground elevation, highest and lowest surface return elevations, energy quantile heights ("relative height" metrics), and other waveform-derived metrics describing the intercepted surface.
Level 2B	L2B data are L2A data that have been processed to sensor units (not all instruments will have a L2B equivalent).
Level 3	Variables mapped on uniform space-time grid scales, usually with some completeness and consistency.
Level 3A	L3A data are generally periodic summaries (weekly, ten-day, monthly) of L2 products.
Level 4	Model output or results from analyses of lower-level data (e.g., variables derived from multiple measurements).

Table 1: NASA data products level and descriptions [1].

1.2 Algorithms for data processing

A brief outline of the derivation of the relevant data products available from the satellite measurements are outline below. The full list of data product algorithms and their descriptions can be found on the OceanColor website.

1.2.1 Remote Sensing Reflectance (Rrs)

The remote sensing reflectance (R_{rs}) plays a foundational role in deriving all measurements from ocean colour sensors. $R_{rs}(\lambda)$ provides the data used in the input to many derived product including the diffuse attenuation.

The ocean-colour satellite sensors measure the up-welling subsurface reflection from solar radiation. These measurements unsurprisingly are significantly hampered by atmospheric reflections from air molecules and aerosols. Additionally there exists significant surface contributions from whitecaps and sun glint as well as atmospheric absorption and scattering for light exiting the water column. Removing these influences requires accurate modelling of the impact they present to the signal measured by the instrument.

The retrieved water-leaving radiance, $\mathrm{L_w}(\lambda)$, at each sensor wavelength, (λ), are normalised, removing effects of solar orientation and atmospheric attenuation from the downwelling radiation. This produces a value of the normalised water-leaving radiance, $\mathrm{nL_w}(\lambda)$ [2]. We then derive the $\mathrm{R_{rs}}(\lambda)$ by dividing this quantity by the mean extraterrestrial solar irradiance, $\mathrm{F_0}(\lambda)$.

In the algorithm utilised by the NASA OceanColor community, the top of the atmosphere (TOA) radiance is assumed to contain a linear combination of various measurement contributions, which we can write given as [3],

$$L_t(\lambda) = \left[L_r(\lambda) + L_a(\lambda) + t_{d_v}(\lambda)L_f(\lambda) + t_{dv}(\lambda)L_w(\lambda)\right] t_{g_v}(\lambda)t_{g_s}(\lambda)f_p(\lambda), \tag{1.2.1}$$

where we have the contributions from,

$L_r(\lambda)$ = the radiance contribution due to Rayleigh scattering by air molecules

$L_a(\lambda)$ = the contribution due to scattering by aerosols, including multiple scattering interactions with the air molecules

$L_f(\lambda)$ = the contribution from surface whitecaps and foam

$L_w(\lambda)$ = the water-leaving component

$t_{d_v}(\lambda)$ = the transmittance of diffuse radiation through the atmosphere in the viewing path from surface to sensor

$t_{g_v}(\lambda)$ = the transmittance loss due to absorbing gases for all up-welling radiation travelling along the sensor view path

$t_{g_s}(\lambda)$ = the transmittance of diffuse radiation through the atmosphere in the viewing path from Sun to surface

$f_p(\lambda)$ = is an adjustment for effects of polarisation.

As we stated above we are interest in the water leaving radiance $L_w(\lambda)$ and thus (with very limited detail provided!), the contributions of each component are simply subtracted from the measurement, leaving us with $L_w(\lambda)$, which we then compute the remote sensing reflectance by taking,

$$R_{rs}(\lambda) = \left(L_w(\lambda)/\left(F_0 f_s \cos\left(\theta_s\right) t_{d_s}\right) f_b(\lambda) f_\lambda\right) \tag{1.2.2}$$

where we have,

F_0 = extraterrestrial solar irradiance

f_s = adjustment of F_0 for variation in Earth-Sun distance

t_{d_s} = the transmittance of diffuse radiation through the atmosphere in the viewing path from Sun to surface

$f_b(\lambda)$ = bidirectional reflectance correction

f_λ = correction for out-of-band response

This algorithm therefore derives the spectral radiance upwelling from beneath the ocean surface, normalised by the downwelling solar irradiance [4]. Expressed as spectral "remote sensing" reflectance, $R_{rs}(\lambda)$ at each sensor wavelength, λ, in the visible domain with units of sr^{-1}.

1.2.2 Diffuse Attenuation Coefficient (Kd)

The diffuse attenuation coefficient, provides a measurement of the turbidity of the ocean in a particular region, in particular how light in the blue to green visible spectrum is scattered by the water. NASA provides a data product of the diffuse attenuation coefficient at 490 nm ($K_d(490)$). The details of the data product are as follows.
When computed through the measurements taken via satellite spectroradiometers, $K_d(490)$ can be derived from measurements of blue-to-green band ratios of remote sensing reflectances (R_{rs}) [5]. The algorithm for this derived data product is a fourth-order polynomial of the ratio of R_{rs} given by,

$$\log_{10}\left(K_{bio}\left(490\right)\right) = a_0 + \sum_{i=1}^{4} a_i \left(\log_{10}\left(\frac{R_{rs}\lambda_{blue}}{R_{rs}\lambda_{green}}\right)\right)^i$$
$$K_d(490) = K_{bio}\left(490\right) + 0.0166. \tag{1.2.3}$$

The fitting parameters for each sensor and satellite are given in the table below. These parameters

	sensor	blue	green	a_0	a_1	a_2	a_3	a_4
KD2S	SeaWiFS	490	555	-0.8515	-1.8263	1.8714	-2.4414	-1.0690
KD2M	MODIS	488	547	-0.8813	-2.0584	2.5878	-3.4885	-1.5061
KD2E	MERIS	490	560	-0.8641	-1.6549	2.0112	-2.5174	-1.1035
KD2V	VIIRS	490	550	-0.8730	-1.8912	1.8021	-2.3865	-1.0453
KD2O	OCTS	490	565	-0.8878	-1.5135	2.1459	-2.4943	-1.1043
KD2C	CZCS	443	520	-1.1358	-2.1146	1.6474	-1.1428	-0.6190
KD2L	OLI/Landsat 8	482	561	-0.9054	-1.5245	2.2392	-2.4777	-1.1099

were computed using the NOMAD: NASA bio-Optical Marine Algorithm Dataset and have been verified by comparisons to alternative models [6, 7]. Further details can be found on the OceanColor algorithm web-page.

1.3 Accessing Data

Data collected by the various satellites can be accessed in two ways, firstly the home page of each satellite may directly link to the data and the associated products for the relevant use case, see for example the home page for the Suomi NPP VIIRS in figure 1. The home page will often contain the relevant links to the data products and also the raw data. It has also been conveniently categorised into the specified use case for each data product.
Most relevant to this project, we follow the links to the **Oceancolor** data and find that we are able to directly access the relevant data at all four of the data processing levels (See table 1). The Oceancolor categorised data levels as seen in figure 2

Alternatively we may directly access data from the NASA Oceancolor website. This data has been collected and grouped by data levels, and can be accessed for specific mis-

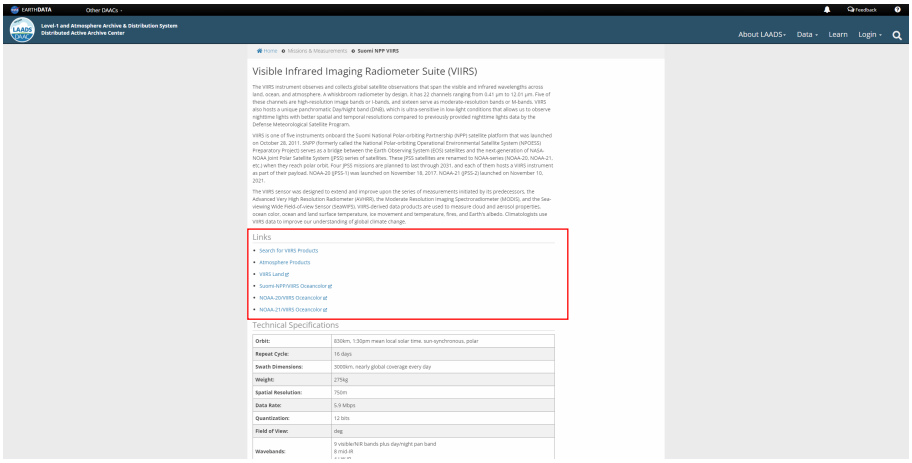


Figure 1: Links to data products for VIIRS available from: https://ladsweb.modaps.eosdis.nasa.gov/missions-and-measurements/viirs/

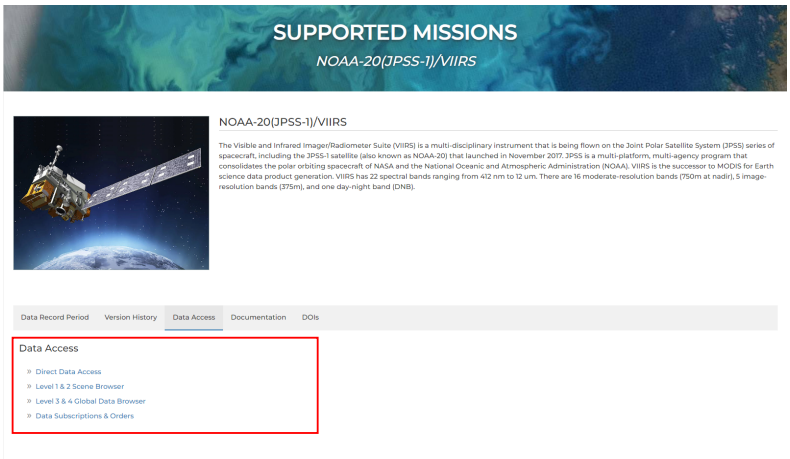


Figure 2: Data access categorised by the various data processing levels for NOAA-20 VIIRS available at: https://oceancolor.gsfc.nasa.gov/about/missions/noaa20/

sions or periods of time. The full range of data access categorisation can be found at https://oceancolor.gsfc.nasa.gov/data/find-data/. This methods provides the greatest ease of access, particularly if a specific data product is required from a specific mission.

1.3.1 NASA Oceancolor Level 3 & 4 Browser

The Oceancolor level 3 and 4 data access browser serves as the most straight forward way to access data from a wide variety of different missions or instruments. The browser homepage can be seen in figure 3 and can be accessed from https://oceancolor.gsfc.nasa.gov/l3/

Data from the Oceancolor level 3 & 4 browsers requires a free Earthdata account to be created (create account here). Once this account has been created you may begin to order and download data. The browser provides many data filtering parameters and can be used to extract data relevant to only a specific area of interest from a specific instrument and period of time.
For example, we may choose to collect the diffuse attenuation coefficient data captured over the

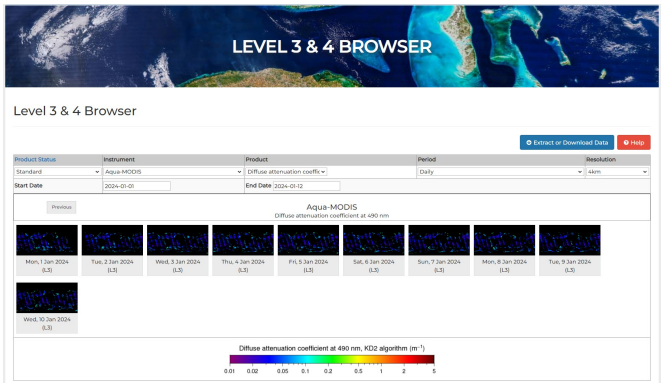
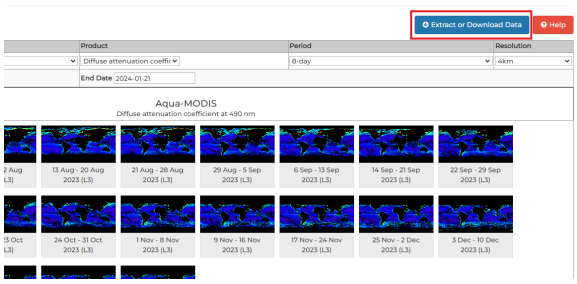


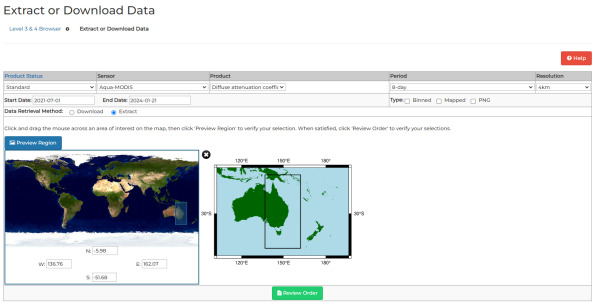
Figure 3: Oceancolor level 3 & 4 data browser. The full range of data products can be accessed from this browser for the full range of data dates collected. The various other options can all be selected from the drop down menu. The browser can be accessed via: https://oceancolor.gsfc.nasa.gov/l3/.

eastern seaboard of Australia. To do this we follow the link to download or extract data, in the top right hand corner of the browser, seen in figure 4a.

After navigating to the data extraction page, and selecting 'Extract' from the data retrieval selection options - one needs to simply select the total region of interest by dragging the region selector seen in figure 4b.



(a) Button to navigator to the data extraction page



(b) Select region of interest to extract local data.

Figure 4: Extracting data from the oceancolor level 3 & 4 data browser over a specific region of interest.

NOTE: Data extracted must be selected as type: '**mapped**' in order to work correctly with the Python function library.

Once the data has been selected, extracted and reviewed, Oceancolor requires the requested data order to be confirmed before processing the request. Once this has been completed a compressed file will be available to download. This file can be extracted using any conventional extraction software or alternatively on windows one is able to extract this file using command prompt, by firstly navigating to the file directory using the command `cd <directory>` where `<directory>` is the location of the .tar file, next run the tar the command `tar -xf <file-name>.tar`. The .tar file will now be extracted to a folder in the same location and is ready to be analysed.

Data Analysis Tools

Once the requested data has been downloaded locally one may begin utilising the python notebook and function library to plot and analyse the data. The function library and full list of dependencies can be found on the github

NOTE: The python function library requires the installation of packages netCDF4 and cartopy. Additionally for animation functions please install imageio and include also imageio-ffmpeg

Packages can be installed using `pip install cartopy` and `pip install netCDF4` however netCDF4 can be slightly troublesome and can alternatively be installed using `conda install -c conda-forge netCDF4`. With these packages installed we may begin using the library. The data plotting function library enables the access and visualisation of the netCDF (.nc) data files that have been downloaded and extracted from the oceancolor browser. We will walk through each of these functions, their inputs and outputs below.

2.1 Setup notebook

Once the required packages have been installed, import the function library, ensuring the function library `netCDF_Lib_v2.py` file is saved in the same directory as the jupyter notebook. Importing the library can be done simply using the command below.

```
from netCDF_Lib_v2 import *
```

Next, define the local path where the folder containing all the `.nc` files is stored, for example:

```
folderpath = r'C:\Users\45418128\OneDrive\LIDAR\Data\requested_files'
```

As a simple first check we can view the files stored in the folder, to do so can call the function `MultiFile_FileNames('path')`. The function will print the full list of files that have been stored in the folder-path specified.

```
MultiFile_FileNames(folderpath)
```

This function will return an array of strings corresponding to the full list of `.nc` files in the folder-path. Typically, and in keeping with the data extraction methods from OceanColor, one would store all the files which corresponds to the data collected by an instrument over and extended period of time, in a single folder.

2.2 Simple data plotting

To plot the data extracted in a single file we need to either copy explicitly an individual file-path for a chosen data file - or alternatively we select a single file from the list given by the function `MultiFile_FileNames('path')`. Choosing the latter methods one would define,

```
File = MultiFile_FileNames(folderpath)[0]
```

Now that we have chosen a single file, we can plot the data using the plotting function `KD490_Plot_Data('filepath')`.

```
KD490_Plot_Data(File)
```

The output of this function gives us a simple visual of the $K_d(490)$ data from the file we have chosen, seen in figure 5.

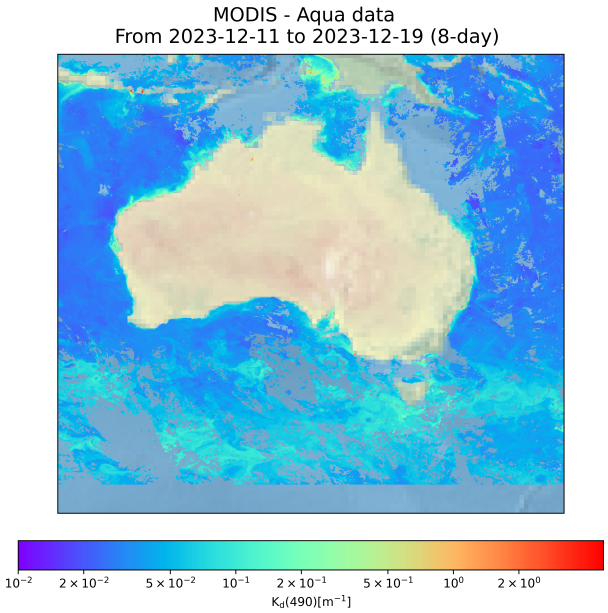


Figure 5: Output figure of the function `KD490_Plot_Data()`.

2.3 Saving plots

To save the output of any plotting function one must simply pass in the keyword argument `save='filename.png'` to the end of the plotting function. For example to save the figure output of the function `KD490_Plot_Data()` as `Example.png`.

```
KD490_Plot_Data(File, save='Example.png')
```

2.4 Plotting a region of interest

We may be interested in plotting data over a specific region of interest or multiple positions of interest, we do so as follows.

Firstly define a list positions using the coordinates in longitude and latitude.

```
PosList = [[148,-18],[150,-20],[153,-22]] # Positions in [Lon, Lat]
```

Then, using the function `CoordsCalcList(Region Size, [Positions])` we are able to generate a list of coordinates bounding a region of interest. For example, if we wish for the region to be 100 x 100 km we could do this by running,

```
PosList = [[148,-18],[150,-20],[153,-22]] # Position in [Lon, Lat]
C_List = CoordsCalcList(100,PosList) # Region to cover [km]
```

We now can print the output of this function to see the generated list of coordinates bounding the region of interest.

```
C_List
```

```
array([[147.55, 148.45, -17.55, -18.45],
       [149.55, 150.45, -19.55, -20.45],
       [152.55, 153.45, -21.55, -22.45]])
```

We see the output has now provided us with a list of coordinates, stored in `C_List`. This list is necessary as it helps in easily defining a region of interest in coordinates which span a certain area size. Further we will use these coordinates for the region specific plotting functions below.
To plot the data in a specified region we do so by running the function `KD490_Plot_Data_Region('filepath', [coords])`. In the example below we call this function and have selected the second set of coordinates given by `C_List`.

```
KD490_Plot_Data_Region (File, C_List[1])
```

The output of this function plots the full data and importantly provides a closer examination of the data in our region of interest, seen in figure 6.

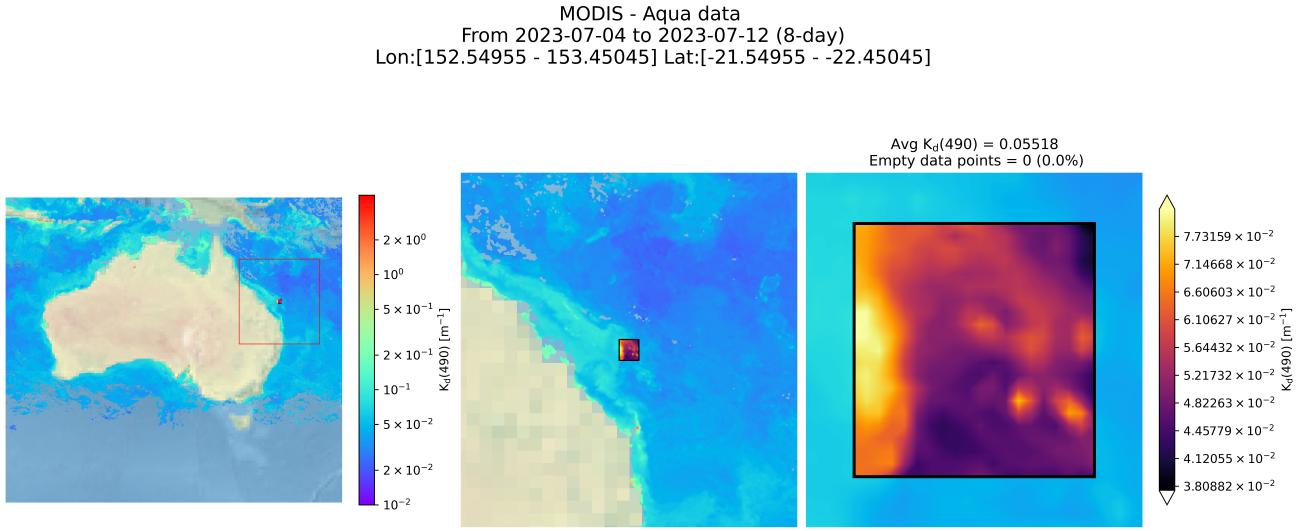


Figure 6: Output figure of the function `KD490_Plot_Data_Region()`.

2.4.1 Data Grid Inspection

Additionally, although the data plotted by the function `KD490_Plot_Data_Region()` provides a simpler plot of the data, one may be interested in the raw data points within the region of interest. To plot the individual data points one needs to simply add the kwarg `grid = True`. For example, to plot the grid in addition to the plot above one would call the function as,

```
KD490_Plot_Data_Region(File, C_List[1], grid=True)
```

After we plot with the additional argument we are able to inspect the explicit grid of data points and further examine the full number of data points used, seen in figure 7.

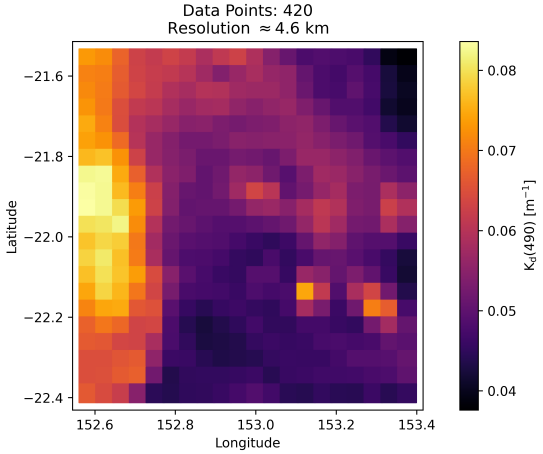


Figure 7: Output figure of the function `KD490_Plot_Data_Region()` where we have specified the argument `grid = True`.

2.5 Coordinates checking

So far we have seen how can produce a list of regions of interest using the function `CoordsCalcList()`, and further we can visually assess the data captured in these regions using the function `KD490_Plot_Data_Region()`. However if we would like to compute the average value of the data in multiple positions of interest, over the full extent of time (as defined by the range of files in a specific folder-path), we do so by calling the function `Coords_Check_avg('path', [coords])`.

```
folderpath_VIIRS = r'C:\Users\OneDrive\LIDAR\Data\LargeDataSet_2'
Coords_Check_avg(folderpath_VIIRS, Coords)
```

The output of this function provides the average value of $K_d(490)$ in the regions provided in the coordinates list. The output of this function can be seen in figure 8.

This function provides a faster method to simply extract the average value of the data in each region over a period of time. However it does not provide a representation of the data variation through time.

2.6 Data variation through time.

We can see in figure 8 that we have computed the average $K_d(490)$ in the multiple regions of interest over a period of time, however this provides us with only a glimpse of the data and we may want to explore how this average value in the region of interest varies through time. To do this we ensure we have downloaded the data we are interested in over an extended period of time, and have stored the folder-path containing the data.

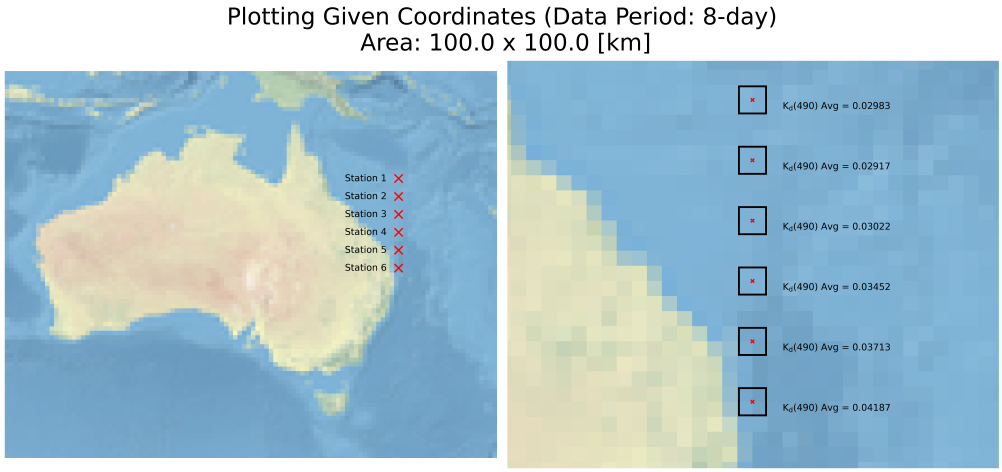


Figure 8: Output figure of the function `Coords_Check_avg()`. The marked regions in the zoomed figure provide a true scale of the chosen size of the area of interest.

```
folderpath_terra = r'C:\Users\OneDrive\LIDAR\Data\LargeDataSet_1'
```

Once we have the folder which contains all the data over an extended period of time, we must also select only one region of interest from out coordinates list, for example we can select and print the first coordinate from the coordinate list.

```
C_List[0]
```

```
array([147.55, 148.45, -17.55, -18.45])
```

After we have selected a single region of interest, we can simply call the plotting function `MultiFile_Data_plotting('path',[coords])`. In the example below we call this function using the set of coordinates given by our coordinates list `C_List`.

```
MutiFile_Data_plotting(folderpath, C_List[0])
```

The output can be seen in figure 9. This function provides us with the spatially averaged value of $K_d(490)$ within the region of interest, over the full range of time (specified by the files in the given folder-path). The function also produces a plot of the percentage of empty data points that were within the region of interest at each moment of time. Additionally this function provides us with the variance of the data at the region of interest, over the full range of time specified by the data set.

2.7 Data at multiple positions and multiple data sets.

Since there are many sources of data from many satellites with different spectrometer instruments onboard, we may like to compare the measurements gathered from these different data sets. To compute and plot the data over an extended period of time, at a single location, and compare this to the same data computed by another satellite we can call the function `MultiFile_MultiInstr_Data_plotting([folders], [coords])`. This function works exactly as the function `MultiFile_Data_plotting()` however we now must pass into this function a list of

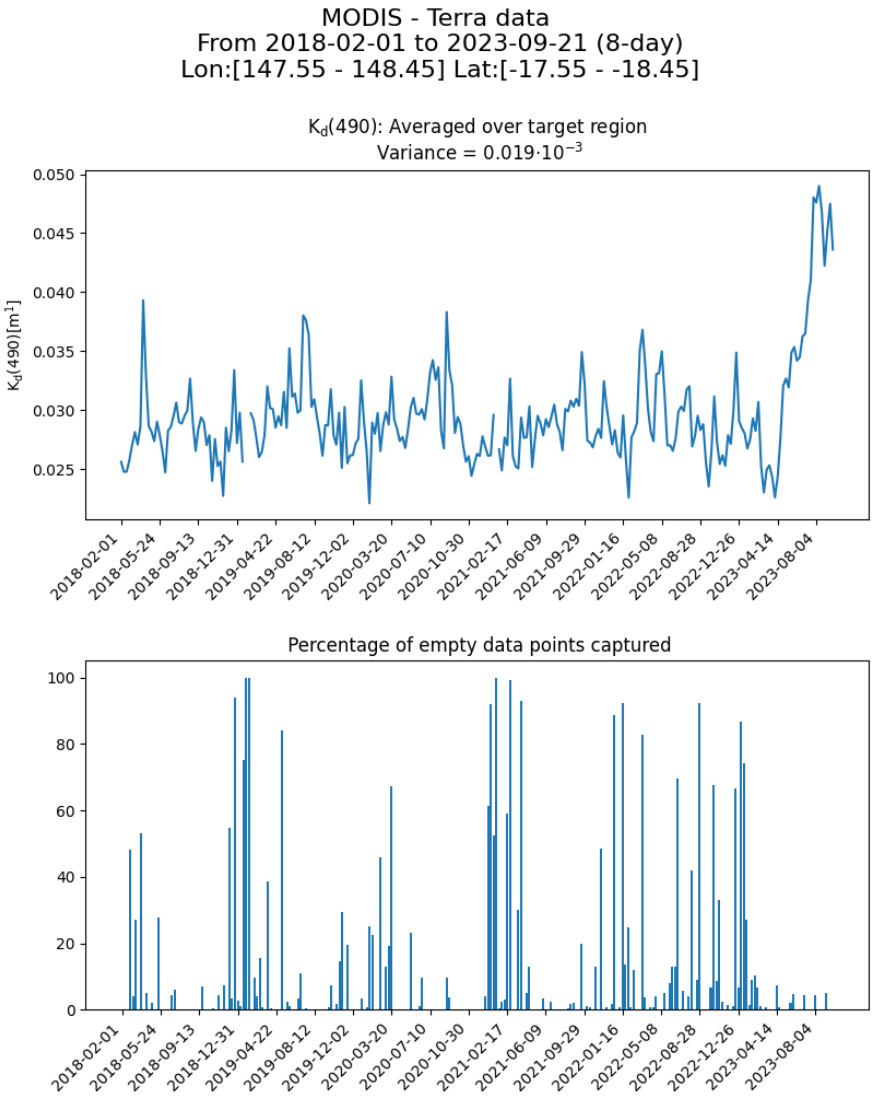


Figure 9: Output of the function `MultiFile_Data_plotting()`. The title of the figure provides information of the satellite and instrument used to capture the data, the full extent of the time range with the time-binning used for the data collection in parentheses and the coordinate extents of the region of interest chosen.

the folder paths for each data set which we wish to compare. To do this we can simply store each dataset path as a variable, for example.

```
folderpath_terra = r'C:\Users\OneDrive\LIDAR\Data\LargeDataSet_1'
folderpath_VIIRS = r'C:\Users\OneDrive\LIDAR\Data\LargeDataSet_2'
folderpath_Aqua = r'C:\Users\OneDrive\LIDAR\Data\LargeDataSet_3'

Folders = (folderpath_terra,folderpath_VIIRS,folderpath_Aqua)
```

We now pass the list of folder-paths into the function as follows.

```
MutiFile_MultiInstr_Data_plotting(Folders, C_List[1])
```

We can now compare the data from multiple instrument datasets, at a single specific location through the entire range of time defined by the files in the folder. The output of this function can be seen in figure 10.

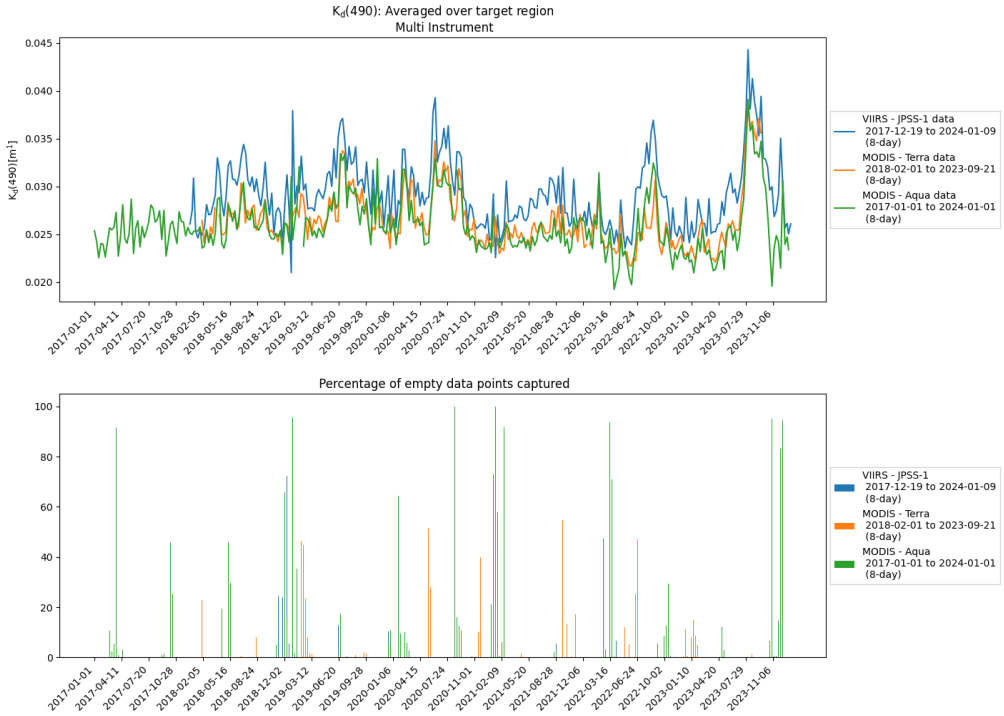


Figure 10: Output figure of the function `MultiFile_MultiInstr_Data_plotting()`.

2.8 Variance at multiple positions

Now that we can compute the data in a region of interest through time, and we have also been able to compute the variance of this data at the region of interest, we may be interested in capturing the variance of the data at multiple positions of interest, for only a single instrument or dataset. Before we plot this data, for pedagogical clarity lets choose to take a larger set of regions of interest using the `CoordsCalcList()` function, where again we choose for the region to be 100 $\times$ 100 km.

```
PosList = [[154,-17],[154,-19],[154,-21],[154,-23],[154,-25],[154,-27]]
C_List = CoordsCalcList(100,PosList)
```

To compute and plot only the variance of the data at these positions we call the function `MultiPos_DataVar_Plot([`*coords*`],[`*positions*`],'`*path*`')`.

```
MultiPos_DataVar_Plot(C_List,PosList,folderpath)
```

This function provides us with the variance of the data at the positions of interest over the full time extent captured by the data provided in the folder-path. The output of this function can be seen in figure 11.

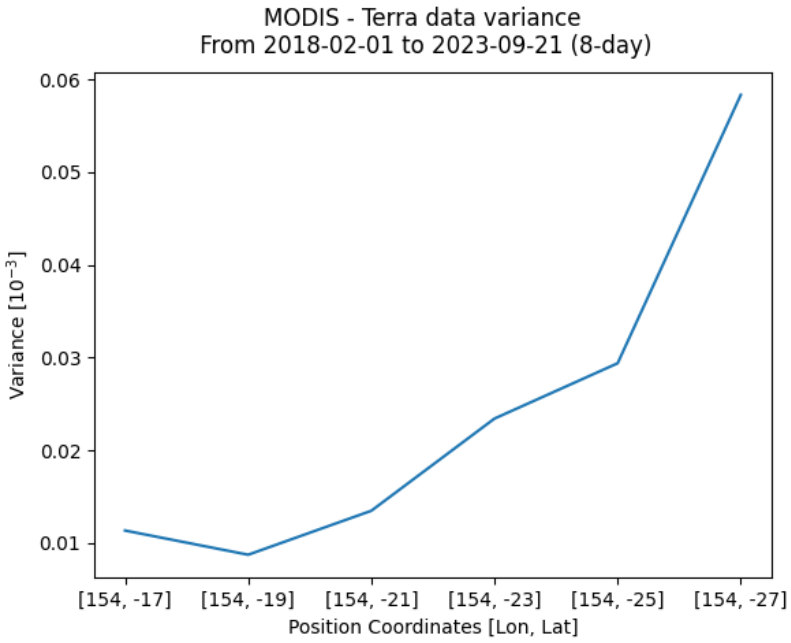


Figure 11: Output figure of the function `MultiPos_DataVar_Plot()`.

2.9 Variance multiple instruments and positions

Similar to the function `MultiFile_MultiInstr_Data_plotting()` we may wish to compare the variance of the data, at multiple positions, across multiple datasets. We do this by calling the function `MultiPos_MultiInstr_DataVar_Plot([coords],[positions],[folders])`

```
Folders = (folderpath_VIIRS,folderpath_terra,folderpath_Aqua)

MultiPos_MultiInstr_DataVar_Plot(Coords,PosList, Folders)
```

In much the same way as the function above, this function will then provide the variance at multiple positions. However we are now able to plot this variance for data captured by multiple instruments, as seen in figure 12. Which can be useful in comparing data gathered by the various instruments at various locations.

2.10 Comparison of data binning

The OceanColor data browser provides the option to collect data which has been binned over various periods of time. For example, we can download data which has been captured each day and compare this to data which is binned into an 8-day period. To compare these data sets we begin by storing the folder-paths as before, for example.

```
VIIRS_weekly = r'C:\Users\OneDrive\LIDAR\Data\LargeDataSet_2'

VIIRS_daily = r'C:\Users\OneDrive\LIDAR\Data\LargeDataSet_4'
```

We once again store these folders into a list.

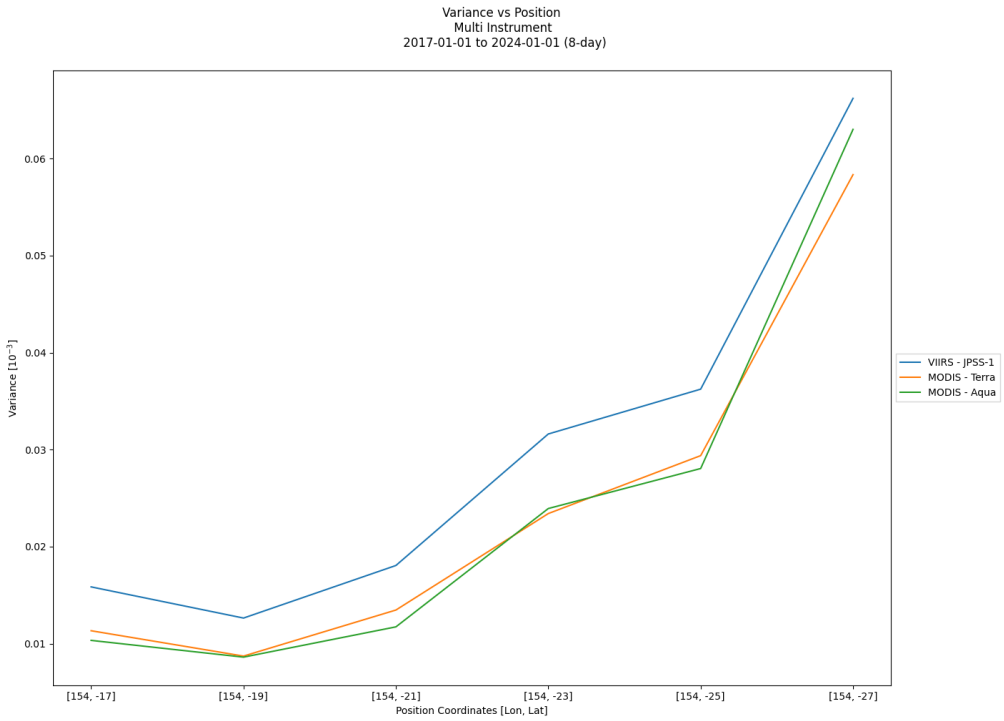


Figure 12: Output figure of the function `MultiPos_MultiInstr_DataVar_Plot()`.

```
Folders_2 = [VIIRS_daily,VIIRS_weekly]
```

We are now ready to pass this folder-path list into the function `Compare_dataBinning_plot('Date Start','Date End', [folders], [coords])`.

```
Compare_dataBinning_plot ('2023-10-18','2023-12-25', Folders_2, Coords[0])
```

This function requires the input of the beginning and end date range to compare the two datasets over, and they should both be given in the format 'YYYY-MM-DD'. The output of this function can be seen in figure 13. From the example output seen in figure 13, the `Compare_dataBinning_plot()` function also provides some statistical analysis of the data. Namely we can see that we have the total variance in the data computed using the weekly or daily binned data-sets in the upper right of the figure. We can also see the weekly variance of the data captured each day overlaid on the upper figure. Further we also compute the percentage root mean squared (or relative standard deviation) which is given as the ratio of the standard variation σ to the mean μ, that is $\%\text{RMS} = \sigma/\mu$.

NOTE: When computing data which spans over two consecutive years (for example taking data from Nov 2023 to Feb 2024) the date selection must be separated into individual years. This is because the 8-day data collected and binned by NASA contains a single 5-day data binning window to collect all 365 days of the year whilst maintaining consistent data for entire calendar years. The 5-day data binning period can cause some peculiar behaviour to the plotting and data analysis and it is recommenced to select only windows of dates which avoid this year cross-over period. Instead break the dates into two separate instances of the function call (for example: function call instance 1 using dates '01-11-2023' - '31-12-2023' and function call instance 2 using dates '01-01-2024' - '01-02-2024'

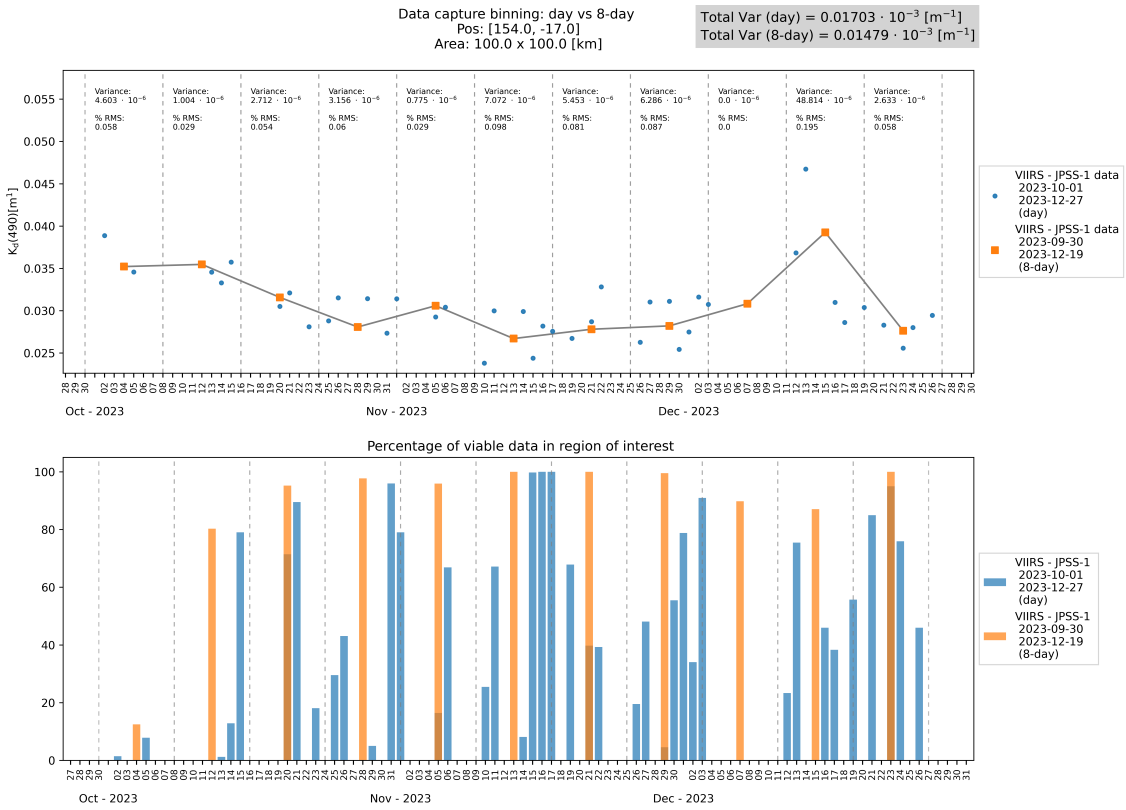


Figure 13: Output figure of the function `Compare_dataBinning_plot()`. The function requires the input of the beginning-date and end-date to compare the datasets over (Format: 'YYYY-MM-DD'). The function also provides the weekly variance of the data captured each day and also the percentage root mean squared (relative standard deviation) which is given as the ratio of the standard variation σ to the mean μ (%RMS $= \sigma/\mu$).

Additionally, observe in the example output of figure 13 the much lower number of viable data points that were been captured at each moment in time when binning data daily, as compared to when binning data weekly. To verify why this has occurred lets plot the data in our chosen region on when the daily binning has a large number of empty points and weekly has a small number. We can choose the daily binning to display the data captured on the 12-12-2023 by selecting the relevant file from the folder path. Similarly we take the weekly data file containing data captured during the week starting on the 11-12-2023.

```
File_daily = MultiFile_FileNames(VIIRS_daily)[-40]
File_weekly = MultiFile_FileNames(VIIRS_weekly)[-5]
```

We call the regional plotting function `KD490_Plot_Data_Region()` for each file. That is, for the single file chosen from the weekly binned dataset

```
KD490_Plot_Data_Region (File_weekly, Coords[0])
```

The output for the weekly binned data can be seen in figure 14. Similarly for the single file chosen

from the daily binned dataset.

```
KD490_Plot_Data_Region (File_daily, Coords[0])
```

The output for the daily binned data can be seen in figure 15. In comparing the two binning types we can clearly see the difference in the data resolution captured.

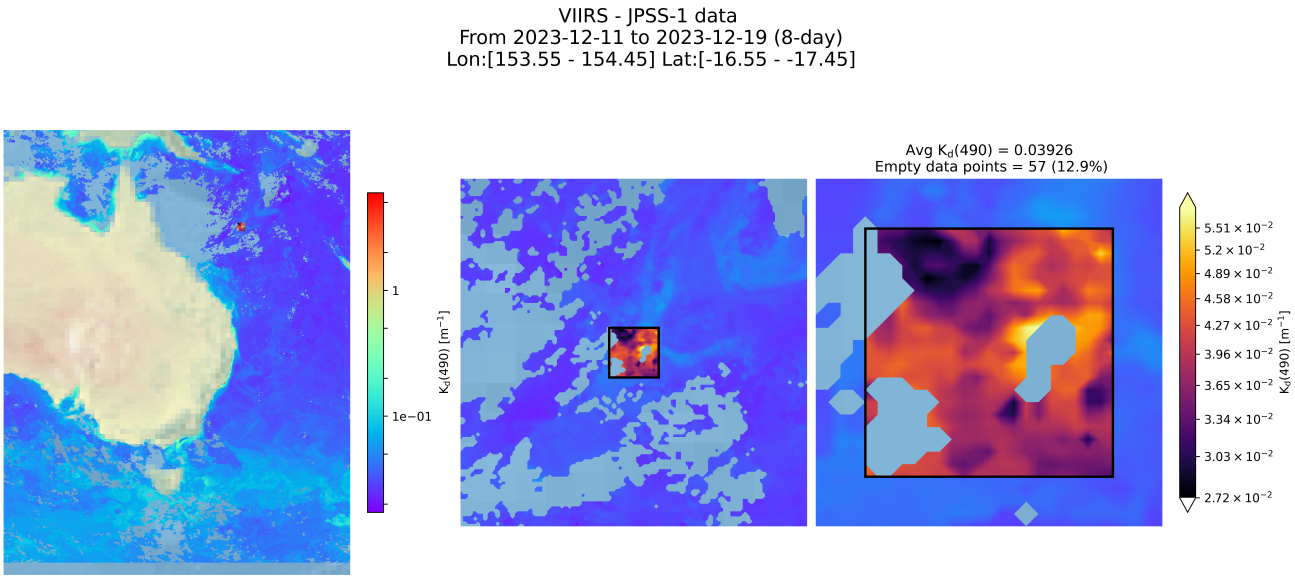


Figure 14: Output figure of the function `KD490_Plot_Data_Region()` for the single file chosen from the weekly binned dataset. The non-coloured areas correspond to those areas which no satellite data could be collected (due mostly to clouds).

2.11 Animating Data

To visualise the data collected by the satellites over an extended period of time one can call the animation function. This function requires the user to provide a folder path which contains a list of `.nc` files and will return the data animated through time. The function also requires the location path in which to save the animation output. Explicitly the function inputs are `Animate_Kd490_Region('file-folder','save-path','name',coords=[],**range)` where we have '*file-folder*' being the location of the data files, '*save-path*' being the desired location to save the animation, `coords=[]` being the coordinates of the region of interest, and finally the keyword optional argument `range=INT` which specifies the number of files to use from the folder, useful if the folder contains a high volume of files and the user wishes to only select only a subset of these. In practice to call the animation function the user may do as follows, firstly specify a folder path,

```
folderpath_terra = r'C:\Users\LIDAR\Data\LargeDataSet_1'
```

Next, we once again generate a coordinate list from a list of positions,

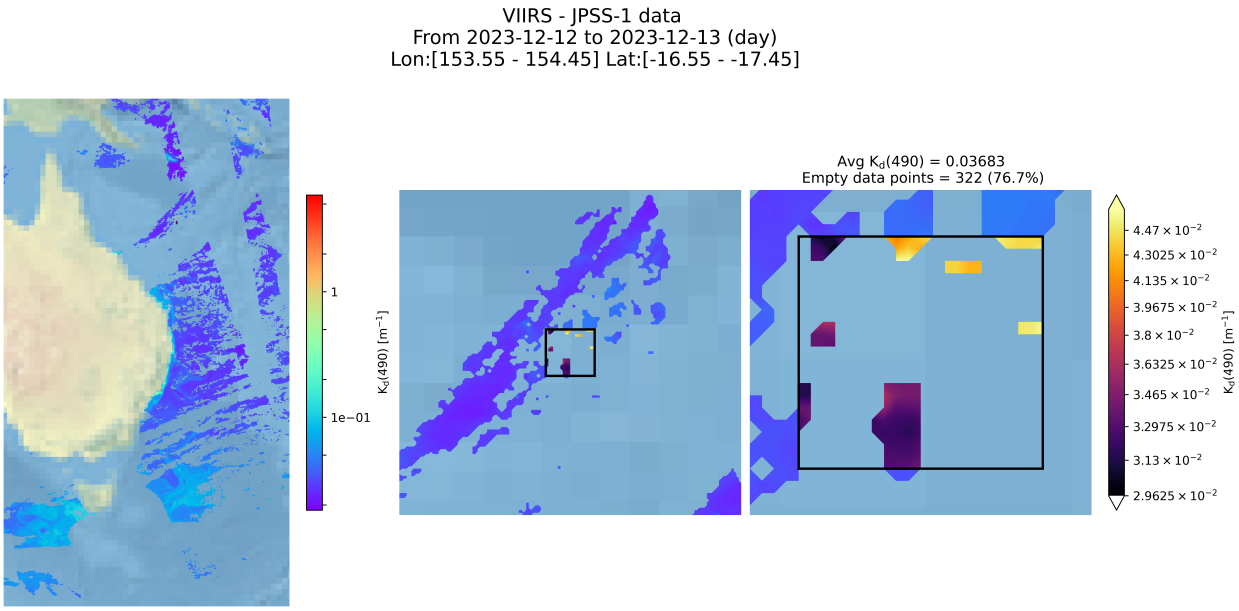


Figure 15: Output figure of the function `KD490_Plot_Data_Region()` for the single file chosen from the daily binned dataset. The non-coloured areas correspond to those areas which no satellite data could be collected (due mostly to clouds).

```
PosList = [[154,-17],[154,-19],[154,-21],[154,-23],[154,-25],[154,-27]]
Coords = CoordsCalcList(100,PosList)
```

and we specify the location which we'd like to save the file in.

```
Saved_path = r'C:\Users\LIDAR\Animations\Terra'
```

Finally we simply call the animation function.

```
Animate_Kd490_Region(folderpath_terra,Saved_path,'Ani_1',Coords[0])
```

This will save the file under the file name 'Ani_1.gif' and provide us with a Gif animation of the data through time.

Alternatively, in exactly the same way as above if we'd like to instead generate the animation as an `.mp4` file rather than a `.gif` we simply call the function,

```
Animate_Kd490_Region_mp4(folderpath_terra,Saved_path,'Ani_1',Coords[0])
```

this function will save the file under the file name 'Ani_1.mp4'. Examples of the animated frames can be seen in figure 16

2.12 Comparing various size regions

One may also be interested in examining the influence of the size of the region of interest on the data captured. To compare the size of a particular region of interest we must first define a list

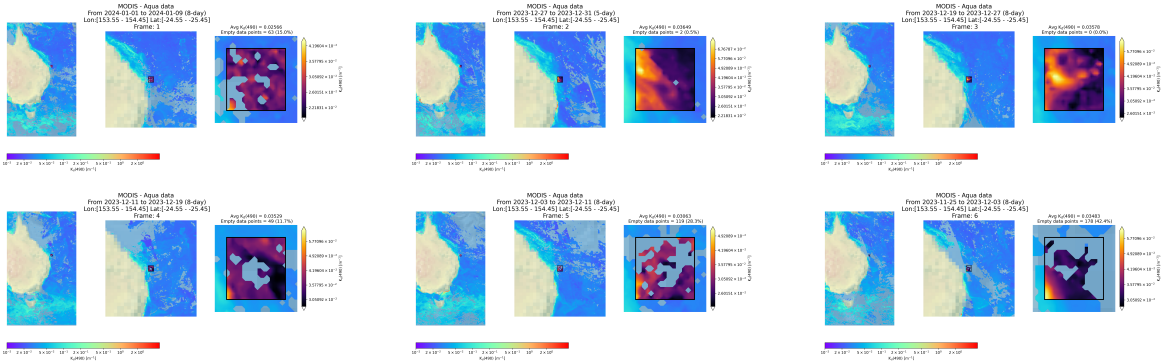


Figure 16: The first few frames of the animated data produced by the functions `Animate_Kd490_Region()` for a .gif file output or `Animate_Kd490_Region_mp4()` for a .mp4 output.

containing the sizes of regions we wish to compare. For example if we wish to compare [100×100] km, [50×50] km and [10×10] km we would define,

```
Sizes = [100,50,10]
```

Assuming we have already defined the file-path and list of positions we may now call the function `MultiPos_MultiROI_DataVar_Plot([sizes],[coords],'folder','start','end')` where we are required to input the time-frame with which we are interested in. **The time-frame inputs for these must be given as strings in the format 'YYYY-MM'**. For example if we wish to plot the data from September 2023 to December 2023 we would call,

```
MultiPos_MultiROI_DataVar_Plot(Sizes,PosList,folder,'2023-09','2023-12')
```

The output of this function can be seen in figure 17. In the figure we notice that we have access to information regarding the number of data points that have been used in the calculation, seen in the title of the figure. Further the output of this figure provides a measure of the standard deviation of the data over this time.

2.13 Monthly Average and Standard Deviation

To compare the data over an extended period of time at multiple positions of interest one can call the function `MultiPos_MonthlyVar_Plot(size,[pos],'folder','start','end')`. To do so by first defining only a few coordinates defining our positions of interest,

```
Pos = [[154,-17],[154,-22],[154,-27]]
```

In the example below we choose to plot the monthly data for the months between '2023-09' and '2023-12'. Further, assuming we have chosen and stored our folder-path as `folder`, and chosen a region size of [50×50] km we would simply call the function as follows,

```
MultiPos_MonthlyVar_Plot(50, Pos, folder, '2023-09','2023-12', Map=False)
```

The output of the function `MultiPos_MonthlyVar_Plot()` can be seen in figure 18.

In particular in figure 18 notice that we have chose to use the kwarg `map = False`, this optional

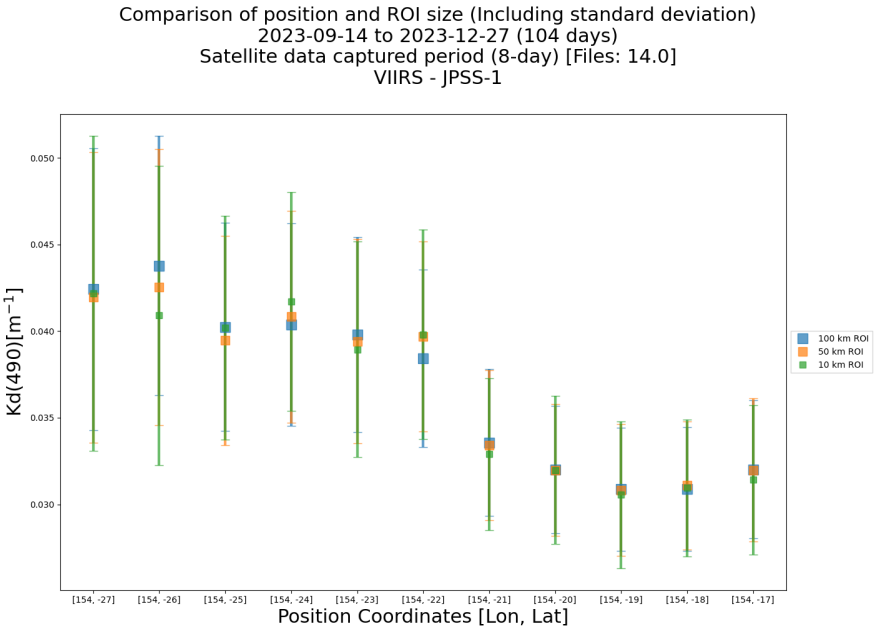


Figure 17: Output figure of the function `MultiPos_MultiROI_DataVar_Plot()` for the selected dates ranging from '2023-09' to '2023-12'. The error bars indicate the standard deviation in the total data captured over this time frame, and the number of files/data points can be seen indicated in the figure title.

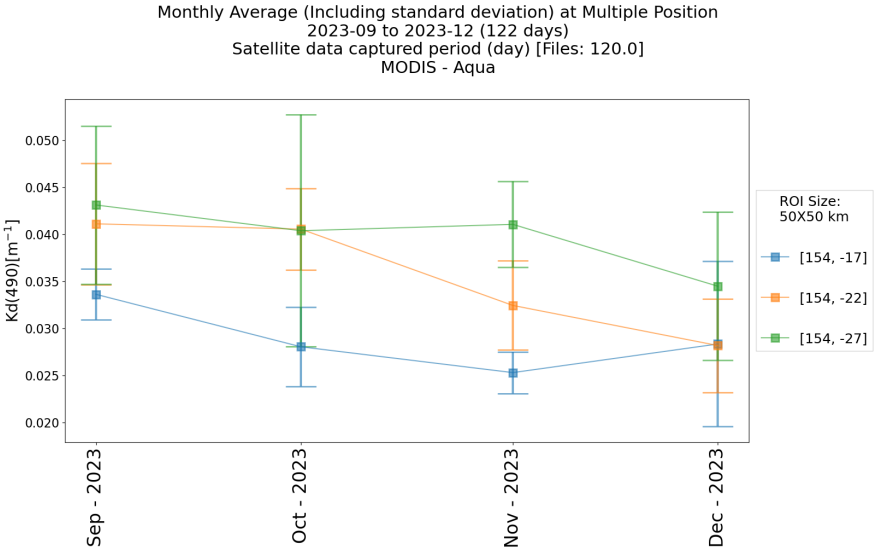


Figure 18: Output figure of the function `MultiPos_MonthlyVar_Plot()` for the selected dates ranging from '2023-09' to '2023-12'. The error bars indicate the standard deviation in the total data captured over this time frame, and the number of files/data points can be seen indicated in the figure title.

argument allows the visualisation of the data points for added convenience. By setting this to `map = True` we are able to visually identify the chosen positions coordinates for easier reference. For example, if we instead call the function as,

```
MultiPos_MonthlyVar_Plot(50, Pos, folder, '2023-09','2023-12', Map=True)
```

We will in fact be able to produce the same data plot however we now get the added benefit of a map reference for our chosen positions of interest. We can see the output of this function in figure 19.

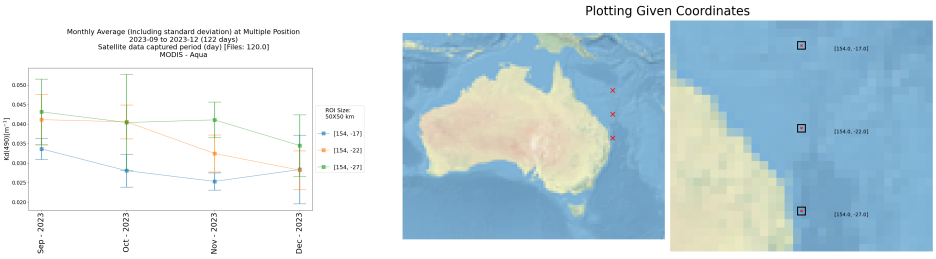


Figure 19: Output figure of the function `MultiPos_MonthlyVar_Plot()` where we have additionally selected the kwarg `Map=True`, this provides a visual of the positions of interest for ease of reference.

With the plotting provided by the function `MultiPos_MonthlyVar_Plot()` and all those which we have seen above, we now have a comprehensive range of tools to analyse and visualise the data collected by satellite spectroradiometers. This data provides the opportunity to build future models of ocean LiDAR systems using accurate historical data in specific geographic locations.

References

[1] National Aeronautics Earthdata and Space Administration (NASA). *Earthdata - Data Processing Levels*. URL: https://www.earthdata.nasa.gov/engage/open-data-services-and-software/data-information-policy/data-levels.

[2] National Aeronautics Earthdata and Space Administration (NASA). *Ocean Color - Remote Sensing Reflectance*. URL: https://oceancolor.gsfc.nasa.gov/resources/atbd/rrs/.

[3] Ziauddin Ahmad et al. "New aerosol models for the retrieval of aerosol optical thickness and normalized water-leaving radiances from the SeaWiFS and MODIS sensors over coastal regions and open oceans". In: *Applied optics* 49.29 (2010), pp. 5545–5560.

[4] Giuseppe Zibordi, Frederic Mélin, and J-F Berthon. "Comparison of SeaWiFS, MODIS and MERIS radiometric products at a coastal site". In: *Geophysical Research Letters* 33.6 (2006).

[5] Zhong-Ping Lee, Ke-Ping Du, and Robert Arnone. "A model for the diffuse attenuation coefficient of downwelling irradiance". In: *Journal of Geophysical Research: Oceans* 110.C2 (2005).

[6] André Morel et al. "Examining the consistency of products derived from various ocean color sensors in open ocean (Case 1) waters in the perspective of a multi-sensor approach". In: *Remote Sensing of Environment* 111.1 (2007), pp. 69–88.

[7] P Jeremy Werdell and Sean W Bailey. "An improved in-situ bio-optical data set for ocean color algorithm development and satellite data product validation". In: *Remote sensing of environment* 98.1 (2005), pp. 122–140.

Quick Reference Guide

```
MultiFile_FileNames(folderpath)
```

Provides the full list of files in a chosen path. Useful when needing to check files in selected folder.

Inputs:

- **folderpath** : string
  The local folder-path containing the netCDF files

Returns:

- The list of the files stored in the path.

```
CoordsCalc(Area, Pos)
```

Call this function to return a list of coordinates defining the region of interest from a **single position** coordinate.

Inputs:

- **Area** : int
  The size of the desired region of interest (ROI) in km.
- **Pos** : array, [Lon,Lat]
  An array containing a single set coordinates defining the centre of a ROI, given as [Longitude, Latitude]

Returns:

- Array containing a single set of coordinates defining the ROI, given as [Lon(W),Lon(E),Lat(N),Lat(S)].

```
CoordsCalcList(Area, PosList)
```

Call this function to return a list of coordinates defining the region of interest from a **list** of coordinates.

Inputs:

- **Area** : int
  The size of the desired region of interest (ROI) in km.
- **PosList**: array, [[Lon_1,Lat_1],[Lon_2,Lat_2], ...]
  An array of coordinates defining the centre of multiple ROIs, given as [Longitude, Latitude]

Returns:

- Array containing the coordinates defining the ROI, given as [[Lon_1(W),Lon_1(E),Lat_1(N),Lat_1(S)], [Lon_2(W),Lon_2(E),Lat_2(N),Lat_2(S)], ...].

```
KD490_Plot_Data(file, **save=' ')
```

Function to plot the data stored in a single NETCDF file.

Inputs:

- **file** : string
  The local file-path of a netCDF file.
- **save**=' ' : **kwargs
  Optional: Save the figure as the desired filename. For example, save='Figure.png'

Returns:

- The data stored in the netCDF file plotted over the default map from cartopy.

```
Coords_Check(coords, **save)
```

Function to plot and display a map of the coordinates calculated. Useful to visualise and confirm the coordinates generated.

Inputs:

- **coords** : array, [[W_1,E_1,N_1,S_1],[W_2,E_2,N_2,S_2], ...]
  An array of coordinates defining a region(s) of interest .
- **save**=' ' : **kwargs
  Optional: Save the figure as the desired filename. For example, save='Figure.png'

Returns:

- The region(s) of interest plotted over the default map from cartopy.

```
Coords_Check_avg(folderpath,coords, **save)
```

Function to plot and display the average value of the data within a region(s) of interested, defined by the coordinates. Useful to visualise and confirm the coordinates generated and quickly access a average value of the data.

Inputs:

- **folderpath** : string
  The local folder-path containing the netCDF files.
- **coords** : array, [[W_1,E_1,N_1,S_1],[W_2,E_2,N_2,S_2], ...]
  An array of coordinates defining a region(s) of interest .
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Figure.png’

Returns:

- The average value of the data in the region(s) of interest plotted over the default map from cartopy.

```
KD490_Plot_Data_Region(file, coords, **grid, **save)
```

Function to plot and display the data within a region of interest (ROI), defined by the coordinates. Useful to visualise and confirm the data collected withing a ROI.

Inputs:

- **file** : string
  The local file-path of a netCDF file.
- **coords** : array, [W,E,N,S]
  An array of coordinates defining a region of interest .
- **grid**=Bool : **kwargs
  Optional: Display the raw grid of data points in the region. Useful for validating the data points taken in the ROI.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Figure.png’

Returns:

- The data plotted over a map, with a subplots providing a detailed view of the ROI and the data within the ROI.

```
KD490_Region_Avg (file, coords)
```

Function to compute the average value of the data stored in a single file. Returns an array containing the values computed, and information regarding the data used.

Inputs:

- **file** : string
  The local file-path of a netCDF file.
- **coords** : array, [W,E,N,S]
  An array of coordinates defining a region of interest .

Returns:

- **array** : [Avg, NaNs, T_0, T_f, Period, Count, Inst]
  An array containing the average value in the ROI (Avg), The percentage of empty data points within the ROI (NaNs), the start date and time of the capture (T_0), the end date and time of the capture (T_f), the time the data was capture over (Period), the total number of data counts used (Count), The instrument used to capture the data (Inst).

```
MultiFile_Reg_avg (folderpath, coords)
```

Function to compute the average value of the data stored in a multiple files stored in the same folder. Returns an array containing the values computed, and information regarding the data used..

Inputs:

- **folderpath** : string
  The local folder-path containing the netCDF files.
- **coords** : array, [W,E,N,S]
  An array of coordinates defining a region of interest .

Returns:

- **array** : [Avg, NaNs, T_0, T_f, Period, Date, Count, Inst]
  An array containing the average value in the ROI (Avg), The percentage of empty data points within the ROI (NaNs), the start date and time of the capture (T_0), the end date and time of the capture (T_f), the time the data was capture over (Period), the date of the data capture (Date), the total number of data counts used (Count), The instrument used to capture the data (Inst).

```
MutiFile_Data_plotting (folderpath, coords,**save)
```

Function to plot the average value of the data stored in a multiple files stored in the same folder.

Inputs:

- **folderpath** : string
  The local folder-path containing the netCDF files.
- **coords** : array, [W,E,N,S]
  An array of coordinates defining a region of interest.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Figure.png’

Returns:

- The plot of the average value within a region of interest over an extended periods of time, defined by the extent of the files in the specified folder.

```
MultiPos_DataVar(CoordsList, folderpath)
```

Function to compute the variance in the data over multiple regions of interest through time.

Inputs:

- **CoordsList** : array, [[W_1,E_1,N_1,S_1],[W_2,E_2,N_2,S_2], ...]
  An array of coordinates defining a region(s) of interest .
- **folderpath** : string
  The local folder-path containing the netCDF files.

Returns:

- **Variance** : array
  An array of the variance of the data within the multiple regions of interest for the total time extent defined by the specified folder.

```
MultiPos_DataVar_Plot(CoordsList,PosList, folderpath,**save)
```

Function to plot the variance in the data over multiple regions of interest through time

Inputs:

- **CoordsList** : array, [[W_1,E_1,N_1,S_1],[W_2,E_2,N_2,S_2], ...]
  An array of coordinates defining a region(s) of interest .
- **PosList**: array, [[Lon_1,Lat_1],[Lon_2,Lat_2], ...]
  An array of coordinates defining the centre of multiple ROIs, given as [Longitude, Latitude]
- **folderpath** : string
  The local folder-path containing the netCDF files.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Figure.png’

Returns:

- A plot displaying the variance in the data at multiple positions of interest.

```
MutiFile_MultiInstr_Data_plotting (folderpaths, coords, **save)
```

Function to plot the data captured at a region of interest (ROI) through time and by multiple instruments. Useful in comparing the data captured by other instruments for validation of values.

Inputs:

- **folderpaths** : array
  An array containing multiple local folder-paths for separate data-sets collected by different instruments.
- **coords** : array, [W,E,N,S]
  An array of coordinates defining a region of interest.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Figure.png’

Returns:

- A plot displaying the data over a period of time as defined by the specified folders, overlaid to compare multiple instruments.

```
MultiPos_MultiInstr_DataVar_Plot (coords, PosList, folderpath, **save)
```

Function to plot the data variance at multiple regions of interest (ROIs) through time and by multiple instruments. Useful in comparing the data captured by other instruments for validation of values.

Inputs:

- **CoordsList** : array, [[W_1,E_1,N_1,S_1],[W_2,E_2,N_2,S_2], ...]
  An array of coordinates defining a region(s) of interest .
- **PosList**: array, [[Lon_1,Lat_1],[Lon_2,Lat_2], ...]
  An array of coordinates defining the centre of multiple ROIs, given as [Longitude, Latitude]
- **folderpath** : array
  An array containing multiple local folder-paths for separate data-sets collected by different instruments.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Figure.png’

Returns:

- A plot displaying the data variance over a period of time as defined by the specified folders overlaid to compare multiple instruments.

```
MultiPos_Data_plot(CoordsList, PosList, folderpath, **save)
```

Function to plot the data through time at multiple regions of interest (ROIs).

Inputs:

- **CoordsList** : array, [[W_1,E_1,N_1,S_1],[W_2,E_2,N_2,S_2], ...]
  An array of coordinates defining a region(s) of interest .
- **PosList**: array, [[Lon_1,Lat_1],[Lon_2,Lat_2], ...]
  An array of coordinates defining the centre of multiple ROIs, given as [Longitude, Latitude]
- **folderpath** : array
  An array containing multiple local folder-paths for separate data-sets collected by different instruments.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Figure.png’

Returns:

- A plot displaying the data over a period of time as defined by the specified folders overlaid to compare multiple instruments.

```
Animate_Kd490_Region(folderpath, savepath, name, coords, **save,
**range)
```

Function to animate the temporal change in data through time at a specific region of interest (ROI). This function will produce a .gif for the animated file.

Inputs:

- **folderpath** : string
  The local folder-path containing the netCDF files.
- **savepath** : string
  The local folder-path where the animation will be saved.
- **name** : string
  The name of the file for the animation once saved.
- **coords** : array, [W,E,N,S]
  An array of coordinates defining a region of interest.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Animation’ (Note: No need for file extension)
- **range**=‘ ’ : **kwargs
  Optional: Save the number of files in the folder to select. For example, range=30.

Returns:

- A gif with the data in the folder animated through time

```
Animate_Kd490_Region_mp4(folderpath, savepath, name, coords, **save,
**range)
```

Function to animate the temporal change in data through time at a specific region of interest (ROI). This function will produce a .mp4 for the animated file.

Inputs:

- **folderpath** : string
  The local folder-path containing the netCDF files.
- **savepath** : string
  The local folder-path where the animation will be saved.
- **name** : string
  The name of the file for the animation once saved.
- **coords** : array, [W,E,N,S]
  An array of coordinates defining a region of interest.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Animation’ (Note: No need for file extension)

- **range**=‘ ’ : **kwargs
  Optional: Save the number of files in the folder to select. For example, range=30.

Returns:

- An mp4 with the data in the folder animated through time.

```
MultiPos_MultiROI_DataVar_Plot(SizeList, PosList, folderpath, start,
end, **save)
```

Function to plot the average of the data and the standard deviation for multiple ROI sizes. Useful to compare the influence of the size of the sampled area.

Inputs:

- **SizeList** : array, [s_1, s_2, ...]
  An array containing the sizes of the ROI in km. For example for a 100×100km and a 10×10km ROI, SizeList=[100, 10]
- **PosList**: array, [[Lon_1,Lat_1],[Lon_2,Lat_2], ...]
  An array of coordinates defining the centre of multiple ROIs, given as [Longitude, Latitude]
- **folderpath** : string
  The local folder-path containing the netCDF files.
- **start** : string, YYYY-MM
  The starting month in which data is to be computed over.
- **end** : string, YYYY-MM
  The ending month in which data is to be computed over.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Animation’ (Note: No need for file extension)

Returns:

- A figure which provides the standard deviation of the data and the average over the time defined by the start and end dates chosen for multiple ROI sizes.

```
MultiPos_MonthlyVar_Plot(Size, PosList, folderpath, start, end, **save,
**Map)
```

Function to plot the monthly average of the data and the standard deviation through time. Useful to compare the changes in the data of the size at multiple positions of interest.

Inputs:

- **Size** : Integer
  Integer defining the size of the ROI in km. For example for a 100×100km ROI, Size = 100
- **PosList**: array, [[Lon_1,Lat_1],[Lon_2,Lat_2], ...]
  An array of coordinates defining the centre of multiple ROIs, given as [Longitude, Latitude]
- **folderpath** : string
  The local folder-path containing the netCDF files.
- **start** : string, YYYY-MM
  The starting month in which data is to be computed over.
- **end** : string, YYYY-MM
  The ending month in which data is to be computed over.
- **save**=‘ ’ : **kwargs
  Optional: Save the figure as the desired filename. For example, save=‘Animation’ (Note: No need for file extension)
- **Map**=Bool : **kwargs
  Optional: Display a map of the chosen coordinates for ease of reference when plotting data in coordinates. Boolean kwarg, to display map simply input, Map = True

Returns:

- A figure which provides the monthly data standard deviation and average at multiple positions, for a total period defined by the start and end dates chosen.
