Environmental Library Documentation

Release V0.0.1

DLR, TUHH, TUD, UC3M

Apr 15, 2022

CONTENTS:

CHAPTER ONE

WHAT IS ENVLIB?

The Python Library EnVLiB is a software package developed by UC3M and DLR. The main idea of EnVLiB is to provide an open-source, easy-to-use, and flexible software tool that efficiently calculates the spatial and temporal resolved climate impact of aviation emissions by using algorithmic climate change functions (aCCFs). Both individual aCCFs of water vapour, NOx-induced ozone and methane, and contrail-cirrus and also merged non-CO2 aCCFs that combine the individual aCCFs can be calculated.

EnVLib is released under XXX Licence. Citation of the ENVLiB connected software documentation paper is kindly requested upon use, with software DOI for EnVLiB (doi:XXX) and version number:

Citation info: Dietmüller, S. Matthes, S., Dahlmann, K., Yamashita, H., Simorgh, A., Soler, M., Linke, F., Lührs, B., Meuser, M., Weder, C., Yin, F., Castino, F., Gerw, V. (2022): A python library for computing individual and merged non-CO2 algorithmic climate change functions, GMD.

HOW TO RUN THE LIBRARY

0. it is highly recomended to create a virtual environment with Python version 3.8:

```
conda create -n name_env python=3.8
conda activate name_env
pip3 install setuptools~=49.6.0
pip3 install pint~=0.19.1
```

1. Clone or download the repository.
2. Locate yourself in the envlib (library folder) path, and run the following line, using terminal (MacOS) or cmd (Windows), which will install all dependencies:

```
python setup.py install
```

CHAPTER THREE

HOW TO USE IT

1. import library:

```
import envlib
from envlib.main_processing import ClimateImpact
```

2. Specify the directories for datasets containing variables on pressure levels and surface in a dictioary as:

```
path = {}
path['path_pl']  = dir_pressure_variables  # dircetory of the dataset containing pressure␣
↪level variables
path['path_sur'] =  dir_surface_variables  # dircetory of the dataset containing surface␣
↪variables
```

3. Specify a directory for the output files:

```
path_save = dir_results    # dircetory to save the output data
```

4. (Optional) Set the preferred configurations in a dictionary (Defult configurations have been defined in the library.)”

```
confg = {}

# If true, it includes efficacies according to Lee et al. (2021)
confg['efficacy'] = True                  # Options: True, False

# Specifies the emission scenario. Currently, pulse and future emission scenarios have␣
↪been implemented
confg['emission_scenario'] = 'future_scenario'       # Options: pulse, future_scenario

# Specifies the climate indicator. Currently, Average Temperature Response has been␣
↪implemented
confg['climate_indicator'] = 'ATR'         # Options: ATR

# Specifies the time horizon over which the metric is calculated
confg['TimeHorizon'] = 20                  # Options: 20, 50, 100

# Specifies the threshold of ice-supersaturated regions
confg['rhi_threshold'] = 0.90               # Options: Depends on the resolution of data␣
↪(0.90, 95, 0.99, 1.0), e.g., in case of ERA5_HRES it is 0.9

"""Output Options"""

```

(continues on next page)

(continued from previous page)

```
# If true, all individual aCCFs converted to K/kg(fuel)
confg['unit_K/kg(fuel)'] = False           # Options: True, False

# If true,  PMO effect included to CH4 aCCF and total NOx aCCF
confg['PMO'] = True                        # Options: True, False

# If true, merged aCCF is calculated
confg['merged'] = True                     # Options: True, False

# NOx and inverse EIs
confg['NOx&inverse_EIs'] = 'TTV'   # Options: 'TTV (typical transantlantic fleet mean␣
↪values)', 'ac_dependent'

# If Confg['NOx&inverse_EIs'] = 'ac_dependent', aircraft type needs to be selected
confg['ac_type'] = 'wide-body'             # Options: 'regional', 'single-aisle', 'wide-
↪body'

# If true, NOx aCCF is calculated (i.e. aCCF-NOx = aCCF-CH4 + aCCF-O3)
confg['NOx_aCCF'] = False                   # Options: True, False

"""Climate Hotspots"""

# If true, climate hotspots are calculated'
confg['Chotspots'] = False                 # Options: True, False

# If true, it assigns binary values to climate hotspots (i.e., 0 for areas with climate␣
↪impacts below the specified threshold, and 1 for areas with higher climate impacts than␣
↪the threshold)
confg['hotspots_binary'] = False            # Options: True, False

# Specifies the constant threshold for determining climate hotspots
confg['hotspots_thr'] = False

# Determines dynamically the threshold for identifying climate hotspots using the␣
↪cumulative distribution of the merged aCCF. The percentiles are also outputted in␣
↪netCDF output file
confg['hotspots_percentile'] = 99         # Options: percentage < 100

""" Statistical analysis of EPS forecast"""

# If true, mean values of aCCFs and variables are saved in the netCDF output file
confg['mean'] = False                     # Options: True, False

# If true, standard deviation of aCCFs and variables are saved in the netCDF output file
confg['std'] = False                      # Options: True, False

""" Output """

# If true, weather variables are saved in the netCDF output file
confg['MET_variables'] = False            # Options: True, False

# If true, polygons containing climate hotspots will be saved in the GeoJson file
confg['geojson'] = False                  # Options: True, False

# Specifies the color of polygons
confg['color'] = 'copper'                 # Options: colors of cmap, e.g., copper, jet,␣
↪Reds
```

(continues on next page)

(continued from previous page)

5. Process inputted data:

```
CI = ClimateImpact(path, horizontal_resolution=resolution, lat_bound=(lat_min, lat_max),␣
↪lon_bound=(lon_min, lon_max), save_path=path_save)
```

6. Calculate aCCFs with respect to the defined settings in the dictionary (i.e., Confg) and store the results in a netCDF file:

```
CI.calculate_accfs(**confg)
```

CHAPTER

FOUR

AN EXAMPLE

0. Here is an example how one can use sample data in test directory of envlib to generate output for a set of user-difned configurations:

```python
import envlib
from envlib.main_processing import ClimateImpact

path_here = 'envlib/'
test_path = path_here + '/test/sample_data/'
path_ = {'path_pl': test_path + 'sample_pl.nc', 'path_sur': test_path + 'sample_sur.nc'}
path_save = test_path + 'env_processed.nc'

confg = {}
confg['efficacy'] = True
confg['emission_scenario'] = 'future_scenario'
confg['climate_indicator'] = 'ATR'
confg['TimeHorizon'] = 20
confg['rhi_threshold'] = 0.90

"""Output Options"""
confg['unit_K/kg(fuel)'] = False
confg['PMO'] = True
confg['merged'] = True
confg['NOx&inverse_EIs'] = 'ac_dependent'
confg['ac_type'] = 'wide-body'
confg['NOx_aCCF'] = False

"""Climate Hotspots"""
confg['Chotspots'] = True
confg['hotspots_binary'] = True
confg['hotspots_thr'] = False
confg['hotspots_percentile'] = 99

""" Output """
confg['MET_variables'] = True
confg['geojson'] = True
confg['color'] = 'copper'


CI = ClimateImpact(path_, horizontal_resolution=0.75, lat_bound=(35, 60.0), lon_bound=(-
↪15, 35), save_path=path_save)
CI.calculate_accfs(**confg)
```

CHAPTER FIVE

HOW TO COMPILE DOCUMENTATION PDF?

You can use the Makefile created by Sphinx to create your documentation. Locate yourself in the doc path.

First clean the _build directory to avoid error or legacy information. Just call:

```
make clean
```

In case you want to build your documentation in latex call **twice**:

```
make latexpdf
```

if you want to do build your in html call:

```
make html
```

Note that you **should not see** any error or warning, this information appears as red text in the terminal.

CHAPTER SIX

CONTENTS:

6.1 Data Processing

`envlib.extract_data.`**`extract_coordinates`**(*ds*, *ex_variables*, *ds_sur=None*)

Extract coordinates (axes) in the dataset defined with different possible names.

Parameters

- **`ds_sur`** –
- **`ds`** (`Dataset`) – Dataset openned with xarray.

Returns ex_var_name List of available coordinates.

Return type list

Returns variables Assigns bool to the axes (e.g., if ensemble members are not available, it sets False).

Return type dict

`envlib.extract_data.`**`extract_data_variables`**(*ds*, *ds_sr=None*, *verbose=False*)

Extract available required variables in the dataset defined with different possible names.

Parameters

- **`ds`** (`Dataset`) – Dataset openned with xarray.
- **`ds_sr`** (`Dataset`) – Dataset containing surface parameters openned with xarray.
- **`verbose`** (`bool`) – Used to show more information.

Returns ex_var_name Available required weather variables.

Return type list

Returns variables Assigns bool to the required wethear variables.

Return type dict

`envlib.extract_data.`**`logic_cal_accfs`**(*variables*)

Creates a dictionary containing logical values showing the possibility to calculate each aCCF.

Parameters **`variables`** (`dict`) – Variables available in the given dataset.

Returns dictionary containing logical values showing the possibility to calculate each aCCF.

Return type dict

`envlib.extend_dim.`**`extend_dimensions`**(*inf_coord*, *ds*, *ds_sur*, *ex_variables*)

Unifies the dimension of all types of given data as either 4-dimensional or 5-dimensional arrays, depending on the existence of ensemble members. e.g., if the data has only two fields: latitude and longitude, this function adds time and level fields.

Parameters

- **ds** (`Dataset`) – Information on original coordinates.
- **ds** – Dataset openned with xarray containing variables on pressure levels.
- **ds_sur** (`Dataset`) – Dataset containing surface parameters openned with xarray.
- **inf_coord** – new coordinates

Returns ds_pl new dataset of pressure level variables regarding the added coordinates

Return type dataset

Returns ds_surf new dataset of surface parameters regarding the added coordinates

Return type dataset

`envlib.processing_surf_vars.`**`extend_olr_pl_4d`**(*sur_var*, *pl_var*, *index*, *fore_step*)

Calculate outgoing longwave radiation (OLR) [W/m2] at TOA from the parameter, top net thermal radiation (ttr) [J/m2], and extend (duplicating) it to all pressure levels for consistency of dimensions. For a specific time, regarding the inputted index, OLR is calculated in 3D (i.e., level, latitude, longitude).

Parameters

- **sur_var** (`Dataset`) – Dataset containing surface parameters openned with xarray.
- **pl_var** (`Dataset`) – Dataset containing pressure level parameters openned with xarray.
- **index** (`int`) – Index of the time.
- **fore_step** (`int`) – Forecast step in hours.

Returns arr OLR with 3D dimensiones (i.e., level, latitude, longitude).

Return type array

`envlib.processing_surf_vars.`**`extend_olr_pl_5d`**(*sur_var*, *pl_var*, *index*, *fore_step*)

Calculate outgoing longwave radiation (OLR) [W/m2] at TOA from the parameter, top net thermal radiation (ttr) [J/m2], and extend (duplicating) it to all pressure levels for consistency of dimensions. For a specific time, regarding the inputted index, OLR is calculated in 4D (i.e., number, level, latitude, longitude).

Parameters

- **sur_var** (`Dataset`) – Dataset containing surface parameters openned with xarray.

- **pl_var** (Dataset) – Dataset containing pressure level parameters openned with xarray.
- **index** (int) – Index of the time that exist in the dataset of pressure level parameters at this step.
- **fore_step** (int) – Forecast step in hours.

Returns arr OLR with 4D dimensiones (i.e., number, level, latitude, longitude).

Return type array

`envlib.processing_surf_vars.`**`get_olr`**(*sur_var*, *pl_var*, *number=True*, *fore_step=None*)

Calculate outgoing longwave radiation (OLR) [W/m2] at TOA from the parameter, top net thermal radiation (ttr) [J/m2]. OLR is calculated in 5D or 4D depending on the existance of ensemble members.

Parameters

- **sur_var** (Dataset) – Dataset containing surface parameters openned with xarray.
- **pl_var** (int) – Dataset containing pressure level parameters openned with xarray.
- **number** (bool) – Determines whether the weather data contains ensemble members or not.
- **fore_step** – Forecast step in hours.

Returns arr OLR.

Return type numpy.ndarray

`envlib.processing_surf_vars.`**`get_olr_4d`**(*sur_var*, *pl_var*, *thr*, *fore_step=None*)

Calculate outgoing longwave radiation (OLR) [W/m2] at TOA from the parameter, top net thermal radiation (ttr) [J/m2]. OLR is calculated in 4D (i.e, time, level, latitude, longitude).

Parameters

- **sur_var** (Dataset) – Dataset containing surface parameters openned with xarray.
- **pl_var** (int) – Dataset containing pressure level parameters openned with xarray.
- **thr** (dict) – Thresholds to automatically determine forecast steps.
- **fore_step** – Forecast step in hours.

Returns arr OLR with 4D dimensiones (i.e., time, level, latitude, longitude).

Return type numpy.ndarray

`envlib.processing_surf_vars.`**`get_olr_5d`**(*sur_var*, *pl_var*, *thr*, *fore_step=None*)

Calculate outgoing longwave radiation (OLR) [W/m2] at TOA from the parameter, top net thermal radiation (ttr) [J/m2]. OLR is calculated in 5D (i.e, time, number, level, latitude, longitude).

Parameters

- **sur_var** (Dataset) – Dataset containing surface parameters openned with xarray.
- **pl_var** (int) – Dataset containing pressure level parameters openned with xarray.
- **thr** (dict) – Thresholds to automatically determine forecast steps.
- **fore_step** – Forecast step in hours.

Returns arr OLR with 5D dimensiones (i.e., time, number, level, latitude, longitude).

Return type numpy.ndarray

6.2 Weather Store

class envlib.weather_store.**WeatherStore**(*weather_data*, *weather_data_sur=None*, *flipud='auto'*, ***weather_config*)

Prepare the data required to calculate aCCFs and store them in a xarray dataset.

__init__(*weather_data*, *weather_data_sur=None*, *flipud='auto'*, ***weather_config*)

Processes the weather data.

Parameters

- **weather_data** – Dataset openned with xarray containing variables on different pressure levels.
- **weather_data_sur** – Dataset openned with xarray containing variables on single pressure level (i.e., outgoing longwave radiation in this case).

get_xarray()

Creates a new xarray dataset containing processed weather variables.

Returns ds xarray dataset containing user-defined variables (e.g., merged aCCFs, mean aCCFs, Climate hotspots).

Return type dataset

reduce_domain(*bounds*, *verbose=False*)

Reduces horizontal domain and time.

Parameters **bounds** – ranges defined as tuple (e.g., lat_bound=(35, 60.0)).

Return type dict

6.3 Calculation of aCCFs

class envlib.accf.**GeTaCCFs**(*wd_inf*, *rhi_thr*)

Calculation of algorithmic climate change functions (aCCFs).

__init__(*wd_inf*, *rhi_thr*)

Prepares the data required to calculate aCCFs and store them in self.

Parameters

- **wd_inf** (Class) – Contains processed weather data with all information.
- **rhi_thr** (float) – Threshold of relative humidity over ice for determining ice-supersaturation.

accf_ch4()

Calculates the aCCF of Methane for pulse emission scenario, average temperature response as climate indicator over next 20 years (P-ATR20-methane [K/kg(NO2)]). To calculate the aCCF of Methane, meteorological variables geopotential and incoming solar radiation are required.

Returns accf Algorithmic climate change function of methane.

Return type numpy.ndarray

accf_dcontrail()

Calculates the aCCF of day-time contrails for pulse emission scenario, average temperature response as climate indicator and 20 years (P-ATR20-contrails [K/km]). To calculate the aCCF of day-time contrails, meteorological variables ourgoing longwave radiation, temperature and relative humidities over ice and water are required. Notice that, temperature and relative humidies are required for the detemiation of presistent contrail formation areas.

Returns accf Algorithmic climate change function of day-time contrails.

Return type numpy.ndarray

accf_h2o()

Calculates the aCCF of water vapour for pulse emission scenario, average temperature response as climate indicator and 20 years (P-ATR20-water-vapour [K/kg(fuel)]). To calculate the aCCF of water vapour, meteorological variable potential vorticity is required.

Returns accf Algorithmic climate change function of water vapour.

Return type numpy.ndarray

accf_ncontrail()

Calculates the aCCF of night-time contrails for pulse emission scenario, average temperature response as climate indicator over next 20 years (P-ATR20-contrails [K/km]). To calculate the aCCF of nighttime contrails, meteorological variables temperature and relative humidities over ice and water are required. Notice that, relative humidies are required for the detemiation of presistent contrail formation areas.

Returns accf Algorithmic climate change function of nighttime contrails.

Return type numpy.ndarray

accf_o3()

Calculates the aCCF of Ozone for pulse emission scenario, average temperature response as climate indicator over next 20 years (P-ATR20-ozone [K/kg(NO2)]). To calculate the aCCF of Ozone, meteorological variables temperature and geopotential are required.

Returns accf Algorithmic climate change function of Ozone.

Return type numpy.ndarray

get_accfs(***problem_config*)

Calculates individual aCCFs, the merged aCCF and climate hotspots based on the defined conversions, parameters and etc.

get_std(*var*, *normalize=False*)

Calculates the standard deviation of the inputted variables over the ensemble members.

Parameters

- **var** – variable.
- **normalize** – If True, it calculates standard deviation over the normalized variable. If False, standard deviation is taken from the original variable.

Return type numpy.ndarray

Return type bool

Returns x_std standard deviation of the variable.

Return type numpy.ndarray

get_xarray()

Creates an xarray dataset containing user-selected variables.

Returns ds xarray dataset containing user-selected variables (e.g., merged aCCFs, mean aCCFs, Climate hotspots).

Return type dataset

:returns encoding :rtype: dict

envlib.accf.**convert_accf**(*name*, *value*, *confg*)

Converts aCCFs based on the selected configurations (i.e., efficacy, climate indicator, emission scenarios and time horizons).

Parameters

- **name** – Name of the species (e.g., 'CH4').
- **value** – Value of the species to be converted (P-ATR20 without efficacy factor).
- **confg** – User-defined configurations for conversions.

Return type string

Return type numpy.ndarray

Return type dict

Returns value Converted aCCF.

Return type numpy.ndarray

`envlib.accf.`**`get_Fin`**(*ds*, *lat*)

Calculates incoming solar radiation.

Parameters

- **`ds`** – dataset to extract the number of day.
- **`lat`** – latitude.

Return type Dataset

Return type numpy.ndarray

Returns Fin Incoming solar radiation.

Return type numpy.ndarray

6.4 Persistent Contrails Formation

`envlib.contrail.`**`get_cont_form_thr`**(*ds*, *member*)

Calculates the thresholds of temperature and relative humidity over water needed for determining the formation criteria of contrails.

Parameters

- **`ds`** (`Dataset`) – Dataset openned with xarray.
- **`member`** (`bool`) – Detemines the presense of ensemble forecasts in the given dataset.

Returns rcontr Thresholds of relative humidity over water.

Return type numpy.ndarray

Returns TLM_e Thresholds of temperature.

Return type numpy.ndarray

`envlib.contrail.`**`get_pcfa`**(*ds*, *member*, ***problem_config*)

Calculates the presistent contrail formation areas (pcfa) which is used to calculate aCCF of (day/night) contrails.

Parameters

- **`ds`** (`Dataset`) – Dataset openned with xarray.
- **`member`** (`bool`) – Detemines the presense of ensemble forecasts in the given dataset.

Returns pcfa Presistent contrail formation areas.

Return type numpy.ndarray

`envlib.contrail.`**`get_relative_hum`**(*ds*, *member*, *intrp=True*)

Calculates the relative humidities over ice and water from the provided relative humidity within ECMWF dataset. In ECMWF data, Relative humidity is defined with respect to saturation of the mixed phase: i.e. with respect to saturation over ice below -23C and with respect to saturation over water above 0C. In the regime in between a quadratic interpolation is applied.

Parameters

- **`ds`** (`Dataset`) – Dataset openned with xarray.
- **`member`** (`bool`) – Detemines the presense of ensemble forecasts in the given dataset.

Returns rcontr Thresholds of relative humidity over water.

Return type numpy.ndarray

Returns TLM_e Thresholds of temperature.

Return type numpy.ndarray

`envlib.contrail.`**`get_rw_from_specific_hum`**(*ds*, *member*)

Calculates relative humidity over water from specific humidity.

Parameters

- **`ds`** (`Dataset`) – Dataset openned with xarray.
- **`member`** (`bool`) – Detemines the presense of ensemble forecasts in the given dataset.

Returns r_w Relative humidity over water.

Return type numpy.ndarray

6.5 Calculation of Alternative Variables

`envlib.calc_altrv_vars.`**`get_pvu`**(*ds*)

Caclulates potential vorticity from meteorological variables temperature and components of wind.

Parameters **`ds`** (`Dataset`) – Dataset openned with xarray.

Returns PVUU potential vorticity unit

Return type numpy.ndarray

`envlib.calc_altrv_vars.`**`get_rh_ice`**(*ds*)

Calculates relative humidity over ice from realtive humidity over water

Parameters **`ds`** (`Dataset`) – Dataset openned with xarray.

Returns rh_ice relative humidity over ice

Return type numpy.ndarray

`envlib.calc_altrv_vars.`**`get_rh_sd`**(*ds*)

Calculates the relative humidity from specific humidity

Parameters **`ds`** (`Dataset`) – Dataset openned with xarray.

Returns rh_sd relative humidity

Return type numpy.ndarray

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

7.1 Acknowledmgements

*This library has been developed within EU-Projects, **FlyATM4E** and **ALARM**.*

- ***FlyATM4E** has received funding from the SESAR Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement No 891317. The JU receives support from the European Union's Horizon 2020 research and innovation programme and the SESAR JU members other than the Union.*
- ***ALARM** has received funding from the SESAR Joint Undertaking (JU) under grant agreement No 891467. The JU receives support from the European Union's Horizon 2020 research and innovation programme and the SESAR JU members other than the Union.*

PYTHON MODULE INDEX