Environmental Library Documentation

Release V0.0.1

DLR, TUHH, TUD, UC3M

Nov 24, 2021

CONTENTS:

CHAPTER

ONE

WHAT IS ENVLIB?

ENVLIB is a libray that calculates algorithmic climate change functions (aCCFs). It is distributed under the GNU Lesser General Public License v3.0.

Citation info: A python library for computing merged non-CO2 algorithmic climate change functions by Simone et al..

CHAPTER

TWO

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
```

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 import main_processing

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

```
path = {'path_pl': location_pressure_variables, 'path_sur': location_surface_variables'}
```

3. Specify the directory for the output file:

:: path_save = location_results

4. **Set the preferred configurations in a dictionary:** Set the preferred configurations in a dictionary. Settings are:

   - **efficacy**: True, False
   - **emission_scenario**: sustained, future_scenario, pulse
   - **limate_indicator**: ATR, GWP
   - **TimeHorizon**: 20, 50, 100
   - **PMO**: True, False
   - **merged**: True, False
   - **NOx**: True, False
   - **Chotspots**: True, False
   - **hotspots_thr**: constant, varying
   - **binary**: True, False
   - **variables**: True, False
   - **mean**: True, False
   - **std**: True, False

```
confg = {'efficacy': False, 'emission_scenario': 'pulse', 'climate_indicator': 'ATR',
↪'TimeHorizon': 20,
         'PMO': False,'merged': True, 'NOx': True, 'Chotspots': True, 'binary': True,
         'hotspots_thr': 1e-13, 'variables': False, 'mean': False, 'std': False}
```

5. Process inputted data:

```
CI = processing_main.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, 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:

```
import envlib
from envlib import main_processing

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 = {'efficacy': False, 'emission_scenario': 'pulse', 'climate_indicator': 'ATR',
↪'TimeHorizon': 20,
             'PMO': False,
             'merged': True, 'NOx': True, 'Chotspots': True, 'binary': True,
             'hotspots_thr': 1e-13, 'variables': False, 'mean': False, 'std': False}

CI = main_processing.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

MODULES:

6.1 envlib

6.1.1 envlib package

Submodules

envlib.accf module

class envlib.accf.**CalAccf**(*wd_inf*)

Bases: object

Calculation of algorithmic climate change functions (aCCFs).

accf_ch4()

Calculates the aCCF of Methane for pulse emission scenario, average temperature response as climate indicator and 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 contrial 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 and 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 contrial 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 and 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*)
Gets the formulations of aCCFs, and calculated user-defined conversions or functions.

get_std(*var*, *normalize=False*)
Calculates standard deviation of a variable over ensemble members.

Parameters

- **var** – variable.
- **normalize** – If True, it calculated standard deviation over the normalized variable, if False,

Return type numpy.ndarray

from the original variable. :rtype: bool

:returns standard deviation of the variable. :rtype: numpy.ndarray

get_xarray()
Build xarray dataset.

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

Return type dataset

:returns encoding :rtype: dict

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

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

envlib.accf.**get_encoding_dict**(*list_name*, *encoding*)

envlib.accf.**get_latlon**(*ds*, *member_bool*)

envlib.calc_altrv_vars module

envlib.calc_altrv_vars.**get_pvu**(*ds*)
Caclulates potential vorticity from meteorological variables temperature and components of winds.

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

Returns **PVUU** potential vorticity unit

Return type numpy.ndarray

envlib.calc_altrv_vars.**get_r**(*ds*)

envlib.calc_altrv_vars.**get_rh_ice**(*ds*)
Calculates the 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

envlib.calc_altrv_vars.**get_rh_wa**(*ds*)
Calculates the relative humidity over water from specific humidity

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

Returns **rh_wa** relative humidity over water

Return type numpy.ndarray

envlib.contrail module

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.

param ds Dataset openned with xarray.

type ds Dataset

param member Detemines the presense of ensemble forecasts in the given dataset.

type member bool

returns rcontr Thresholds of relative humidity over water.

rtype numpy.ndarray

returns TLM_e Thresholds of temperature.

rtype 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.

param ds Dataset openned with xarray.

type ds Dataset

param member Detemines the presense of ensemble forecasts in the given dataset.

type member bool

returns rcontr Thresholds of relative humidity over water.

rtype numpy.ndarray

returns TLM_e Thresholds of temperature.

rtype numpy.ndarray

envlib.contrail.**get_rw_from_specific_hum**(*ds*, *member*)
Calculates of 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

envlib.contrail.**potential_con_cir_cov**(*ds*, *r_crit*, *r_ice*)

envlib.database module

envlib.emission_indices module

envlib.extend_dim module

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 exsitance of ensemple members. e.g., for a data only has two fields, latitude and longitude, this function addds time and level fields.

Parameters

- **ds** (Dataset) – information on original coordinates.
- **ds** – Dataset openned with xarray containg 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.extract_data module

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 ensmeble members are not available, it assgins 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*)

Build a dictionary containing logical valules correspond to the possibility to calculate of aCCF.

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

Returns Dictionary containing logical valules correspond to the possibility to calculate of aCCFs.

Return type dict

envlib.io module

envlib.main_processing module

class envlib.main_processing.**ClimateImpact**(*path*, ***problem_config*)

Bases: object

auto_plotting()

calculate_accfs(***seetings*)

generate_output()

envlib.processing_surf_vars module

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 repeat it for to pressure levels for the sake of consistency of dimensions. For a specific time regarding 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 that exist in the dataset of pressure level parameters at this step.
- **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 repeat it for to pressure levels for the sake of consistency of dimensions. For a specific time regarding 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

envlib.weather_store module

class envlib.weather_store.**GeoArrayHandler**
Bases: object

axes: dict

bitriangular_filter = array([[0.0625, 0.125 , 0.0625], [0.125 , 0.25 , 0.125], [0.0625, 0.

decimate(*array*)

decimate_3d(*array*)

decimate_4d(*array*)

decimate_5d(*array*)

down2(*array*)
Decimates a 2D array by a factor of two after applying a triangular filter

classmethod down2_coord(*array*)

get_coords()

triangular_filter = array([0.25, 0.5 , 0.25])

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

Processing weather data

axes

get_xarray()
Build xarray dataset.

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

Return type dataset

reduce_domain(*bounds*, *verbose=False*)
Reduces horizontal domain and time .

Returns bounds ranges defined as tuple (e.g., lat_bound=(35, 60.0)).

Return type dict

class envlib.weather_store.**WeatherStore_**
Bases: envlib.weather_store.GeoArrayHandler

axes

envlib.weather_store.**get_bound_indexes**(*arr*, *bounds*, *verbose=False*)
Determine indices of a given array (e.g., latitude and longitude) needed for cutting geographical areas with respect to the user-difined bounds.

Parameters

- **arr** (numpy.ndarray) – a given array (e.g., latitude and longitude).
- **bounds** (tuple) – user-defined bounds.
- **verbose** (bool) – Show determined indices.

Returns slice(low, high) return the determined low and high indices of the given array that includes the

defined bounds. :rtype: slice

Module contents

6.1.2 setup module

6.1.3 test package

Submodules

test.test_main module

test.test_main.**test_main**()

Module contents

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

7.1 Acknowledmgements

*This library has been developed within **ALARM** and **FLyATM4E** Projects.*

- **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

e

t