CLIMaCCF Documentation

Release V1.0

DLR, TUHH, TUD, UC3M

Jul 25, 2022

GETTING STARTED

CHAPTER

ONE

INTRODUCTION

About: The Python Library CLIMaCCF is a software package developed by UC3M and DLR. The main idea of CLIMaCCF 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). The 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.

License: CLIMaCCF is released under GNU General Public License Licence (Version 3). Citation of the CLIMaCCF connected software documentation paper is kindly requested upon use, with software DOI for CLIMaCCF (doi:XXX) and version number:

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

Support: Support of all general technical questions on CLIMaCCF, i.e. installation, application and development will be provided by Abolfazl Simorgh (abolfazl.simorgh@uc3m.es), Simone Dietmüller (Simone.Dietmueller@dlr.de), and Hiroshi Yamashita (Hiroshi.Yamashita@dlr.de).

Core developer team: Abolfazl Simorgh (UM3M), Simone Dietmüller (DLR), Hiroshi Yamashita (DLR), Manuel Soler (UC3M), Sigrun Matthes (DLR)

CHAPTER

TWO

GETTING STARTED:

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

2.1 Installation

The installation is the first step to working with CLIMaCCF. 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 CLIMaCCF 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:

```
conda create -n env_CLIMaCCF
conda activate env_CLIMaCCF
```

1. Clone or download the repository.
2. Locate yourself in the CLIMaCCF (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. The installation package contains a set of sample data and an example script for testing purpose. 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.

2.2 Configuration

The scope of CLIMaCCF is to provide individual and merged aCCFs as spatially and temporally resolved information considering meteorology from the actual synoptical situation, the 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 CLIMaCCF, theses settings are defined in a dictionary, called *confg* (i.e., confg ['name'] = value). Notice that default Default values for the settings have been defined within the library database; thus, defining dictionary *confg* is optional and, if included, overwrites the default ones.

```
confg = {}

"""Configuration of algorithmic climate change functions aCCFs """

# If true, efficacies are included
confg['efficacy'] = True      # Options: True, False
confg['efficacy-option'] = 'lee et al. (2021)'      # Option one: 'lee_2021'␣
↪(includes efficacies according to Lee et al. (2021)), Option two: {'CH4': xx,
↪'O3': xx, 'H2O': xx, 'Cont.': xx, 'CO2': xx} (user-defined efficacies assigned␣
↪to xx)

# Specifies the version of the prototype aCCF
confg['aCCF-V'] = 'V1.1'      # currently 2 options: 'V1.0': Yin et al. (2022),
↪'V1.1': Matthes et al. (2022)

# User-defined scaling factors of the above secelted aCCF version. Not␣
↪recommented to be changed, unless modification of the aCCFs is wanted (e.g.␣
↪sensitivity studies)
confg['aCCF-scalingF'] = {'CH4': 1, 'O3': 1, 'H2O': 1, 'Cont.': 1, 'CO2': 1}

# Specifies the emission scenario of the climate metric. Currently, pulse␣
↪emission and increasing future emission scenario (business as usual) included
confg['emission_scenario'] = 'future_scenario'      # Options: 'pulse' and
↪'future_scenario'

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

# Specifies the time horizon (in years) over which the selected climate indicator␣
↪is calculated
confg['TimeHorizon'] = 20      # Options: 20, 50, 100

# Determination of areas favorable for the formation of persistent contrails␣
↪(needed to calculate aCCF of (day/night) contrails).
confg['PCFA'] = ISSR      # Options: 'ISSR' (Ice-supersaturated reigons), 'SAC'␣
↪(Ice-supersaturation reigons with Schmidt-Appleman Criterion (Appleman, 1953))

# Specifies the thresholds of relative humidity over ice and temperature in order␣
↪to identify ice supersaturated regions. Note that the threshold of relative␣
↪humidity over ice depends on the resolution of the input data (for more details␣
↪see Dietmueller et al. 2022)
```

(continues on next page)

(continued from previous page)

```
confg['ISSR'] =  {'rhi_threshold': 0.95, 'temp_threshold': 235}      # Options␣
↪for 'rhi_threshold': user defined threshold value < 1. Threshold depends on the␣
↪used data set, e.g., in case of the reanalysis data product ERA5 with high␣
↪resolution (HRES) it is 0.9

# Parameters for calculating Schmidt-Appleman Criterion (SAC). These parameters␣
↪can vary for different aircraft types.
confg ['SAC'] = {'Q': 43 * 1e6, 'eta': 0.3, 'EI_H2O': 1.25}      # 'EI_H2O':␣
↪water vapour emission's index in [kg(H2O)/kg(fuel)], 'Q': Fuel specific energy␣
↪in [J/kg], 'eta': Engine's overall efficiency

""" Technical Specifiactions of Aircraft/Engine dependent Parameters"""

# Specifies NOx Emission Index (NOx_EI) and flown distance per kg burnt fuel (F_
↪km)
confg['NOx_EI&F_km'] = 'TTV'      # Options: 'TTV' for typical transantlantic␣
↪fleet mean values from literature and  'ac_dependent' for altitude and aircraft/
↪engine dependent values. Note that "If Confg['NOx_EI&F_km'] = 'TTV', the␣
↪following confg['ac_type'] is ignored."

# If Confg['NOx_EI&F_km'] = 'ac_dependent', aggregated aircraft type needs to be␣
↪selected. Note that these values take into account the altitude dependence of␣
↪NOx_EI and F_km (for more details see Dietmueller et al. 2022)
confg['ac_type'] = 'wide-body'      # Options: 'regional', 'single-aisle', 'wide-
↪body'


# weather-dependent coefficients for calculating NOx emission index using Boeing␣
↪Fuel Flow Method 2 (BFFM2)
confg['Coef.BFFM2'] = True      # Options: True, False
confg['method_BFFM2_SH'] = 'SH'

"""Output Options"""

# If true, the primary mode ozone (PMO) effect is included to the CH4 aCCF and␣
↪the total NOx aCCF
confg['PMO'] = True      # Options: True, False

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

# If true, all individual aCCFs are converted to K/kg(fuel) and outputted in this␣
↪unit.
confg['unit_K/kg(fuel)'] = False      # Options: True, False

# If true, merged non-CO2 aCCF is calculated
confg['merged'] = True      # Options: True, False

# If true, climate hotspots (regions that are very senitive to aviation␣
↪emissisions) are calculated (for more details see Dietmueller et al. 2022)
```

(continues on next page)

(continued from previous page)

```
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). 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

# 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

# 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

# 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

""" Output Options for Statistical analysis of Ensemble prediction system (EPS)␣
↪data products """

# 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

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

Another alternative is to include these settings in the separate configuration file and then load them in the main script. In the directory of CLIMaCCF, one can find a sample configuration file, including the mentioned settings in the YAML file format (i.e., config-user.yml), and can call them in the main script using

```
with open("config-user.yml", "r") as ymlfile: confg = yaml.load(ymlfile)
```

2.3 Input

To calculate aCCFs, some meteorological variables are required. CLIMaCCF takes these variables as input (See Table 5 of the connected paper (i.e., Dietmüller et al. (2021)). These variables are Temperature, Geopotential height, Relative humidity over ice, and Potential vorticity at different pressure levels, and outgoing longwave radiation (or top net thermal radiation) and incoming solar 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 (for both reanalysis and forecast data products). The user should provide two datasets, separating data provided at each pressure level and surface variables, typically collected in different datasets. Within CLIMaCCF, the directories of these two datasets are to be defined as follows:

```
input_dir = {}
input_dir['path_pl']  = dir_pressure_variables  # Directory for input data␣
↪provided in pressure levels such as temperature, geopotential and 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
```

Table 1: Main input prameters required for CLIMaCCF.

Parameter	Short name	Units	ID
Pressure	pres	$[K.m^2/Kg.s]$	54
Potential vorticity	pv	$[K.m^2/Kg.s]$	60
Geopotential	z	$[m^2/s^2]$	129
Temperature	t	$[K]$	130
Relative Humidity	r	[%]	157
Top Net Thermal Radiation	ttr	$[J/m^2]$	179
TOA Incident Solar Radiation	tisr	$[J/m^2]$	212

In addition to the locations of input data, the directory of the CLIMaCCF needs to be specified within input_dir:

```
input_dir ['path_lib'] = CLIMaCCF_dir      # Directory of CLIMaCCF
```

Finally, the directory where all outputs will be written is to be inputted by the user:

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

2.4 Running & Output

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

```
import CLIMaCCF
from CLIMaCCF.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 1) extracting variables within inputted data, 2) calculating required variables from alternative ones in case of missing some variables (see Table 5 of the connected paper), 3) unifying the naming and dimension of variables, and 4)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 higher than the resolution of the inputted meteorological data.

```
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)
```

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, the output file contains both individual and merged aCCFs if confg [‘merged’] = True and the inputted metrological parameters if confg [‘MET_variables’] = True. The dimension of outputted variables for the Ensemble prediction system (EPS) data products is (time, member, pressure level, latitude, longitude), and for the deterministic ones is (time, pressure level, latitude, longitude). 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: pressure levels * number of time) will be generated in the specified output directory.

CHAPTER THREE

MODULES:

3.1 Processing of meteorological input data

`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. If the data has only two fields: latitude and longitude, this function adds time and level fields, (e.g., for the deterministic data products: (latitude:360, longitude:720) -> (time:1, pressure level:1, latitude:360, longitude:720)).

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 meteorological input data from alternative variables

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

Caclulates potential vorticity [in PVU] from meteorological variables pressure, temperature and x and y component of the wind using MetPy (https://www.unidata.ucar.edu/software/metpy/).

Parameters

ds (Dataset) – Dataset openned with xarray.

Returns PVU

potential vorticity [in PVU]

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 [in %]

Return type
numpy.ndarray

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

Calculates the relative humidity over ice/water from specific humidity

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

Returns rh_sd
relative humidity over water/ice [%]

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 Contrail Formation

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

Calculates the threshold temperature and threshold of relative humidity over water required for contrail formation (Schmidt-Applemann-Citerion, Applemann 1953).
A good approximation of the Schmidt-Appleman Criterion is given in Schumann 1996.

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 for liquid saturation.

Return type
numpy.ndarray

Returns T_critT_crit_
Threshold temperature for Schmidt-Appleman

Return type
numpy.ndarray

envlib.contrail.**get_pcfa**(*ds, member, confg*)

Calculates the presistent contrail formation areas (pcfa) by using the Schmidt-Appleman Criterion (Appleman, 1953). Areas of presistent contrail formation are needed to calculate aCCF of (day/night) contrails.

Parameters

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

Returns pcfa
Presistent contrail formation areas (PCFA).

Return type
numpy.ndarray

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

Relative humiditiy over ice and water provided by ECMWF dataset. In ECMWF 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 ri
Relative humidity over ice.

Return type
numpy.ndarray

Returns rw
Relative humidity over water.

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 prototype aCCFs

`class` `envlib.accf.`**`GeTaCCFs`**(*wd_inf*)

Calculation of algorithmic climate change functions (aCCFs).

`__init__`(*wd_inf*)

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

Parameters
wd_inf (`Class`) – Contains processed weather data with all information.

`accf_ch4`()

Calculates the aCCF of methane according to Yin et al. 2022 (aCCF-V1.0) and Matthes et al. 2022 (aCCF-V1.1): aCCF values are given in average temperature response as over next 20 years, assuming pulse emission (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 according to Yin et al. 2022 (aCCF-V1.0) and Matthes et al. 2022 (aCCF-V1.1): aCCF values are given in average temperature response as over next 20 years, assuming pulse emissions (P-ATR20-contrails [K/km]). To calculate the aCCF of day-time contrails, meteorological variables temperature and relative humidity over ice are required. Notice that, relative humidity over ice is 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 according to Yin et al. 2022 (aCCF-V1.0) and Matthes et al. 2022 (aCCF-V1.1): aCCF values are given in average temperature response as over next 20 years, assuming pulse emission (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 according to Yin et al. 2022 (aCCF-V1.0) and Matthes et al. 2022 (aCCF-V1.1): aCCF values are given in average temperature response as over next 20 years, assuming pulse emissions (P-ATR20-contrails [K/km]). To calculate the aCCF of night-time contrails, meteorological variables temperature and relative humidity over ice are required. Notice that, relative humidity over ice is 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 according to Yin et al. 2022 (aCCF-V1.0) and Matthes et al. 2022 (aCCF-V1.1): aCCF values are given in average temperature response as over next 20 years, assuming pulse emission (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 configurations, 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 CLIMaCCF to generate output for a set of user-defined configurations:

```
import CLIMaCCF
from CLIMaCCF.main_processing import ClimateImpact

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

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

confg = {}

""" Configuration of algorithmic climate change functions aCCFs"""

confg['efficacy'] = True
confg['efficacy-option'] = 'lee_2021'
confg['aCCF-V'] = 'V1.1'
confg['aCCF-scalingF'] = {'CH4': 1, 'O3': 1, 'H2O': 1, 'Cont.': 1, 'CO2': 1}
confg['emission_scenario'] = 'future_scenario'
confg['climate_indicator'] = 'ATR'
confg['TimeHorizon'] = 20
confg['PCFA'] = ISSR
confg['ISSR'] = {'rhi_threshold': 0.95, 'temp_threshold': 235}
confg ['SAC'] = {'Q': 43 * 1e6, 'eta': 0.3, 'EI_H2O': 1.25}

""" Technical Specifiactions of Aircraft/Engine dependent Parameters"""

confg['NOx_EI&F_km'] = 'TTV'
confg['ac_type'] = 'wide-body'
confg['Coef.BFFM2'] = True
confg['method_BFFM2_SH'] = 'SH'


"""Output Options"""
```

(continues on next page)

(continued from previous page)

```
confg['PMO'] = True
confg['NOx_aCCF'] = False
confg['unit_K/kg(fuel)'] = False
confg['merged'] = True
confg['Chotspots'] = False
confg['hotspots_binary'] = False
confg['hotspots_percentile'] = 99
confg['MET_variables'] = False
confg['geojson'] = False
confg['color'] = 'copper'

""" Output Options for Statistical analysis of Ensemble prediction system (EPS)␣
↪data products """

confg['mean'] = False
confg['std'] = False


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

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

The output netCDF file is generated in: *CLIMaCCF/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 = 'CLIMaCCF/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]
```

(continues on next page)

(continued from previous page)

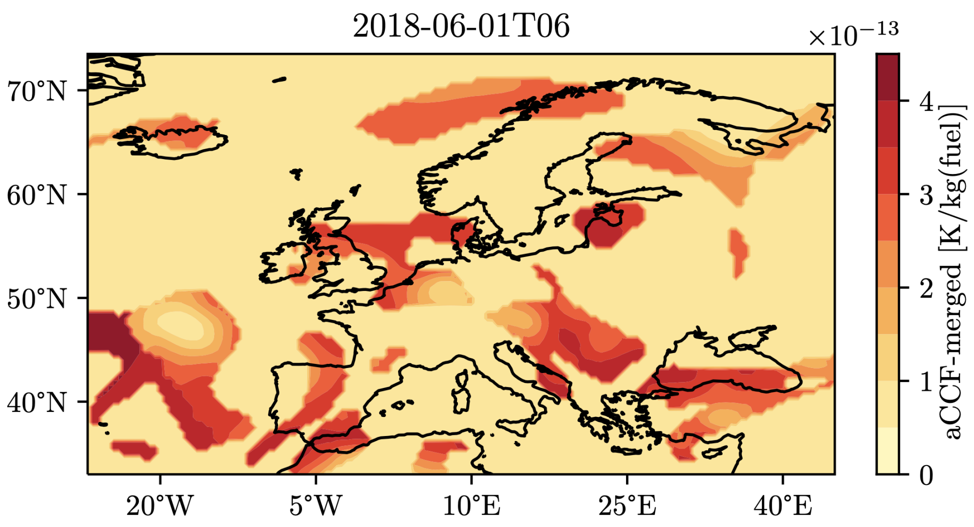
```
time = np.datetime64('2018-06-01T06')
pressure_level = 250
time_idx = np.where (ds.time.values == time)[0][0]
pl_idx   = np.where (ds.level.values == pressure_level) [0][0]
aCCF_merged  = np.flipud(ds['aCCF_merged'].values[time_idx, pl_idx, :, :])[::1,␣
↪::1]

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',
                    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()
```

(continues on next page)

(continued from previous page)

```
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