Environmental Library Documentation

Release V0.0.1

DLR, TUHH, TUD, UC3M

May 11, 2022

GETTING STARTED

1 Introduction 1

2 Getting started: 3
- 2.1 Installation . . . 3
- 2.2 Configuration . . . 3
- 2.3 Input . . . 5
- 2.4 Running . . . 6
- 2.5 Output . . . 6

3 Modules: 7
- 3.1 Data Processing . . . 7
- 3.2 Calculation of Alternative Variables . . . 10
- 3.3 Weather Store . . . 10
- 3.4 Persistent Contrails Formation . . . 11
- 3.5 Calculation of aCCFs . . . 12

4 An example 15

5 Indices and tables 19
- 5.1 Acknowledmgements . . . 19

Python Module Index 21

CHAPTER

ONE

INTRODUCTION

About: 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-CO_2 aCCFs that combine the individual aCCFs can be calculated.

License: 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-CO_2 algorithmic climate change functions, GMD.

Support: XXX

CHAPTER

TWO

GETTING STARTED:

This section briefly presents the necessary information required to get started with EnVLiB.

2.1 Installation

The installation is the first step to working with EnVLiB. In the following, the steps required to install the library are provided (Some parts need to be modified, for instance, I need to check the current envlib is compatible with which versions of pythons, and also, for release, we may decide to publish it under PyPi, so downloading or cloning the library is not the only option).

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 (in MacOS and Linux) or cmd (Windows), which will install all dependencies:

```
python setup.py install
```

3. There is sample data and a test script in the package. To run it, at the library folder, enter the following command:

```
python setup.py pytest
```

4. The library runs successfully if env_processed.nc is generated at the library folder/test/sample_data/. One can visualize the file using a visualization tool (e.g., ferret, NCO, Panoply, etc.).

2.2 Configuration

The scope of EnVLiB is to provide individual and merged aCCFs as spatially and temporally resolved information considering meteorology from the actual synoptical situation, the engine/aircraft type, the selected physical climate metric, and the selected version of prototype algorithms in individual aCCFs. Consequently, some user-preferred settings need to be defined. Within EnVLib, theses settings are defined in a dictionary, called *confg* (i.e., confg [‘name’] = value). Notice that default configurations have been defined within the library; thus, defining dictionary *confg* is optional. Information on the settings, options, and default values is provided in the following.

```
confg = {}

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

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

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

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

# Specifies the threshold of relative humidity over ice in order to identify ice␣
↪supersaturated regions. Note that this threshold
confg['rhi_threshold'] = 0.90               # Options: depends on the resolution of the␣
↪input data (see SEction XX Dietmüller et al. 2021)
                                            # e.g., in case of ERA5_HRES it is 0.9

"""Output Options"""

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

# If true,  PMO effect included to CH$_4$ aCCF and total NOx aCCF
confg['PMO'] = True                         # Options: True, False (default: False)

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

# NOx and inverse EIs
confg['NOx&inverse_EIs'] = 'TTV'   # Options: 'TTV (typical transantlantic fleet mean␣
↪values)', 'ac_dependent' (default: TTV)
                                   # Note that "If Confg['NOx&inverse_Eis'] = 'TTV', the␣
↪following confg['ac_type'] is ignored."

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

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

"""Climate Hotspots"""

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

# 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)
```

(continues on next page)

(continued from previous page)

```
# If false, it assigns 0 for areas with climate impacts below the specified threshold and␣
↪gives actual values for those areas with higher climate impacts than the threshold.
confg['hotspots_binary'] = False             # Options: True, False (default: True)

# Determines dynamically the threshold for identifying climate hotspots by calculating␣
↪the e.g., 99th percentile term of the of the normal distribution of the respective␣
↪merged aCCF The percentiles are also outputted in netCDF output file
confg['hotspots_percentile'] = 99          # Options: percentage < 100 (default: 99)

""" Statistical analysis of EPS forecast """
# The following two options (confg['mean'], confg['std']) are ignored if the input data␣
↪are deterministic

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

""" Output """

# If true, all meteorological input variables are saved in the netCDF output file in same␣
↪resolution as aCCFs
confg['MET_variables'] = False             # Options: True, False (default: False)

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

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

2.3 Input

To calculate aCCFs, some meteorological variables are required. EnVLib takes these variables as input (See Table 5 of the connected paper (i.e., Dietmüller et al. (2021))). These variables are: Temperature, Geopotential, Relative humidity, and Potential vorticity unit at different pressure levels, and outgoing longwave radiation (or top net thermal radiation) at the top of the atmosphere. The current implementation of the Library is compatible with the standard of the European Centre for Medium-Range Weather Forecasts (ECMWF) data. Since the pressure level and surface variables are typically provided within different datasets, users should provide different datasets. Within EnVLiB, the directories of these two datasets are to be defined in a dictionary as follows:

```
input_dir = {}
input_dir['path_pl']  = dir_pressure_variables  # Directory for input data provided in␣
↪pressure levels such as temperature, geopotentialand relative humidity
input_dir['path_sur'] =  dir_surface_variables  # Directory for input data provided in␣
↪single pressure level such as top net thermal radiation at the the TOA
```

In addition to the directories for input data, directory of the EnvLib needs to be specified within input_dir:

```
input_dir ['path_lib'] = EnVLiB_dir      # Directory of EnVLiB
```

Finally, destination directory where all output will be written is to be inputted by user:

```
output_dir = dir_results    # Destination directory where all output will be written
```

2.4 Running

After defining configurations and inputting required directories, EnVLiB is ready to generate outputs. First of all, we import library:

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

Then, the inputted variables will be processed by using the following function. The processing in this step is mainly related to extracting variables within inputted data, calculating required variables from alternative ones in case of missing some variables (see Table 5 of the connected paper), unifying the naming and dimension of variables, and changing the resolution and geographical area. The preferred horizontal resolution and geographical area are inputted to the function. Notice that the horizontal resolution cannot be increased.

```
CI = ClimateImpact(input_dir, horizontal_resolution=resolution, lat_bound=(lat_min, lat_
↪max), lon_bound=(lon_min, lon_max), save_path=output_dir)
```

After processing the weather data, aCCFs are calculated using the following command with respect to the defined settings in the dictionary (i.e., confg) and saved within the netCDF file format in the specified directory.

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

2.5 Output

Following the previous steps, an output file (in netCDF format) will be generated. The output file contains different variables depending on the selected configurations (in *confg*). For instance, this file can contain both aCCFs and meteorological variables (if confg ['MET_variables'] = true). The generated netCDF file is compatible with well-known visualization tools such as ferret, NCO, and Panoply. In addition to the netCDF file, if one selects: confg['geojson'] = True, confg[Chotspots] = True, some GeoJson files (number of pressure levels * number of time) will be generated in the specified output file.

CHAPTER THREE

MODULES:

3.1 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

3.2 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

3.3 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

3.4 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

3.5 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(distance flown)]). 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(distance flown)]). 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

CHAPTER

FOUR

AN EXAMPLE

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

```
import envlib
from envlib.main_processing import ClimateImpact

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

""" %%%%%%%%%% CONFIGURATIONS %%%%%%%%%% """

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'] = 'TTV'
confg['ac_type'] = 'wide-body'
confg['NOx_aCCF'] = False

"""Climate Hotspots"""

confg['Chotspots'] = False
confg['hotspots_binary'] = False
confg['hotspots_percentile'] = 99

""" Statistical analysis of EPS forecast """
confg['mean'] = False
confg['std'] = False

""" Output """
confg['MET_variables'] = False
```

(continues on next page)

(continued from previous page)

```
confg['geojson'] = False
confg['color'] = 'copper'

""" %%%%%%%%%%%%%%%% MAIN %%%%%%%%%%%%%%%% """

CI = ClimateImpact(input_dir, horizontal_resolution=0.5, save_path=output_dir)
CI.calculate_accfs(**confg)
```

The output netCDF file is generated in: *envlib/test/sample_data/env_processed.nc*. In the following, a script is provided, enabling visualize the output.

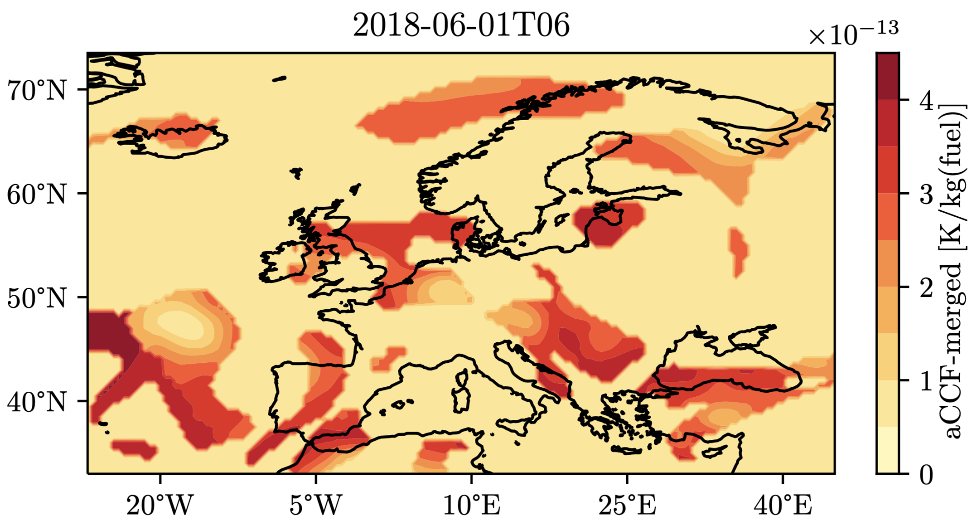
```
from cartopy.mpl.geoaxes import GeoAxes
import cartopy.crs as ccrs
from cartopy.mpl.geoaxes import GeoAxes
from cartopy.mpl.ticker import LongitudeFormatter, LatitudeFormatter
import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.axes_grid1 import AxesGrid
import numpy as np
import xarray as xr

plt.rc('font',**{'family':'serif','serif':['cmr10']})
plt.rc('text', usetex=True)
font = {'family' : 'normal',
        'size'   : 13}

path = 'envlib/test/sample_data/env_processed.nc'
ds = xr.open_dataset(path, engine='h5netcdf')
lats = ds['latitude'].values
lons = ds['longitude'].values
lons1,lats1 = np.meshgrid(lons,lats)

cc_lon = np.flipud(lons1)[::1, ::1]
cc_lat = np.flipud(lats1)[::1, ::1]


time = np.datetime64('2018-06-01T06')
pressure_level = 250
time_idx = np.where (ds.time.values == time)[0][0]
pl_idx   = np.where (ds.level.values == pressure_level) [0][0]
aCCF_merged  = np.flipud(ds['aCCF_merged'].values[time_idx, pl_idx, :, :])[::1, ::1]

def main():
    projection = ccrs.PlateCarree()
    axes_class = (GeoAxes,
                 dict(map_projection=projection))


    fig = plt.figure(figsize=(5,5))
    axgr = AxesGrid(fig, 111, axes_class=axes_class,
                    nrows_ncols=(1,1),
                    axes_pad=1.0,
                    share_all = True,
                    cbar_location='right',
                    cbar_mode='each',
```

(continues on next page)

(continued from previous page)

```
                    cbar_pad=0.2,
                    cbar_size='3%',
                    label_mode='')  # note the empty label_mode

    for i, ax in enumerate(axgr):

        xticks = [-20, -5, 10, 25, 40, 55]
        yticks = [0,10,20, 30, 40,  50,  60, 70, 80]
        ax.coastlines()
        ax.set_xticks(xticks, crs=projection)
        ax.set_yticks(yticks, crs=projection)
        lon_formatter = LongitudeFormatter(zero_direction_label=True)
        lat_formatter = LatitudeFormatter()
        ax.xaxis.set_major_formatter(lon_formatter)
        ax.yaxis.set_major_formatter(lat_formatter)
        ax.set_title(time)
        p = ax.contourf(cc_lon, cc_lat, aCCF_merged,
                        transform=projection,
                        cmap='YlOrRd')

        axgr.cbar_axes[i].colorbar(p)
        cax = axgr.cbar_axes[i]
        axis = cax.axis[cax.orientation]
        axis.label.set_text('aCCF-merged [K/kg(fuel)]')

    plt.show()

main()
```

For instance, using the script, one should get the following figure for the merged aCCF at 250hPa for 2018-06-01T06:



CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

5.1 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

envlib.accf, 14
envlib.calc_altrv_vars, 10
envlib.contrail, 11
envlib.extend_dim, 8
envlib.extract_data, 7
envlib.processing_surf_vars, 8
envlib.weather_store, 11