dardar_tool Python Package User Manual

Arttu Väisänen*, FMI
Larisa Sogacheva, FMI

December 10, 2025

Contents

*arttu.vaisanen[a]fmi.fi

1 User Manual

This document contains a brief description how to setup and use the dardar_tool.

2 Information

2.1 General

DARDAR (raDAR/liDAR) is an algorithm that combines data from NASA's CloudSat radar and the CALIPSO lidar to create comprehensive cloud products by leveraging the strengths of both instruments. By applying the optimal estimation, DARDAR compares simulated radar and lidar backscatter with actual measurements to determine the best estimate of cloud properties, even when one instrument's signal is attenuated or missing. The algorithm (Delanoe and Hogan, 2008, 2010; Cazenave et al., 2019) retrieves cloud properties, such as particle size and ice water content, and produces detailed masks for cloud and aerosol classification with high vertical and horizontal resolution. The combination of radar's ability to penetrate deep clouds and lidar's sensitivity to thin clouds and liquid water offers a more complete picture of cloud structure and microphysics than either instrument could provide alone. DARDAR products are DARDAR_CLOUD and DARDAR_MASK. DARDAR_CLOUD focuses on retrieving ice cloud properties and microphysics from the collocated CloudSat and CALIPSO measurements. DARDAR_MASK provides a detailed cloud and aerosol classification mask.

DARDAR V3.00 uses the usual Brown and Francis modified mass-size relationship (https://doi.org/10.25326/450), while DARDAR V3.10 uses the Heymfield's composite mass-size relationship (https://doi.org/10.25326/449).

dardar_tool Python Package software provides functionality to download DARDAR_CLOUD and DARDAR_MASK product data files from ICARE portal (https://www.icare.univ-lille.fr/dardar/). Extraction, plotting, and data resolution modification functionalities are only provided for DARDAR_CLOUD files. Detailed descriptions of this functions with examples are provided in section 4.

References

[1] Delanoë, J., and R. J. Hogan, 2008: *A variational scheme for retrieving ice cloud properties from combined radar, lidar, and infrared radiometer*, J. Geophys. Res., 113, D07204, doi:10.1029/2007JD009000.

[2] Delanoë, J., and R. J. Hogan, 2010: *Combined CloudSat-CALIPSO-MODIS retrievals of the properties of ice clouds.*, J. Geophys. Res., 115, D00H29, doi:10.1029/2009JD012346.

[3] Cazenave, Q., Ceccaldi, M., Delanoë, J., Pelon, J., Groß, S., and Heymsfield, A.: *Evolution of DARDAR-CLOUD ice cloud retrievals: new parameters and impacts on the retrieved microphysical properties*, Atmos. Meas. Tech., 12, 2819–2835, https://doi.org/10.5194/amt-12-2819-2019, 2019.

2.2 Technical Notes:

Some important technical notes that user should keep in mind.

2.2.1 Setup files

Setup_files are created with **setup()** function provided by the package. These setup_files will contain the encrypted server access information for the download functionalities of the package. Download functionalities need to know setup_file location to be able to use this information for server access. Functions that has these contain the variable **key_location** that indicates the location. Some of the functions this is not mandatory if download options are not used.

Important: Don't share these files to any else because they contain the passwords and username.

2.2.2 Look Up Table

Package uses a look up table for quick processing of user spatiotemporal location searches. The spatial location is in a $30° \times 30°$ grid and the search functions use this to locate the correct files. Because of this the search is conservative and even though **download()** function has downloaded some data to users computer **extractor()** function might not pick anything up. **check_overpass()** function with plotting gives the idea of area where the data is located.

2.2.3 Versiot

Software provides access to tree different DARDAR products, that are listed on table 1. Table shows also the time ranges that are accessible with this tools **download()** function.

Variable	Time range	Description
V30	2006-2019	Version 3-00: algorithm version 3.0.1, Brown and Francis modified mass-size relationship
V31	2006-2019	Version 3-10: algorithm version 3.0.1, Heymfield's composite mass-size relationship
V2	2006-2017	Version 2.1.1: algorithm version 2.1.1, Brown and Francis modified mass-size relationship

Table 1: Table of data accessible by this tool. First column is the variable name for functions, second is the provided time range and third the short describtion of the product.

2.3 How to cite

Arttu Väisänen, & Larisa Sogacheva. (2025). A tool to download and process DARDAR data for model evaluation. Finnish Meteorological Institute. https://doi.org/10.57707/FMI-B2SHARE.593C60A403C844EE857A21775C97838D

2.4 License

AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.5 Acknowledgements

The development of this software has been supported by European Union's Horizon 2020 research and innovation programme under grant agreement No. 101003826 via project CRiceS. Additionally, we wish to thank Antti Kukkurainen for his work testing the software and ICARE for providing the data and download service (https://www.icare.univ-lille.fr/dardar/).

3 Setup

To do a quick setup of this software follow these steps.

1. Create a icare account: https://www.icare.univ-lille.fr

2. Install python3 to your computer. We recommend conda.

2.1 **Important:** The software is locked in python version 3.10.18 so this version needs to be used. We recommend to use an separate environment for the use of this pakage.

3. After python has been installed, install the package with the following line:

pip install git+https://github.com/crices-h2020/dardar_tool.git

4. **Optional, but recommended:** Install IPython or an editor that uses it for better plotting functionality. **Note:** Also matplotlib pakages backend can have an impact to plotting functions performance. We have tested macosx, TkAgg, QtAgg backends with IPython and iniline with Spyder editor and those have worked as intended. Also operating system might have and effect on these.

5. Import dardar_tool with following line:

```
import dardar_tool as dt
```

After import run function dt.setup() from package to create setup files.
Note: download functionalities of this package need to know the key_location, ergo the location of these setup files.

6. You can test that everything works by printing all function handles in following way.

```
import dardar_tool as dt

print(dt.download)
print(dt.extractor)
print(dt.extract_compare)
print(dt.check_overpass)
print(dt.read_data)
print(dt.plot_saved_data)
print(dt.plot_local_data)
print(dt.compare_user_data)
print(dt.column_cumsum)
print(dt.scale_resolution)
```

4 Functions

All functions are described under in detail.

4.1 download

4.1.1 Description

Function downloads DARDAR_CLOUD or DARDAR_MASK files to user computer based on input. Spatiotemporal input is used to create a coarse time location info for the algorithm that is used to download all files that may contain relevant data. Because of this files might not contain any relevant data when analysed futher with other functions. **Note**: DARDAR_MASK files are not otherwise supported by this software.

4.1.2 Function

```
download(start_date, end_date, lat_1, lat_2, lon_1, lon_2,
         version="V30", cloud_0_mask_1=0,
         save_path=None , key_location="")
```

4.1.3 Output

Function downloads DARDAR_CLOUD or DARDAR_MASK files to user specified location or uses the default which is working directory.

4.1.4 Input variables

start_date: Starting date and time
end_date: Ending date and time
Accepted time format is python strings with following notations: YYYY-MM-DD, YYYY-MM-DD:hh, YYYY-MM-DD:hh-mm. ATTENTION: If just dates are used without hours or minutes, code will interpret them just as the first minute of the day.

lat_1: Upper latitude. Accepted range: [-90, 90]
lat_2: Lower latitude. Accepted range: [-90, 90]
lon_1: Accept range: [-180, 180]
lon_2: Accept range: [-180, 180]
Longitudes can be provided in two ways with the ranges. Either lon_1 $<$ lon_2 or lon_1 $>$ lon_2 which rotates the square.

Area of interest is given with four coordinate variables. Coordinates create a four sided square that is used to identify the area of interest. Coordinates follow the pattern presented in figure 1.

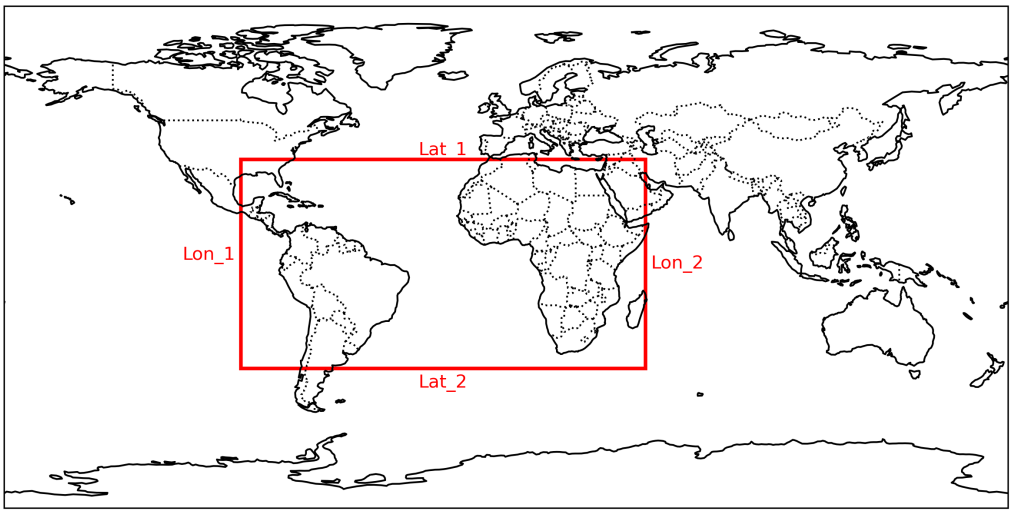

Figure 1: Coordinate search area box definition.

version: (default="V30") is for choosing the product version to use. By default the chosen product is "V30". Table 2 shows the supported products by this software, time range, and some additional information.

Variable	Time range	Description
V30	2006-2019	Version 3-00: algorithm version 3.0.1, Brown and Francis modified mass-size relationship
V31	2006-2019	Version 3-10: algorithm version 3.0.1, Heymfield's composite mass-size relationship
V2	2006-2017	Version 2.1.1: algorithm version 2.1.1, Brown and Francis modified mass-size relationship

Table 2: Supported product, availability time range, and description for more information.

cloud_0_mask_1: (default=0) downloads the DARDAR_CLOUD files, 1: downloads DARDAR_MASK files.

save_path: Path were downloaded files are saved. Default location is the current working directory.

key_location: (default="") To download files, password needs to be provided to get access to the servers. This is done by providing the directory location of the setup files created with setup() function. By default the location is assumed to be the current working directory.

4.1.5 Example

An example of using this function could be that user wants to get data from area enclosed by latitudes 36° and 45° and longitudes 6° and 12° from time interval 2013-12-1 to end of 2013-12-2. In the code date 2013-12-3 is used as en time because this is the first minute of the next day. User has the setup files located in directory: example, that is then told to function. User also wants the files to be downloaded to current working directory so the save_path is left empty. Code is showcased under:

```
import dardar_tool as dt

dt.download("2013-12-01", "2013-12-03", 45, 36, 6, 12,
        key_location="./example")
```

4.2 extractor

4.2.1 Description

Function extracts data from users chosen variable from a user specified area of interest in a user defined time range. User can provide their own vertical profile or use DARDAR products own profile.

Note: Files might not contain any information related to user specified spatiotemporal location even though they were downloaded. This is because download function uses a coarse location information that includes all files that may contain relevant inforamtion.

Support of this function is only provided to DARDAR_CLOUD files.

Function returns the extracted data in either python variables or saves the data on netcdf files.

4.2.2 Function

```
time_array, latitude_array, longitude_array,
vertical_profile, variable
=
extractor(start_date, end_date, lat_1, lat_2, lon_1, lon_2,
            variable_name, vertical_profile=None,
            exclude_top_on=True, exclude_bottom_on=True,
            verbose=1, auto_load_missing_files=False,
            version="V30", save_results=False,
            save_results_name="cloud_data",
            raw_data_loc=None,
            key_location="")
```

4.2.3 Output

Function will return time array, latitude array, longitude array, vertical profile, and variable that is extracted from the given data and defined spatiotemporal location. Data is either returned as python variables or saved to a netcdf file, depending on the input variables.

4.2.4 Input variables

start_date: Starting date and time
end_date: Ending date and time
Accepted time format is python strings with following notations: YYYY-MM-DD, YYYY-MM-DD:hh, YYYY-MM-DD:hh-mm. ATTENTION: If just dates are used without hours or minutes, code will interpret them just as the first minute of the day.

lat_1: Upper latitude. Accepted range: [-90, 90]
lat_2: Lower latitude. Accepted range: [-90, 90]
lon_1: Accept range: [-180, 180]
lon_2: Accept range: [-180, 180]
Longitudes can be provided in two ways with the ranges. Either lon_1 < lon_2 or lon_1 > lon_2 which rotates the square.

Area of interest is given with four coordinate variables. Coordinates creates a four sided square that is used to identify the area of interest. Coordinates follow the pattern presented in figure 1.

variable_name: Takes DARDAR_CLOUD variable name that user is interested in. Output of function will vary depending if chosen variable contains a vertical profile or not. In table 3 are listed retrieved variables of DARDAR product.

DARDAR product variable	Description
extinction	Retrieved visible extinction coefficient
effective_radius	Retrieved effective radius
lidar_ratio	Retrieved extinction-to-backscatter ratio
N0star	Retrieved normalized number concentration param.
iwc	Retrieved Ice Water Content
DARMASK_Simplified_Categorization	DARMASK Categorization Flags

Table 3: Some variables from DARDAR product and description of them.

vertical_profile: (default=None) User can provide their own vertical profile which the variable is converted to using weighted average. Vertical profile needs to be provided as a numpy array. Function handles both ascending and descending orders by converting them to ascending. Output also obeys this rule. If DARDAR variable does not contain a vertical profile, code ignores the input. If user do not have their own vertical profile or they want to use the DARDAR products own profile, vertical_profile variable can be

ignored or set to None.

In the case that the users vertical_profile is smaller than the DARDAR products vertical profile following options determine the behavior of the averaging. This case is presented in figure 2 left.

> **exclude_top_on:** (default=True) Determines the behavior on top of user profile, figure 2 left a. If set True the part a will not be calculated in to the average of last height. If set False a will be included in averaging to the last layer of user profile H_6.
>
> **exclude_bottom_on:** (default=True) Determines the behavior on bottom of user profile figure 2 left b. If set True the part b will not be calculated in to the average of the bottom height. If set False b will be included in averaging to the bottom layer of user profile H_0.

In the case that users profile is larger than the DARDAR products vertical profile, figure 2 right, the values that exceed this profile is set to zeros, in figure c and d.

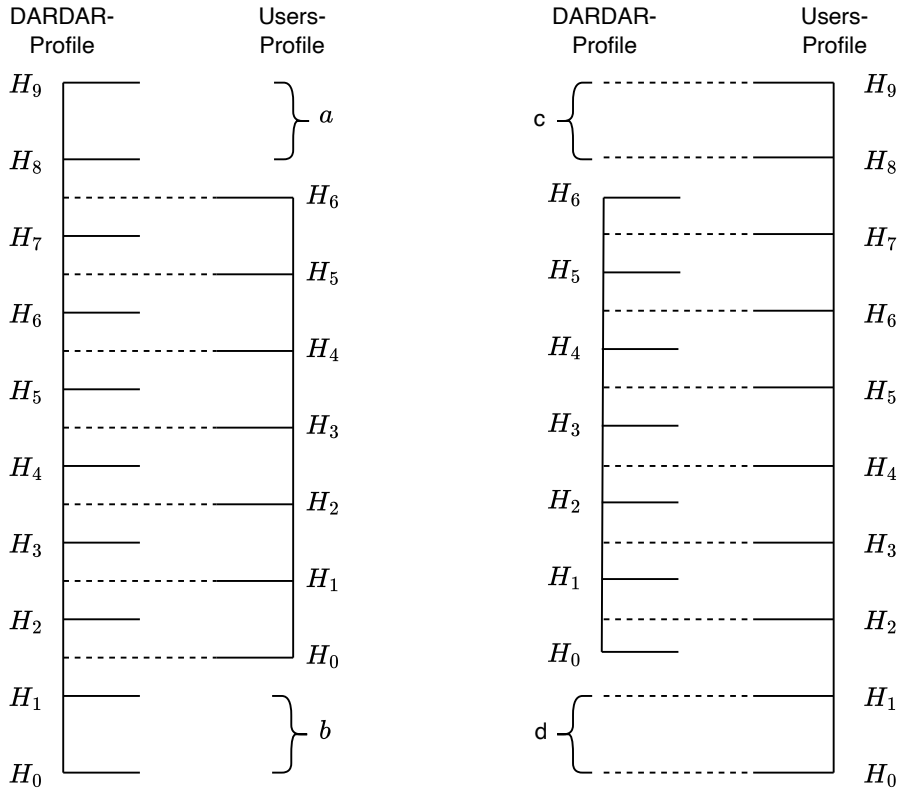

Figure 2: On the left a case were users vertical profile is smaller than the DARDAR products own. On the right a case were users profile is larger than the DARDAR products profile.

verbose: (default=1) Function has 3 levels of printing (0, 1, 2). 0=No printing except errors and warnings. 1=Limited. 2=Extensive.

auto_load_missing_files: (default=False) If True, code automatically downloads all missing data for the user chosen spatio temporal location. Files will be downloaded to raw_data_loc directory. User needs to provide the key_location variable for this to work if the setup files are not in working directory.

version: (default="V30") is for choosing product version to use. By default the chosen product is "V30". Table 2 shows the supported products by this software, time range, and some additional information.

save_results: (default=False) If you want to save results, select True for this option and the data will be saved to a netcdf4 file. NOTE: if time range or area of interest is

big, using save option is recommended.

save_results_name: (default="cloud_data") If results will be saved, location and file name can be adjusted with this input variable. By default results will be saved in working directory with file name cloud_data.

raw_data_loc: (default=None) User can specify the directory containing downloaded DARDAR_CLOUD data. Code tries by default to find it from the working directory.

key_location: (default="") To download files password needs to be provided to get access to the servers. This is done by providing the directory location of the setup files created with setup() function. This option is needed for the auto_load_missing_files to work. By default the location is assumed to be the current working directory.

4.2.5 Example

An example of using this function could be that user wants to extract data from previously downloaded files located in directory example_dir. User is interested in extraction of Retrieved Ice Water Content so variable "iwc" is used. User also wants the data in python variables so saving is not needed, which is the default option. Spatiotemporal location information of the region of interest is given and it is latitudes 36° and 45°, longitudes 6° and 12°, and time interval 2013-12-1 to end of 2013-12-2. In the code date 2013-12-3 is used as en time because this is the first minute of the next day.

```
import dardar_tool as dt

tim_arr, lat_ar, lon_arr, vert_prof, var = dt.extractor("2013-12-01",
                                                        "2013-12-03", 45, 36, 6, 12,
                                                        "iwc", raw_data_loc="./exaple_dir")
```

4.3 extract_compare

4.3.1 Description

Function plots user defined vertical profile of the defined spatiotemporal location and compares it with the DARDAR standard profile. This is done flight wise which are separated by the flight separation parameters.

4.3.2 Function

```
extract_compare(start_date, end_date, lat_1, lat_2, lon_1, lon_2,
                variable_name, vertical_profile,
                exclude_top_on=True, exclude_bottom_on=True,
                flight_separation_time=1, flight_separation_unit="m",
                colormap="viridis", verbose=1,
                auto_load_missing_files=False, version="V30",
                raw_data_loc=None,
                key_location="")
```

4.3.3 Output

Function returns a vertical profile comparison (figure 3) of separate flights. Function calculates a new averaged vertical profile based on users vertical profile. The new profile is shown in figure 3 on the left, and DARDAR-product profile on the right.

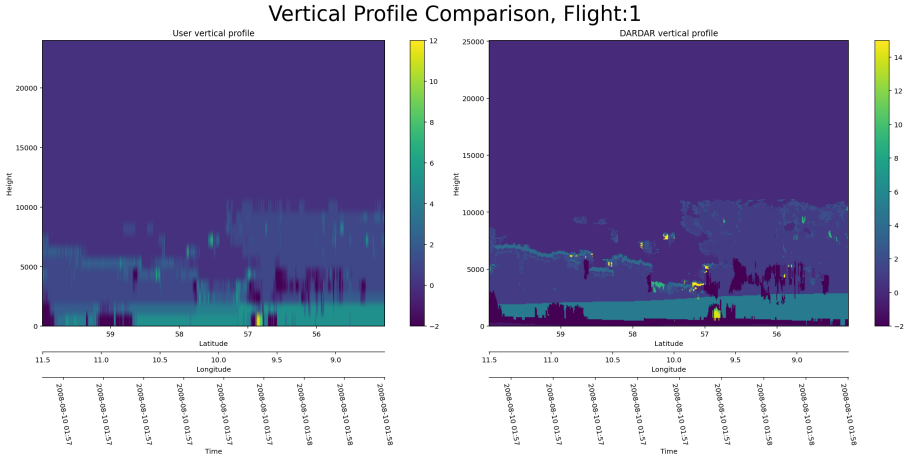

Figure 3: Example figure of comparison function output. On left is users vertical profile based image and on right is the DARDAR-product.

4.3.4 Input variables

start_date: Starting date and time
end_date: Ending date and time
Accepted time format is python strings with following notations: YYYY-MM-DD, YYYY-MM-DD:hh, YYYY-MM-DD:hh-mm. ATTENTION: If just dates are used without hours or minutes, code will interpret them just as the first minute of the day.

lat_1: Upper latitude. Accepted range: [-90, 90]
lat_2: Lower latitude. Accepted range: [-90, 90]
lon_1: Accept range: [-180, 180]
lon_2: Accept range: [-180, 180]
Longitudes can be provided in two ways with the ranges. Either lon_1 $<$ lon_2 or lon_1 $>$ lon_2 which rotates the square.

Area of interest is given with four coordinate variables. Coordinates creates a four sided square that is used to indetify the area of interest. Coordinates follow the pattern presented in figure 1.

variable_name: Takes DARDAR_CLOUD variable name that user is interested in. Output of function will vary depending if chosen variable contains a vertical profile or not. In table 3 are listed retrieved variables of DARDAR product.

vertical_profile: User needs to provide their own vertical profile which the variable is converted to using weighted average. Vertical profile needs to be provided as a numpy array. Function handles both ascending and descending orders by converting them to ascending. Output also obeys this rule.

In the case that the users vertical_profile is smaller than the DARDAR products vertical profile following options determine the behavior of the averaging. This case is presented in figure 2 left.

> **exclude_top_on:** (default=True) Determines the behavior on top of user profile, figure 2 left a. If set True the part a will not be calculated in to the average of last height. If set False a will be included in averaging to the last layer of user profile H_6.
>
> **exclude_bottom_on:** (default=True) Determines the behavior on bottom of user profile figure 2 left b. If set True the part b will not be calculated in to

the average of the bottom height. If set False b will be included in averaging to the bottom layer of user profile H_0.

In the case that users profile is larger than the DARDAR products vertical profile, figure 2 right, the values that exceed this profile is set to zeros, in figure c and d.

flight_separation_time: (default=1) Determines the amount of time in data that will separate the data to different flights. This information will be combined with flight_separation_unit to perform the separation.
flight_separation_unit: (default="m") Determines the unit of time used with combination with flight_separation_time to separate data to different flights. Possible time units are "D"=day, "h"=hour, "m"=minute, "s"=second, and "ms"=millisecond.

colormap: (default="viridis") with this variable you can change the colormap. This package uses matplotlib so colormaps defined there can be used.

verbose: (default=1) Function has 3 levels of printing (0, 1, 2). 0=No printing except errors and warnings. 1=Limited. 2=Extensive.

auto_load_missing_files: (default=False) If True code automatically downloads all missing data for the user chosen spatio temporal location. Files will be downloaded to raw_data_loc directory. User needs to provide the key_location variable for this to work if the setup files are not in working directory.

version: (default="V30") is for choosing product version to use. By default the chosen product is "V30". The table 2 show the supported products by this software, time range, and some additional information.

raw_data_loc: (default=None) User can specify the directory containing downloaded DARDAR_CLOUD data. Code tries by default to find it from the working directory.

key_location: (default="") To download files password needs to be provided to get access to the servers. This is done by providing the directory location of the setup files created with setup() function. This option is needed for the auto_load_missing_files to work. By default the location is assumed to be the current working directory.

4.3.5 Example

An example of using this function could be that user wants to compare their own profile, that is defined with pythons numpy pakage to variable prof, to the original DARDAR standard profile. User is interested of Retrieved Ice Water Content so variable ”iwc” is used. Spatiotemporal location information of the region of interest is given and it is latitudes 36° and 45°, longitudes 6° and 12°, and time interval 2013-12-1 to end of 2013-12-2. In the code date 2013-12-3 is used as end time because this is the first minute of the next day. User has modified the flight separation time to 10 minutes from the default that is 5 minutes. But in this case there is only one overpass over this time period. Example of the function output is shown in figure 4.

```
import dardar_tool as dt
import numpy as np

prof = np.arange(0, 25000, 1000)

dt.extract_compare("2013-12-01", "2013-12-03",
                   45, 36, 6, 12,
                   "iwc", prof,
                   flight_separation_time=10)
```

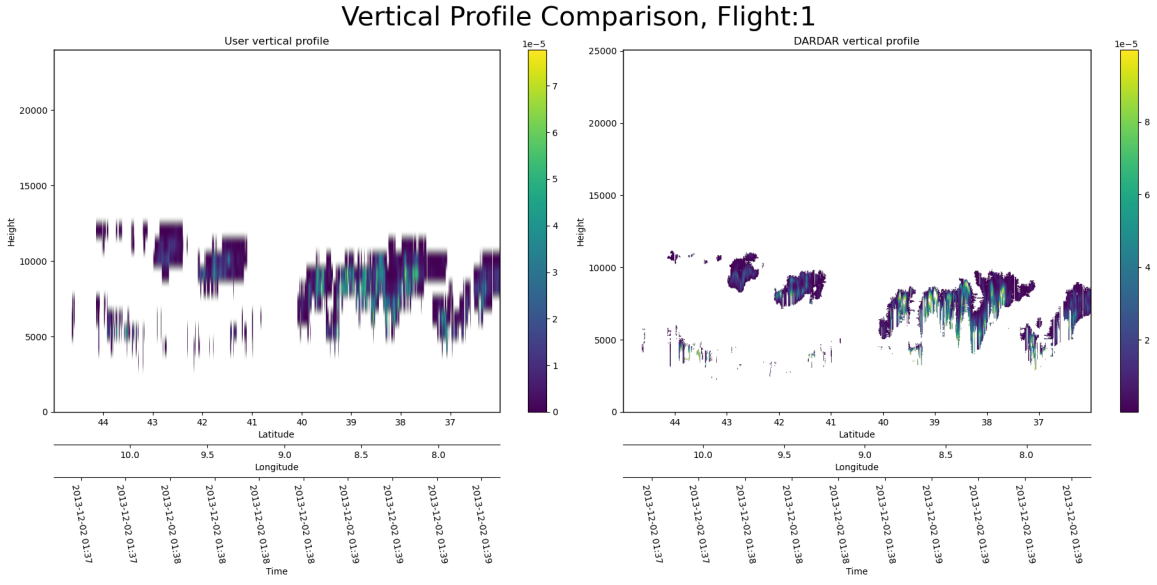

Figure 4: Example of the comparison plot. On the left is the averaged profile of the user and on the right the standard DARDAR product profile.

4.4 check_overpass

4.4.1 Description

This function can be used to check if there is some data that may contain relevant information to the spatiotemporal area of interest.

4.4.2 Function

```
check_overpass(start_date, end_date, lat_1, lat_2, lon_1, lon_2,
               version="V30", plotting_on=True)
```

4.4.3 Output

If plotting is enabled this function will plot all the overpasses of spatiotemporal area of interest and give also some text information in terminal/consol. In figure 5 is an example of one overpass plot. Satellites trajectory information is in $30° \times 30°$ grid, so the exact location of satellite is not known.

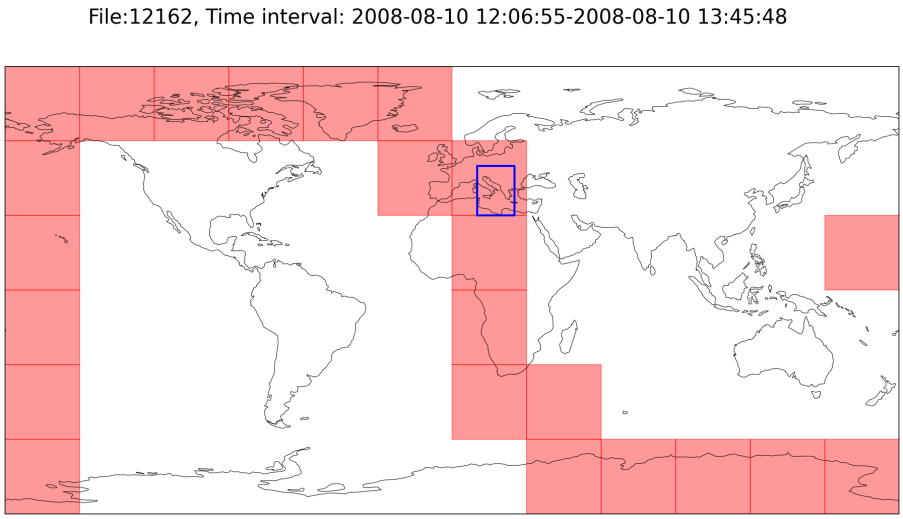

Figure 5: Example image of an overpass plot. On red is the product trajectory and on blue is the users selected spatiotemporal area of interest.

4.4.4 Input variables

start_date: Starting date and time
end_date: Ending date and time
Accepted time format is python strings with following notations: YYYY-MM-DD, YYYY-MM-DD:hh, YYYY-MM-DD:hh-mm. ATTENTION: If just dates are used without hours

or minutes code will interpret them just as first minute of day.

lat_1: Upper latitude. Accepted range: [-90, 90]
lat_2: Lower latitude. Accepted range: [-90, 90]
lon_1: Accept range: [-180, 180]
lon_2: Accept range: [-180, 180]
Longitudes can be provided in two ways with the ranges. Either lon_1 < lon_2 or lon_1 > lon_2 which rotates the square.

Area of interest is given with four coordinate variables. Coordinates creates a four sided square that is used to indetify the area of interest. Coordinates follow the pattern presented in figure 1.

version: (default="V30") is for choosing product version to use. By default the chosen product is "V30". The table 2 show the supported products by this software, time range, and some additional information.

plotting_on: (default=True) This option will display all the overpasses in a graphic form by default. It will also print information to terminal/console. If set False only printed information will be shown.

4.4.5 Example

An example of using this function could be that user wants to quickly check if there is a possibility that data exists to an given spatiotemporal location of latitudes 36° and 45°, longitudes 6° and 12°, and time interval 2013-12-1 to end of 2013-12-2. In the code date 2013-12-3 is used as end time because this is the first minute of the next day. Outputs of the function are presented in figures 6, if plotting is enables, and 7, the terminal/consol print that is always shown.

```
import dardar_tool as dt

dt.check_overpass("2013-12-01", "2013-12-03", 45, 36, 6, 12)
```

Figure 6: Plotted overflights of the example if plotting is enabled.

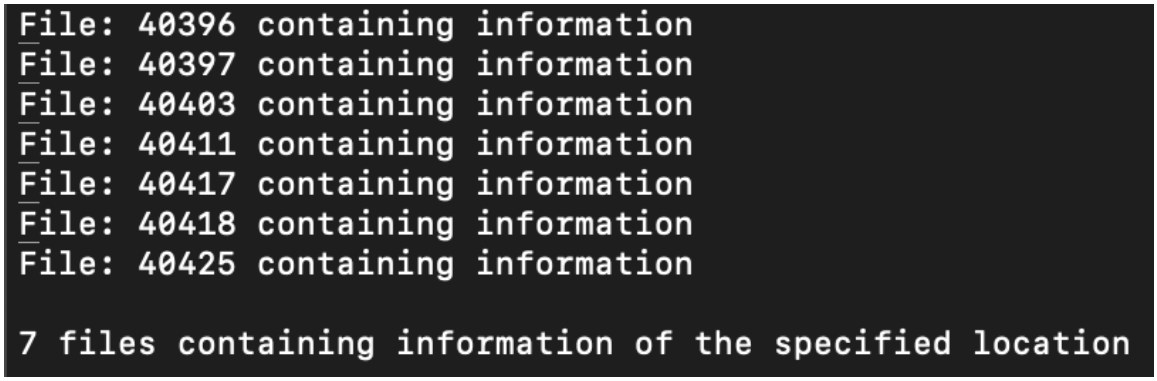

Figure 7: Example of the terminal/consol print given by the function.

4.5 read_data

4.5.1 Description

This function reads the saved results created with extractor function to python variables.

4.5.2 Function

```
time_array, latitude_array, longitude_array,
vertical_profile, variable
=
read_data(filename)
```

4.5.3 Output

Function will return time array, latitude array, longitude array, vertical profile, and variable that is created with extractor function.

4.5.4 Input variables

filename: Location and name of the saved file. By default working directory is assumed as location.

4.5.5 Example

An example of using this function could be that user wants to read data file named results_name_example, that has been created with extractor function. User uses only one variable for output, so the function will return a tuple containing all the arrays.

```
import dardar_tool as dt
dt.extractor("2013-12-01", "2013-12-03", 45, 36, 6, 12, "iwc",
             save_results=True,
             save_results_name="results_name_example")

results = dt.read_data("results_name_example.nc")
```

4.6 plot_saved_data

4.6.1 Description

Function plots saved netcdf data that is created with extractor function. Given plots contain spatiotemporal map and vertical profile of the flight. It also separates overpasses based on the given flight separation parameters.

4.6.2 Function

```
plot_saved_data(filename, zoom_map_in_out=5,
                flight_separation_time=1, flight_separation_unit="m",
                colormap="viridis")
```

4.6.3 Output

Function returns figures of separated flights from saved data. Figures 8 contain a flight path on left and vertical profile on right.

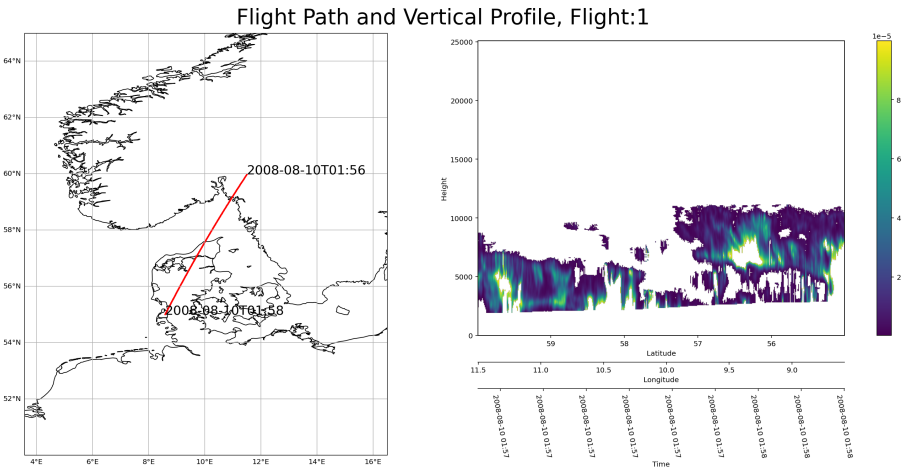

Figure 8: Example figure of an flight showing flight path on left and vertical profile on right.

4.6.4 Input variables

filename: Location and name of the saved file.

zoom_map_in_out: (default=5) Zooms flight plot out by given amount of degrees.

flight_separation_time: (default=1) Determines the amount of time in data that will

separate the data to different flights. This information will be combined with flight_separation_unit to perform the separation.

flight_separation_unit: (default="m") Determines the unit of time used with combination with flight_separation_time to separate data to different flights. Possible time units are "D"=day, "h"=hour, "m"=minute, "s"=second, and "ms"=millisecond.

colormap: (default="viridis") with this variable you can change the colormap. This package uses matplotlib so colormaps defined there can be used.

4.6.5 Example

An example of using this function could be that user wants to plot data from a saved file named results_name_example, that has been created with extractor function. The given output of function is a plot that is presented in figure 9.

```
import dardar_tool as dt

dt.extractor("2013-12-01", "2013-12-03", 45, 36, 6, 12, "iwc",
             save_results=True,
             save_results_name="results_name_example")

dt.plot_saved_data("results_name_example")
```

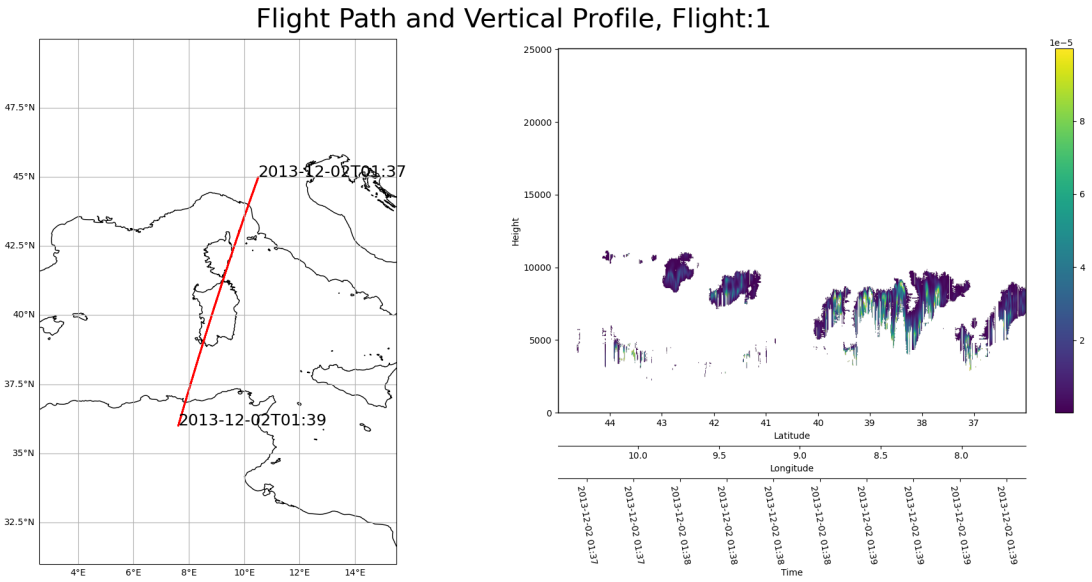

Figure 9: Output example of the plot saved data function.

4.7 plot_local_data

4.7.1 Description

Function plots python variables given by extractor function and plots it with spatiotemporal map and vertical profile of the flight. It also separates overpasses based on the given flight separation parameters.

4.7.2 Function

```
plot_local_data(time_array, latitude_array, longitude_array,
                vertical_profile, variable, zoom_map_in_out=5,
                flight_separation_time=1, flight_separation_unit="m",
                colormap="viridis")
```

4.7.3 Output

Function outputs an plot with spatiotemporal location on map shown in figure 10 on left and vertical profile of flight on right.

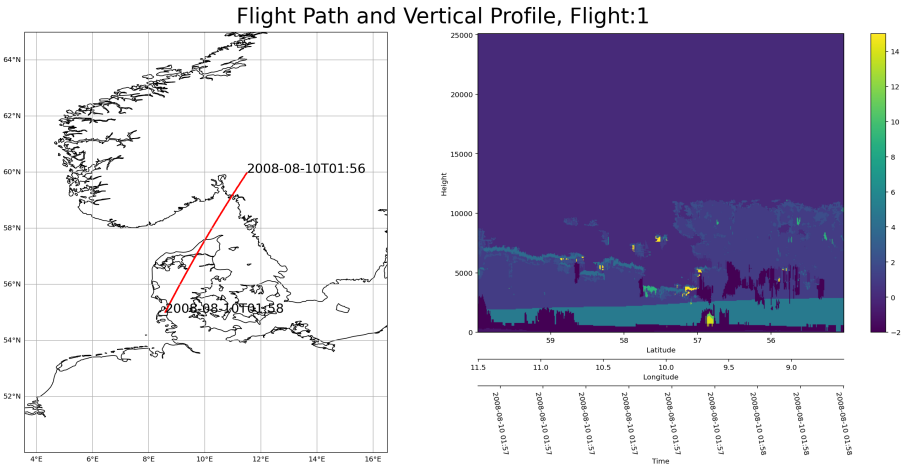

Figure 10: Example plot given by function for one overpass.

4.7.4 Input variables

time_array: Time array returned by extractor or loaded from netcdf file.

latitude_array: Latitude array returned by extractor or loaded from netcdf file.

longitude_array: Longitude array returned by extractor or loaded from netcdf file.

vertical_profile: Vertical profile returned by extractor or loaded from netcdf file.

variable: Variable array returned by extractor or loaded from netcdf file.

zoom_map_in_out: (default=5) Zooms flight plot out by given amount of degrees.

flight_separation_time: (default=1) Determines the amount of time in data that will separate the data to different flights. This information will be combined with flight_separation_unit to perform the separation.
flight_separation_unit: (default="m") Determines the unit of time used with combination with flight_separation_time to separate data to different flights. Possible time units are "D"=day, "h"=hour, "m"=minute, "s"=second, and "ms"=millisecond.

colormap: (default="viridis") with this variable you can change the colormap. This package uses matplotlib so colormaps defined there can be used.

4.7.5 Example

An example of using this function could be that user wants to plot data from a saved file name results_name_example.nc, or variables created by extractor function. I this case results_name_example.nc contains same the completely same data as what is produced by the extractor function. The given output is same for both functions and is presented in figure 11.

```
import dardar_tool as dt

tim_arr1, lat_arr1, lon_arr1, vert_prof1, var1 =
            dt.read_data("results_name_example.nc")
tim_arr2, lat_arr2, lon_arr2, vert_prof2, var2 =
            dt.extractor("2013-12-01",
                "2013-12-03", 45, 36, 6, 12,
                "iwc", raw_data_loc="./exaple_dir")

dt.plot_local_data(tim_arr1, lat_ar1, lon_arr1, vert_prof1, var1)
dt.plot_local_data(tim_arr2, lat_ar2, lon_arr2, vert_prof2, var2)
```

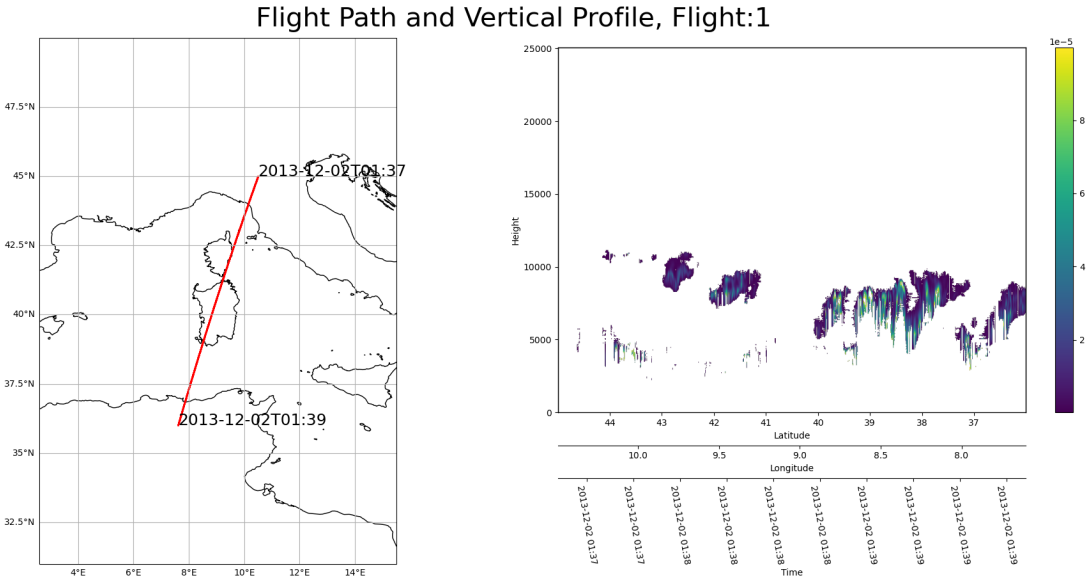

Figure 11: Output example of the plot local data function.

4.8 compare_user_data

4.8.1 Description

With this function user can compare their own data with the DARDAR-product. Users data needs to be in the same spatiotemporal resolution as given DARDAR data.

4.8.2 Function

```
compare_user_data(tim_array, lat_array, lon_array,
                  vertical_profile, user_variable, variable,
                  zoom_map_in_out=5,
                  flight_separation_time=1, flight_separation_unit="m",
                  colormap="viridis")
```

4.8.3 Output

Function outputs an trajectory image with cumulative sum of the vertical column values compared to DARDAR-product values on figure 12 left. On the right there is a vertical profile comparison. **Note:** Difference in these comparisons is calculated: difference = user_variable - DARDAR_variable.

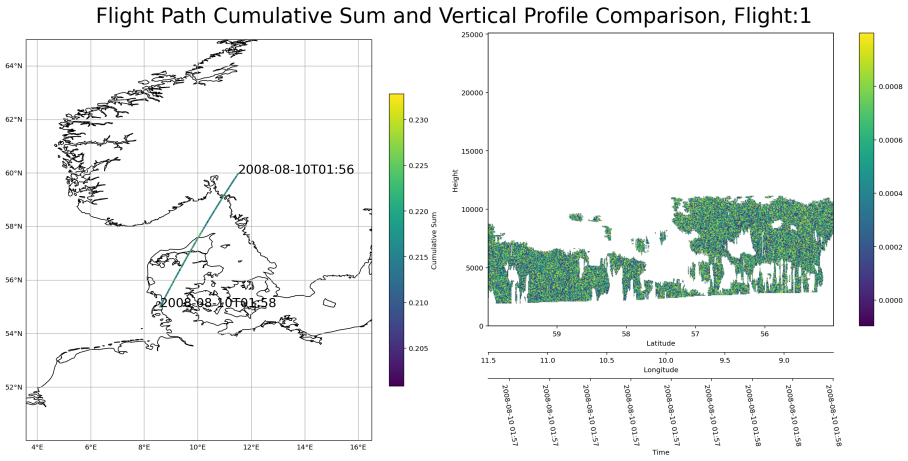

Figure 12: Example of users data compared to DARDAR-product. User data is in this case white noise, that can be observed from figures.

4.8.4 Input variables

time_array: Time array returned by extractor or loaded from netcdf file.

latitude_array: Latitude array returned by extractor or loaded from netcdf file.

longitude_array: Longitude array returned by extractor or loaded from netcdf file.

vertical_profile: Vertical profile returned by extractor or loaded from netcdf file.

user_variable: Users variable to be compared with variable from DARDAR-product. **Note:** Array axis dimensions need to be same with the variable. In DARDAR data first axis is the spatiotemporal location (time, latitude, longitude) and the second axis is the profile size.

variable: Variable array returned by extractor or loaded from netcdf file.

zoom_map_in_out: (default=5) Zooms flight plot out by given amount of degrees.

flight_separation_time: (default=1) Determines the amount of time in data that will separate the data to different flights. This information will be combined with flight_separation_unit to perform the separation.
flight_separation_unit: (default="m") Determines the unit of time used with combination with flight_separation_time to separate data to different flights. Possible time units are "D"=day, "h"=hour, "m"=minute, "s"=second, and "ms"=millisecond.

colormap: (default="viridis") with this variable you can change the colormap. This package uses matplotlib so colormaps defined there can be used.

4.8.5 Example

An example of using this function could be that user wants to compare their own model data against DARDAR-product. In this example user has loaded their data to the users_own_data_variable, which in this case is users_own_data_variable = var + 1. Because of this the result plot show constant values for cumulative sum and vertica profile values shown in figure 13.

```
import dardar_tool as dt

tim_arr, lat_arr, lon_arr, vert_prof, var =
                dt.extractor("2013-12-01",
                    "2013-12-03", 45, 36, 6, 12,
```

```
        "iwc", raw_data_loc="./exaple_dir")

users_own_data_variable = load_own_data()

dt.compare_user_data(tim_arr, lat_arr, lon_arr, vert_prof,
        var, users_own_data_variable)
```

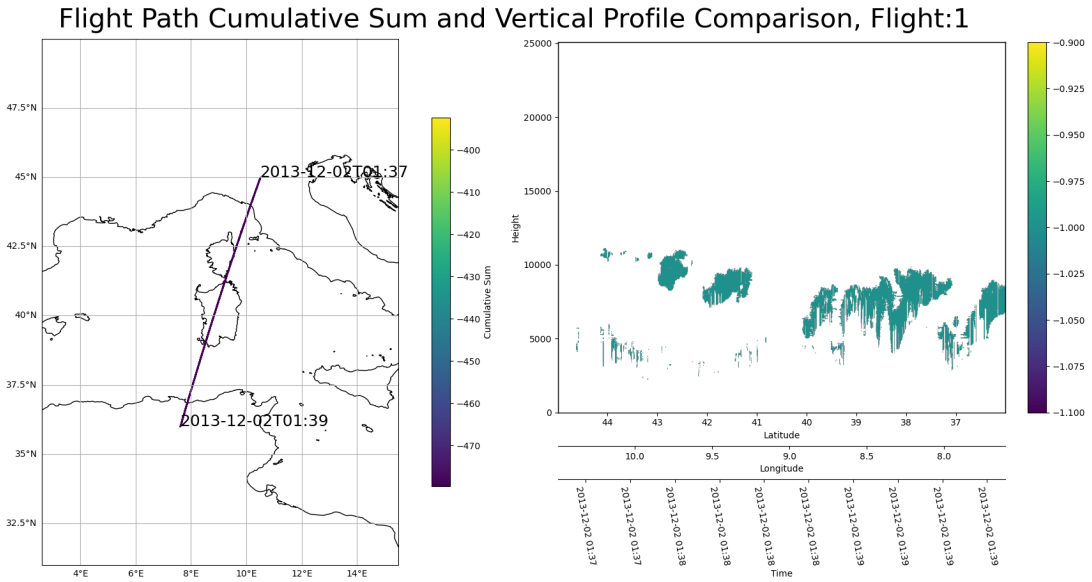

Figure 13: Output example of comparison plot.

4.9 column_cumsum

4.9.1 Description

Calculates cumulative column sum of variable values.

4.9.2 Function

```
cumulative_variable_sums
=
column_cumsum(profile_values)
```

4.9.3 Output

Returns cumulative column sums of provided variable.

4.9.4 Input variables

variable_name: Takes DARDAR_CLOUD variable name that user is interested in. Output of function will vary depending if chosen variable contains a vertical profile or not. In table 3 are listed retrieved variables of DARDAR product.

4.9.5 Example

An example of using this function could be that user wants to see the cumulative sums of the columns along the trajectory and plot them.

```
import dardar_tool as dt

tim_arr, lat_arr, lon_arr, vert_prof, var =
            dt.extractor("2013-12-01",
                "2013-12-03", 45, 36, 6, 12,
                "iwc", raw_data_loc="./exaple_dir")

cumulative_sum_of_columns = dt.column_cumsum(var)

import matplotlib.pyplot as plt

plt.plot(time_arr, cumulative_sum_of_columns)
```

```
plt.xlabel("Time")
plt.ylabel("Cumulative sum of iwc")
plt.show()
```

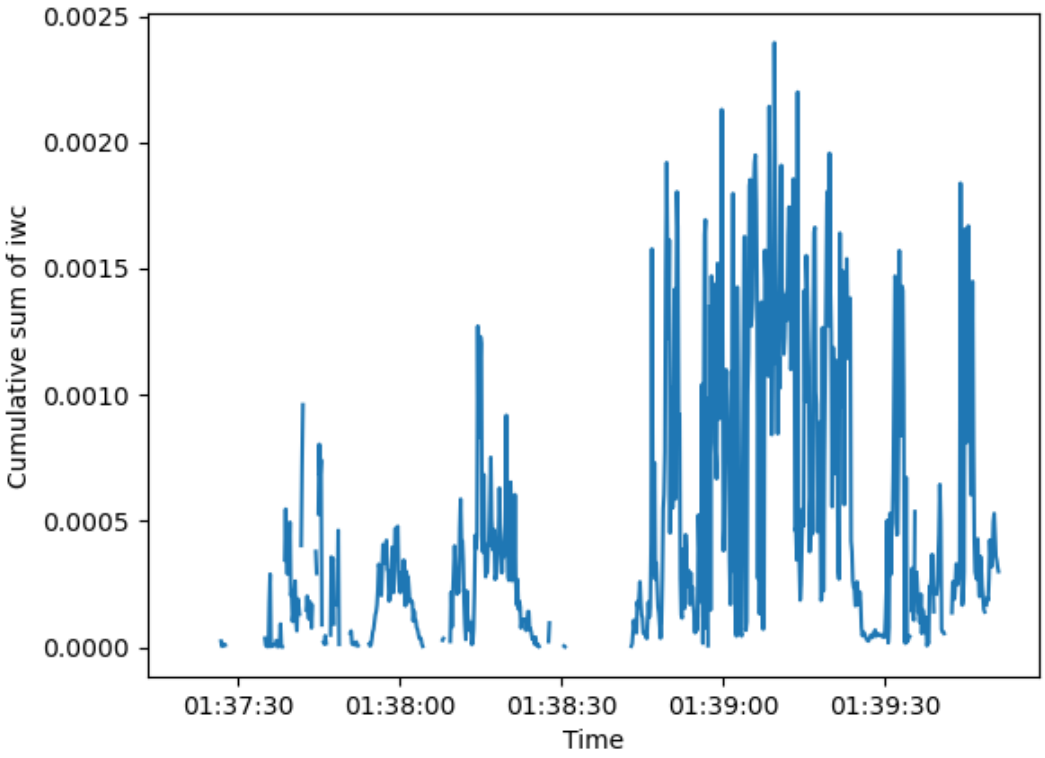

Figure 14: Plot given by the code example.

4.10 scale_resolution

4.10.1 Description

Scales spatiotemporal resolution of data by user provided constants and averages data to this resolution.

4.10.2 Function

```
time_scaled, latitude_scaled, longitude_scaled,
vertical_profile_scaled, variable_scaled
=
scale_resolution(time_array, latitude_array, longitude_array,
                 vertical_profile, variable,
                 latitude_resolution=None, longitude_resolution=None,
                 height_resolution=None, time_resolution=None)
```

4.10.3 Output

Returns time, latitude, longitude, vertical profile, and variable arrays in new scaled resolution.

4.10.4 Input variables

time_array: Time array returned by extractor or loaded from netcdf file.

latitude_array: Latitude array returned by extractor or loaded from netcdf file.

longitude_array: Longitude array returned by extractor or loaded from netcdf file.

vertical_profile: Vertical profile returned by extractor or loaded from netcdf file.

variable: Variable array returned by extractor or loaded from netcdf file.

latitude_resolution: (default=None) Latitude averaging constant in degrees. The constant determines the resolution by specifying the interval between consecutive values on the latitude scale. If None resolution will not be changed.

longitude_resolution: (default=None) Longitude averaging constant in degrees. The

constant determines the resolution by specifying the interval between consecutive values on the longitude scale. If None resolution will not be changed.

height_resolution: (default=None) Height averaging constant in meters. The constant determines the resolution by specifying the interval between consecutive values on the height scale. If None resolution will not be changed. **Note:** If user wants to use a custom profile, extractor provides this functionality.

time_resolution: (default=None) Time averaging constant give in numpy python package np.timedelta64() format. The constant determines the resolution by specifying the interval between consecutive values on the time scale. If None resolution will not be changed.

4.10.5 Example

An example of using this function could be that user wants to scale their data in latitudes with 0.25° and in time to 1s resolution, other variable can stay in original DARDAR format. This can be achieved setting latitude_resolution=0.25°, latitude_resolution and height_resolution to None, and time_resolution=np.timedelta64(1, "s"). After the function call both original and scaled data are plotted and are shown in figure 15. **Note:** In this case scaled data image is fully blue because the mask is removed from it when scaling is applied.

```
import dardar_tool as dt

tim_arr, lat_arr, lon_arr, prof, var =
            dt.extractor("2013-12-01",
                "2013-12-03", 45, 36, 6, 12,
                "iwc", raw_data_loc="./exaple_dir")

time_scaler = np.timedelta64(1, "s")

tim_sc, lat_sc, lon_sc, prof_sc, var_sc =
                dt.scale_resolution(tim_arr, lat_arr, lon_arr,
                prof, var,
                0.25, None, None, time_scaler)
```

```
dt.plot_local_data(tim_arr, lat_arr, lon_arr, prof, var)
dt.plot_local_data(tim_sc, lat_sc, lon_sc, prof_sc, var_sc)
```

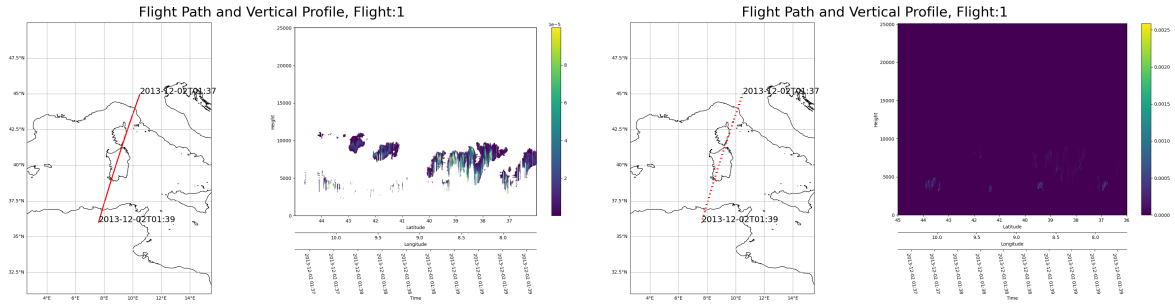

Figure 15: Example profiles from the code. On the left original profile and on the right the scaled profile.