CLIMADA petals documentation

Release 4.1.0

CLIMADA contributors

Feb 19, 2024

CONTENTS

CLIMADA stands for CLIMate ADAptation and is a probabilistic natural catastrophe impact model, that also calculates averted damage (benefit) thanks to adaptation measures of any kind (from grey to green infrastructure, behavioural, etc.).

CLIMADA is primarily developed and maintained by the Weather and Climate Risks Group at ETH Zürich.

This is the documentation of the CLIMADA **Petals** module. Its purpose is generating different types of hazards and more specialized applications than available in the CLIMADA Core module.

Attention: CLIMADA Petals builds on top of CLIMADA Core and is **not** a standalone module. Before you start working with Petals, please check out the documentation of the CLIMADA Core module, in particular the installation instructions.

Jump right in:

- *README*
- Installation (Core and Petals)
- GitHub Repository
- *Module Reference*

Hint: ReadTheDocs hosts multiple versions of this documentation. Use the drop-down menu on the bottom left to switch versions. `stable` refers to the most recent release, whereas `latest` refers to the latest development version.

CHAPTER

ONE

SOFTWARE DOCUMENTATION PER PACKAGE

1.1 climada_petals.engine package

1.1.1 climada_petals.engine.supplychain module

1.1.2 climada_petals.engine.warn module

1.2 climada_petals.entity package

1.2.1 climada_petals.entity.exposures package

climada_petals.entity.exposures.openstreetmap package

climada_petals.entity.exposures.openstreetmap.osm_dataloader module

climada_petals.entity.exposures.black_marble module

class climada_petals.entity.exposures.black_marble.**BlackMarble**(**args*, *meta=None*, *description=None*, *ref_year=2018*, *value_unit='USD'*, *crs=None*, ***kwargs*)

Bases: `Exposures`

Defines exposures from night light intensity, GDP and income group. Attribute region_id is defined as: - United Nations Statistics Division (UNSD) 3-digit equivalent numeric code - 0 if country not found in UNSD. - -1 for water

set_countries(*countries*, *ref_year=2016*, *res_km=None*, *from_hr=None*, *admin_file='admin_0_countries'*, ***kwargs*)

Model countries using values at reference year. If GDP or income group not available for that year, consider the value of the closest available year.

Parameters

- **countries** (*list or dict*) – list of country names (admin0 or subunits) or dict with key = admin0 name and value = [admin1 names]
- **ref_year** (*int, optional*) – reference year. Default: 2016
- **res_km** (*float, optional*) – approx resolution in km. Default: nightlights resolution.

- **from_hr** (*bool, optional*) – force to use higher resolution image, independently of its year of acquisition.
- **admin_file** (*str*) – file name, admin_0_countries or admin_0_map_subunits
- **kwargs** (*optional*) – 'gdp' and 'inc_grp' dictionaries with keys the country ISO_alpha3 code. 'poly_val' list of polynomial coefficients [1,x,x^2,…] to apply to nightlight (DEF_POLY_VAL used if not provided). If provided, these are used.

climada_petals.entity.exposures.crop_production module

climada_petals.entity.exposures.crop_production.**DEF_HAZ_TYPE = 'RC'**

Default hazard type used in impact functions id.

climada_petals.entity.exposures.crop_production.**BBOX = (-180, -85, 180, 85)**

"Default geographical bounding box of the total global agricultural land extent

climada_petals.entity.exposures.crop_production.**YEARCHUNKS = {'ISIMIP2': {'1860soc': {'endyear': 1860, 'startyear': 1661, 'yearrange': (1800, 1860)}, '2005soc': {'endyear': 2299, 'startyear': 2006, 'yearrange': (2006, 2099)}, '2100rcp26soc': {'endyear': 2299, 'startyear': 2100, 'yearrange': (2100, 2299)}, 'histsoc': {'endyear': 2005, 'startyear': 1861, 'yearrange': (1976, 2005)}, 'rcp26soc': {'endyear': 2099, 'startyear': 2006, 'yearrange': (2006, 2099)}, 'rcp60soc': {'endyear': 2099, 'startyear': 2006, 'yearrange': (2006, 2099)}}, 'ISIMIP3': {'2015soc': {'endyear': 2014, 'startyear': 1850, 'yearrange': (1983, 2013)}, 'histsoc': {'endyear': 2014, 'startyear': 1850, 'yearrange': (1983, 2013)}}}**

start and end years per ISIMIP version and senario as in ISIMIP-filenames of landuse data containing harvest area per crop

climada_petals.entity.exposures.crop_production.**FN_STR_VAR = 'landuse-15crops_annual'**

fix filename part in input data

climada_petals.entity.exposures.crop_production.**CROP_NAME = {'mai': {'fao': 'Maize', 'input': 'maize', 'print': 'Maize'}, 'ri1': {'fao': 'Rice, paddy', 'input': 'rice', 'print': 'Rice 1st season'}, 'ri2': {'fao': 'Rice, paddy', 'input': 'rice', 'print': 'Rice 2nd season'}, 'ric': {'fao': 'Rice, paddy', 'input': 'rice', 'print': 'Rice'}, 'soy': {'fao': 'Soybeans', 'input': 'oil_crops_soybean', 'print': 'Soybeans'}, 'swh': {'fao': 'Wheat', 'input': 'temperate_cereals', 'print': 'Spring Wheat'}, 'whe': {'fao': 'Wheat', 'input': 'temperate_cereals', 'print': 'Wheat'}, 'wwh': {'fao': 'Wheat', 'input': 'temperate_cereals', 'print': 'Winter Wheat'}}**

mapping of crop names

climada_petals.entity.exposures.crop_production.**IRR_NAME = {'combined': {'name': 'combined'}, 'firr': {'name': 'irrigated'}, 'noirr': {'name': 'rainfed'}}**

Conversion factor weight [tons] to nutritional value [kcal]. Based on Mueller et al. (2021), https://doi.org/10.1088/1748-9326/abd8fc :

"For the aggregation of different crops, we compute total calories, assuming net water contents of 12% for maize, spring and winter wheat, 13% for rice and 9% for soybean, according to Wirsenius (2000) and caloric contents of the "as purchased" biomass (i.e. including the water content) of 3.56kcal/g for maize, 2.8kcal/g for rice, 3.35kcal/g for soybean and of 3.34kcal/g for spring and winter wheat, following FAO (2001)." (Müller et al., 2021)

Version 1: conversion factors for crop biomass "as purchased",
here applied as default for FAO-normalized production: Production [kcal] = Production [t] * KCAL_PER_TON [kcal/t]

climada_petals.entity.exposures.crop_production.**YEARS_FAO = (2008, 2018)**

Default years from FAO used (data file contains values for 1991-2018)

class climada_petals.entity.exposures.crop_production.**CropProduction**(**args*, *meta=None*, *description=None*, *ref_year=2018*, *value_unit='USD'*, *crs=None*, ***kwargs*)

Bases: Exposures

Defines agriculture exposures from ISIMIP input data and FAO crop data

geopandas GeoDataFrame with metadata and columns (pd.Series) defined in Attributes and Exposures.

crop

crop typee.g., 'mai', 'ric', 'whe', 'soy'

Type
str

set_from_isimip_netcdf(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_isimip_netcdf instead.

classmethod from_isimip_netcdf(*input_dir=None*, *filename=None*, *hist_mean=None*, *bbox=None*, *yearrange=None*, *cl_model=None*, *scenario=None*, *crop=None*, *irr=None*, *isimip_version=None*, *unit=None*, *fn_str_var=None*)

Wrapper to fill exposure from NetCDF file from ISIMIP. Requires historical mean relative cropyield module as additional input.

Parameters

- **input_dir** (*Path or str*) – path to input data directory, default: {CONFIG.exposures.crop_production.local_data}/Input/Exposure
- **filename** (*string*) – name of the landuse data file to use, e.g. "histsoc_landuse-15crops_annual_1861_2005.nc""
- **hist_mean** (*str or array*) – historic mean crop yield per centroid (or path)
- **bbox** (*list of four floats*) – bounding box: [lon min, lat min, lon max, lat max]
- **yearrange** (*int tuple*) – year range for exposure set e.g., (1990, 2010)
- **scenario** (*string*) – climate change and socio economic scenario e.g., '1860soc', 'histsoc', '2005soc', 'rcp26soc','rcp60soc','2100rcp26soc'
- **cl_model** (*string*) – abbrev. climate model (only for future projections of lu data) e.g., 'gfdl-esm2m', 'hadgem2-es', 'ipsl-cm5a-lr','miroc5'
- **crop** (*string*) – crop type e.g., 'mai', 'ric', 'whe', 'soy'
- **irr** (*string*) – irrigation type, default: 'combined' f.i 'firr' (full irrigation), 'noirr' (no irrigation) or 'combined'= firr+noirr
- **isimip_version** (*str*) – 'ISIMIP2' (default) or 'ISIMIP3'

- **unit** (*string*) – unit of the exposure (per year) f.i 't/y' (default), 'USD/y', or 'kcal/y'
- **fn_str_var** (*string*) – FileName STRing depending on VARiable and ISIMIP simuation round

Return type
Exposure

set_from_area_and_yield_nc4 (**args*, ***kwargs*)

This function is deprecated, use LitPop.from_area_and_yield_nc4 instead.

classmethod from_area_and_yield_nc4 (*crop_type*, *layer_yield*, *layer_area*, *filename_yield*, *filename_area*, *var_yield*, *var_area*, *bbox=(-180, -85, 180, 85)*, *input_dir=PosixPath('/home/docs/climada/data/ISIMIP_crop/Input/Exposure')*)

Set crop_production exposure from cultivated area [ha] and yield [t/ha/year] provided in two netcdf files with the same grid.

Both input files need to be netcdf format and come with dimensions 'lon', 'lat' and 'crop'. The information which crop type is saved in which crop layer in each input files needs to be provided manually via the parameters 'layer_*'.

A convenience wrapper around this expert method is provided with from_spam_ray_mirca().

Parameters

- **crop_type** (*str*) – Crop type, e.g. 'mai' for maize, or 'ric', 'whe', 'soy', etc.
- **layer_yield** (*int*) – crop layer in yield input data set. Index typically starts with 1.
- **layer_area** (*int*) – crop layer in area input data set. Index typically starts with 1.
- **filename_yield** (*str*) – Name of netcdf-file containing gridded yield data. Requires coordinates 'lon', 'lat', and 'crop'.
- **filename_area** (*str*) – Name of netcdf-file containing gridded cultivated area. Requires coordinates 'lon', 'lat', and 'crop'.
- **var_yield** (*str*) – variable name to be extracted from yield file, e.g. 'yield.rf', 'yield.ir', 'yield.tot', or depending on netcdf structure.
- **var_area** (*str*) – variable name to be extracted from area file, e.g. 'cultivated area rainfed', 'cultivated area irrigated', 'cultivated area all', or depending on netcdf structure.
- **bbox (tuple of four floats)** (*bounding box:*) – bounding box to be extracted: (lon min, lat min, lon max, lat max). The default is (-180, -85, 180, 85).
- **input_dir** (*Path, optional*) – directory where input data is found. The default is {CONFIG.exposures.crop_production.local_data}/Input/Exposure.

Returns
crop production exposure instance based on yield and area data

Return type
CropProduction

set_from_spam_ray_mirca (**args*, ***kwargs*)

This function is deprecated, use CropPoduction.from_spam_ray_mirca instead.

classmethod from_spam_ray_mirca (*crop_type*, *irrigation_type='all'*, *bbox=(-180, -85, 180, 85)*, *input_dir=PosixPath('/home/docs/climada/data/ISIMIP_crop/Input/Exposure')*)

Wrapper method around from_area_and_yield_nc4().

Set crop_production exposure from cultivated area [ha] and yield [t/ha/year] provided in default input files. The default input files are based on the public yield data from SPAM2005 with gaps filled based on Ray et.al (2012); and cultivated area from MIRCA2000, both as post-processed by Jägermeyr et al. 2020; See https://doi.org/10.1073/pnas.1919049117 for more information and cite when using this data for publication.

Parameters

- **crop_type** (*str*) – Crop type, e.g. 'mai' for maize, or 'ric', 'whe', 'soy', etc.
- **irrigation_type** (*str, optional*) – irrigation type to be extracted, the options are: 'all' : total crop production, i.e. irrigated + rainfed 'firr' : fully irrigated 'noirr' : not irrigated, i.e., rainfed The default is 'all'
- **bbox (list of four floats)** (*bounding box:*) – [lon min, lat min, lon max, lat max]
- **input_dir** (*Path, optional*) – directory where input data is found. The default is {CONFIG.exposures.crop_production.local_data}/Input/Exposure.

Returns
Crop production exposure based on SPAM and MIRCA data set

Return type
Exposure

set_mean_of_several_isimip_models (**args*, ***kwargs*)

This function is deprecated, use CropPoduction.from_mean_of_several_isimip_models instead.

classmethod from_mean_of_several_isimip_models (*input_dir=None*, *hist_mean=None*, *bbox=None*, *yearrange=None*, *cl_model=None*, *scenario=None*, *crop=None*, *irr=None*, *isimip_version=None*, *unit=None*, *fn_str_var=None*)

Wrapper to init exposure from several NetCDF files with crop yield data from ISIMIP.

Parameters

- **input_dir** (*string*) – path to input data directory
- **hist_mean** (*array*) – historic mean crop production per centroid
- **bbox** (*list of four floats*) – bounding box: [lon min, lat min, lon max, lat max]
- **yearrange** (*int tuple*) – year range for exposure set,e.g., (1976, 2005)
- **scenario** (*string*) – climate change and socio economic scenario e.g., 'histsoc' or 'rcp60soc'
- **cl_model** (*string*) – abbrev. climate model (only when landuse data is future projection) e.g., 'gfdl-esm2m' etc.
- **crop** (*string*) – crop type e.g., 'mai', 'ric', 'whe', 'soy'
- **irr** (*string*) – irrigation type f.i 'rainfed', 'irrigated' or 'combined'= rainfed+irrigated
- **isimip_version** (*str*) – 'ISIMIP2' (default) or 'ISIMIP3'
- **unit** (*string*) – unit of the exposure (per year) f.i 't/y' (default), 'USD/y', or 'kcal/y'
- **fn_str_var** (*string*) – FileName STRing depending on VARiable and ISIMIP simuation round

Return type
Exposure

set_value_to_kcal(**args*, ***kwargs*)

This function is deprecated, use function value_to_kcal instead.

set_value_to_usd(**args*, ***kwargs*)

This function is deprecated, use functiom value_to_usd instead.

aggregate_countries()

Aggregate exposure data by country.

Returns

- **list_countries** (*list*) – country codes (numerical ISO3)
- **country_values** (*array*) – aggregated exposure value

climada_petals.entity.exposures.crop_production.**value_to_kcal**(*exp_cp*, *biomass=True*)

Converts the exposure value from tonnes to kcal per year using conversion factor per crop type.

Parameters

- **exp_cp** (*CropProduction*) – CropProduction exposure object with units tonnes per year ('t/y')
- **biomass** (*bool, optional*) – if true, KCAL_PER_TON['biomass'] is used (default, for FAO normalized crop production). If False, KCAL_PER_TON['drymatter'] is used (best for crop model output in dry matter, default for raw crop model output). Default: True

Returns

new_exp – CropProduction exposure object with unit 'kcal/y'

Return type

CropProduction

climada_petals.entity.exposures.crop_production.**value_to_usd**(*exp_cp*, *input_dir=None*, *yearrange=None*)

Calculates the exposure in USD per year using country and year specific data published by the FAO, requires crop production exposure with unit 't/y'

Parameters

- **exp_cp** (*CropProduction*) – CropProduction exposure object with units tonnes per year ('t/y')
- **input_dir** (*Path or str, optional*) – directory containing the input (FAO pricing) data, default: {CONFIG.exposures.crop_production.local_data}/Input/Exposure
- **yearrange** (*np.array, optional*) – year range for prices, can also be set to a single year Default is set to the arbitrary time range (2000, 2018) The data is available for the years 1991-2018
- **crop** (*str*) – crop type e.g., 'mai', 'ric', 'whe', 'soy'

Returns

new_exp – CropProduction exposure object with unit 'USD/y'

Return type

CropProduction

climada_petals.entity.exposures.crop_production.**init_full_exp_set_isimip**(*input_dir=None*, *filename=None*, *hist_mean_dir=None*, *output_dir=None*, *bbox=None*, *yearrange=None*, *unit=None*, *isimip_version=None*, *return_data=False*)

Generates CropProduction instances (exposure sets) for all files found in the
input directory and saves them as hdf5 files in the output directory. Exposures are aggregated per crop and irrigation type.

Parameters

- **input_dir** (*str or Path*) – path to input data directory, default: {CONFIG.exposures.crop_production.local_data}/Input/Exposure
- **filename** (*string*) – if not specified differently, the file 'histsoc_landuse-15crops_annual_1861_2005.nc' will be used
- **output_dir** (*string*) – path to output data directory
- **bbox** (*list of four floats*) – bounding box: [lon min, lat min, lon max, lat max]
- **yearrange** (*array*) – year range for hazard set, e.g., (1976, 2005)
- **isimip_version** (*str*) – 'ISIMIP2' (default) or 'ISIMIP3'
- **unit** (*str*) – unit in which to return exposure (e.g., t/y or USD/y)
- **return_data** (*boolean*) – returned output False: returns list of filenames only, True: returns also list of data

Returns

- **filename_list** (*list*) – all filenames of saved initiated exposure files
- **output_list** (*list*) – list containing all inisiated Exposure instances

climada_petals.entity.exposures.crop_production.**normalize_with_fao_cp**(*exp_firr*, *exp_noirr*, *input_dir=None*, *yearrange=None*, *unit=None*, *return_data=True*)

Normalize (i.e., bias correct) the given exposures countrywise with the mean crop production quantity documented by the FAO. Refer to the beginning of the script for guidance on where to download the required crop production data from FAO.Stat.

Parameters

- **exp_firr** (*crop_production*) – exposure under full irrigation
- **exp_noirr** (*crop_production*) – exposure under no irrigation
- **input_dir** (*Path or str*) – directory containing exposure input data, default: {CONFIG.exposures.crop_production.local_data}/Input/Exposure
- **yearrange** (*array*) – the mean crop production in this year range is used to normalize the exposure data Default is set to the arbitrary time range (2008, 2018) The data is available for the years 1961-2018
- **unit** (*str*) – unit in which to return exposure (t/y or USD/y)
- **return_data** (*boolean*) – returned output True: returns country list, ratio = FAO/ISIMIP, normalized exposures, crop production per country as documented by the FAO and calculated by the ISIMIP dataset False: country list, ratio = FAO/ISIMIP, normalized exposures

Returns

- **country_list** (*list*) – List of country codes (numerical ISO3)
- **ratio** (*list*) – List of ratio of FAO crop production and aggregated exposure for each country
- **exp_firr_norm** (*CropProduction*) – Normalized CropProduction (full irrigation)
- **exp_noirr_norm** (*CropProduction*) – Normalized CropProduction (no irrigation)
- **Returns** (*optional*)
- **fao_crop_production** (*list*) – FAO crop production value per country
- **exp_tot_production** (*list*) – Exposure crop production value per country (before normalization)

climada_petals.entity.exposures.crop_production.**normalize_several_exp**(*input_dir=None*, *output_dir=None*, *yearrange=None*, *unit=None*, *return_data=True*)

Multiple exposure sets saved as HDF5 files in input directory are normalized (i.e. bias corrected) against FAO statistics of crop production.

Parameters

- **input_dir** (*Path or str*) – directory containing exposure input data
- **output_dir** (*Path or str*) – directory containing exposure datasets (output of exposure creation)
- **yearrange** (*array*) – the mean crop production in this year range is used to normalize the exposure data (default 2008-2018)
- **unit** (*str*) – unit in which to return exposure (t/y or USD/y)
- **return_data** (*boolean*) – returned output True: lists containing data for each exposure file. Lists: crops, country list, ratio = FAO/ISIMIP, normalized exposures, crop production per country as documented by the FAO and calculated by the ISIMIP dataset False: lists containing data for each exposure file. Lists: crops, country list, ratio = FAO/ISIMIP, normalized exposures

Returns

- **crop_list** (*list*) – List of crops

- **country_list** (*list*) – List of country codes (numerical ISO3)
- **ratio** (*list*) – List of ratio of FAO crop production and aggregated exposure for each country
- **exp_firr_norm** (*list*) – List of normalized CropProduction Exposures (full irrigation)
- **exp_noirr_norm** (*list*) – List of normalize CropProduction Exposures (no irrigation)
- **fao_crop_production** (*list, optional*) – FAO crop production value per country
- **exp_tot_production** (*list, optional*) – Exposure crop production value per country (before normalization)

climada_petals.entity.exposures.crop_production.**semilogplot_ratio**(*crop*, *countries*, *ratio*, *output_dir=None*, *save=True*)

Plot ratio = FAO/ISIMIP against country codes.

Parameters

- **crop** (*str*) – crop to plot
- **countries** (*list*) – country codes of countries to plot
- **ratio** (*array*) – ratio = FAO/ISIMIP crop production data of countries to plot
- **save** (*boolean*) – True saves figure, else figure is not saved.
- **output_dir** (*str*) – directory to save figure

Returns

- *fig (plt figure handle)*
- *axes (plot axes handle)*

climada_petals.entity.exposures.gdp_asset module

class climada_petals.entity.exposures.gdp_asset.**GDP2Asset**(**args*, *meta=None*, *description=None*, *ref_year=2018*, *value_unit='USD'*, *crs=None*, ***kwargs*)

Bases: Exposures

set_countries(*countries=[]*, *reg=[]*, *ref_year=2000*, *path=None*)

Model countries using values at reference year. If GDP or income group not available for that year, consider the value of the closest available year.

Parameters

- **countries** (*list*) – list of country names ISO3
- **ref_year** (*int, optional*) – reference year. Default: 2016
- **path** (*string*) – path to exposure dataset (ISIMIP)

climada_petals.entity.exposures.spam_agrar module

climada_petals.entity.exposures.spam_agrar.**DEF_HAZ_TYPE = 'CP'**

Default hazard type used in impact functions id.

climada_petals.entity.exposures.spam_agrar.**FILENAME_SPAM = 'spam2005V3r2_global'**

Add Docstring!

Type
TODO

climada_petals.entity.exposures.spam_agrar.**FILENAME_CELL5M = 'cell5m_allockey_xy.csv'**

Add Docstring!

Type
TODO

climada_petals.entity.exposures.spam_agrar.**FILENAME_PERMALINKS = 'spam2005V3r2_download_permalinks.csv'**

Add Docstring!

Type
TODO

climada_petals.entity.exposures.spam_agrar.**BUFFER_VAL = -340282306073709652508363335590014353408**

Hard coded value which is used for NANs in original data

climada_petals.entity.exposures.spam_agrar.**SPAM_URL = 'https://dataverse.harvard.edu/api/access/datafile/:persistentId?persistentId=doi:10.7910/DVN/DHXBJX/'**

URL stem for accessing data set files through api

climada_petals.entity.exposures.spam_agrar.**SPAM_DATASET = 'https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DHXBJX'**

Data files can be downloaded from this location if api access fails

class climada_petals.entity.exposures.spam_agrar.**SpamAgrar**(**args*, *meta=None*, *description=None*, *ref_year=2018*, *value_unit='USD'*, *crs=None*, ***kwargs*)

Bases: Exposures

Defines agriculture exposures from SPAM (Global Spatially-Disaggregated Crop Production Statistics Data for 2005 Version 3.2) https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DHXBJX

Attribute region_id is defined as: - United Nations Statistics Division (UNSD) 3-digit equivalent numeric code - 0 if country not found in UNSD. - -1 for water

init_spam_agrar(***parameters*)

initiates agriculture exposure from SPAM data:

https://dataverse.harvard.edu/ dataset.xhtml?persistentId=doi:10.7910/DVN/DHXBJX

Parameters

- **data_path** (*str*) – absolute path where files are stored. Default: SYSTEM_DIR
- **country** (*str*) – Three letter country code of country to be cut out. No default (global)
- **name_adm1** (*str*) – Name of admin1 (e.g. Federal State) to be cut out. No default
- **name_adm2** (*str*) – Name of admin2 to be cut out. No default
- **spam_variable** (*str*) – select one agricultural variable: 'A' physical area 'H' harvested area 'P' production 'Y' yield 'V_agg' value of production, aggregated to all crops, food and non-food (default) Warning: for A, H, P and Y, currently all crops are summed up
- **spam_technology** (*str*) – select one agricultural technology type: 'TA' all technologies together, ie complete crop (default) 'TI' irrigated portion of crop 'TH' rainfed high inputs portion of crop 'TL' rainfed low inputs portion of crop 'TS' rainfed subsistence portion of crop 'TR' rainfed portion of crop (= TA - TI, or TH + TL + TS) ! different impact_ids are assigned to each technology (1-6)
- **save_name_adm1** (*Boolean*) – Determines how many aditional data are saved: False: only basics (lat, lon, total value), region_id per country True: like 1 + name of admin1
- **haz_type** (*str*) – hazard type abbreviation, e.g. 'DR' for Drought or 'CP' for CropPotential

1.2.2 climada_petals.entity.impact_funcs package

climada_petals.entity.impact_funcs.drought module

class climada_petals.entity.impact_funcs.drought.**ImpfDrought**

Bases: `ImpactFunc`

Impact function for droughts.

__init__()

Empty initialization.

Parameters

- **impf_id** (*int, optional*) – impact function id. Default: 1
- **intensity** (*np.array, optional*) – intensity array SPEI [-]. default: intensity defintion 1 (minimum) default_sum: intensity definition 3 (sum over all drought months)

Raises

ValueError –

set_default(**args*, ***kwargs*)

This function is deprecated, use ImpfDrought.from_default instead.

set_default_sum(**args*, ***kwargs*)

This function is deprecated, use ImpfDrought.from_default_sum instead.

set_default_sumthr(**args*, ***kwargs*)

This function is deprecated, use ImpfDrought.from_default_sumthr instead.

set_step(**args*, ***kwargs*)

This function is deprecated, use ImpfDrought.from_step instead.

classmethod from_default()

Returns
impf – Default impact function.

Return type
ImpfDrought

classmethod from_default_sum()

Returns
impf – Default sum impact function.

Return type
ImpfDrought

classmethod from_default_sumthr()

Returns
impf – Default sum threshold impact function.

Return type
ImpfDrought

classmethod from_step()

Returns
impf – step impact function.

Return type
ImpfDrought

climada_petals.entity.impact_funcs.drought.**IFDrought**()

Is ImpfDrought now

Deprecated since version The: class name IFDrought is deprecated and won't be supported in a future version. Use ImpfDrought instead

climada_petals.entity.impact_funcs.relative_cropyield module

class
climada_petals.entity.impact_funcs.relative_cropyield.**ImpfRelativeCropyield**

Bases: ImpactFunc

Impact functions for agricultural droughts.

__init__()

Initialization.

Parameters

- **haz_type** (*str, optional*) – Hazard type acronym (e.g. 'TC').
- **id** (*int or str, optional*) – id of the impact function. Exposures of the same type will refer to the same impact function id.
- **intensity** (*np.array, optional*) – Intensity values. Defaults to empty array.
- **mdd** (*np.array, optional*) – Mean damage (impact) degree for each intensity (numbers in [0,1]). Defaults to empty array.

- **paa** (*np.array, optional*) – Percentage of affected assets (exposures) for each intensity (numbers in [0,1]). Defaults to empty array.
- **intensity_unit** (*str, optional*) – Unit of the intensity.
- **name** (*str, optional*) – Name of the ImpactFunc.

set_relativeyield(**args*, ***kwargs*)

This function is deprecated, use ImpfRelativeCropyield.impf_relativeyield instead.

classmethod impf_relativeyield()

Impact functions defining the impact as intensity

Returns
impf

Return type
climada.entity.impact_funcs.ImpfRelativeCropyield instance

climada_petals.entity.impact_funcs.relative_cropyield.**IFRelativeCropyield**()

Is ImpfRelativeCropyield now

Deprecated since version The: class name IFRelativeCropyield is deprecated and won't be supportedin a future version. Use ImpfRelativeCropyield instead

climada_petals.entity.impact_funcs.river_flood module

class climada_petals.entity.impact_funcs.river_flood.**ImpfRiverFlood**

Bases: ImpactFunc

Impact functions for tropical cyclones.

__init__()

Initialization.

Parameters

- **haz_type** (*str, optional*) – Hazard type acronym (e.g. 'TC').
- **id** (*int or str, optional*) – id of the impact function. Exposures of the same type will refer to the same impact function id.
- **intensity** (*np.array, optional*) – Intensity values. Defaults to empty array.
- **mdd** (*np.array, optional*) – Mean damage (impact) degree for each intensity (numbers in [0,1]). Defaults to empty array.
- **paa** (*np.array, optional*) – Percentage of affected assets (exposures) for each intensity (numbers in [0,1]). Defaults to empty array.
- **intensity_unit** (*str, optional*) – Unit of the intensity.
- **name** (*str, optional*) – Name of the ImpactFunc.

classmethod from_region(*region*)

Create a new ImpfRiverFlood object with parameters for the specified world region.

Parameters
region (*str*) – Use damage function parameters for this world region. Supported values: "Africa", "Asia", "Europe", "NorthAmerica", "Oceania", "SouthAmerica".

Returns
impf – New ImpfRiverFlood object with parameters for the specified world region.

Return type
ImpfRiverFlood

Raises
ValueError –

set_RF_Impf_Africa(**args*, ***kwargs*)
This function is deprecated, use ImpfRiverFlood.from_region instead.

set_RF_Impf_Asia(**args*, ***kwargs*)
This function is deprecated, use ImpfRiverFlood.from_region instead.

set_RF_Impf_Europe(**args*, ***kwargs*)
This function is deprecated, use ImpfRiverFlood.from_region instead.

set_RF_Impf_NorthAmerica(**args*, ***kwargs*)
This function is deprecated, use ImpfRiverFlood.from_region instead.

set_RF_Impf_Oceania(**args*, ***kwargs*)
This function is deprecated, use ImpfRiverFlood.from_region instead.

set_RF_Impf_SouthAmerica(**args*, ***kwargs*)
This function is deprecated, use ImpfRiverFlood.from_region instead.

climada_petals.entity.impact_funcs.river_flood.**IFRiverFlood**()
Is ImpfRiverFlood now

Deprecated since version The: class name IFRiverFlood is deprecated and won't be supported in a future version. Use ImpfRiverFlood instead

climada_petals.entity.impact_funcs.wildfire module

class climada_petals.entity.impact_funcs.wildfire.**ImpfWildfire**(*haz_type='WFsingle'*)
Bases: ImpactFunc

Impact function for wildfire.

__init__(*haz_type='WFsingle'*)
Initialization.

Parameters

- **haz_type** (*str, optional*) – Hazard type acronym (e.g. 'TC').
- **id** (*int or str, optional*) – id of the impact function. Exposures of the same type will refer to the same impact function id.
- **intensity** (*np.array, optional*) – Intensity values. Defaults to empty array.
- **mdd** (*np.array, optional*) – Mean damage (impact) degree for each intensity (numbers in [0,1]). Defaults to empty array.
- **paa** (*np.array, optional*) – Percentage of affected assets (exposures) for each intensity (numbers in [0,1]). Defaults to empty array.
- **intensity_unit** (*str, optional*) – Unit of the intensity.
- **name** (*str, optional*) – Name of the ImpactFunc.

classmethod from_default_FIRMS(*i_half=295.01*, *impf_id=1*)

This function sets the impact curve to a sigmoid type shape, as common in impact modelling. We adapted the function as proposed by Emanuel et al. (2011) which hinges on two parameters (intercept (i_thresh) and steepness (i_half) of the sigmoid).

$$f = \frac{i^3}{1 + i^3}$$

with

$$i = \frac{MAX[(i_{lat,lon} - i_{thresh}), 0]}{i_{half} - i_{thresh}}$$

The intercept is defined at the minimum intensity of a FIRMS value (295K) which leaves the steepness (i_half) the only parameter that needs to be calibrated.

Here, i_half is set to 295 K as a result of the calibration performed by Lüthi et al. (in prep). This value is suited for an exposure resolution of 1 km.

Calibration was further performed for:

- 4 km: resulting i_half = 409.4 K
- 10 km: resulting i_half = 484.4 K

Calibration has been performed globally (using EMDAT data) and is based on 84 damage records since 2001.

Intensity range is set between 295 K and 500 K as this is the typical range of FIRMS intensities.

Parameters

- **i_half** (*float, optional, default = 295.01*) – steepnes of the IF, [K] at which 50% of max. damage is expected
- **if_id** (*int, optional, default = 1*) – impact function id

Returns
Impf

Return type
climada.entity.impact_funcs.ImpfWildfire instance

set_default_FIRMS(**args*, ***kwargs*)

This function is deprecated, use ImpfWildfire.from_default_FIRMS instead.

1.3 climada_petals.hazard package

1.3.1 climada_petals.hazard.emulator package

climada_petals.hazard.emulator.const module

climada_petals.hazard.emulator.const.**TC_BASIN_GEOM = {'EP': [[-180.0, -75.0, 0.0, 9.0], [-180.0, -83.5, 9.0, 15.0], [-180.0, -92.0, 15.0, 18.0], [-180.0, -99.9, 18.0, 60.0]], 'EPE': [[-135.0, -75.0, 0.0, 9.0], [-135.0, -83.5, 9.0, 15.0], [-135.0, -92.0, 15.0, 18.0], [-135.0, -99.9, 18.0, 60.0]], 'EPW': [[-180.0, -135.0, 0.0, 60.0]], 'GB': [[-179.9, 180.0, -50.0, 60.0]], 'NA': [[-99.0, 13.0, 18.0, 60.0], [-91.0, 13.0, 15.0, 18.0], [-83.5, 13.0, 9.0, 15.0], [-78.0, 13.0, 0.0, 9.0]], 'NAN': [[-99.0, 13.0, 31.0, 60.0]], 'NAS': [[-99.0, 13.0, 18.0, 31.0], [-91.5, 13.0, 15.0, 18.0], [-83.5, 13.0, 9.0, 15.0], [-78.0, 13.0, 0.0, 9.0]], 'NI': [[37.0, 99.0, 0.0, 30.0]], 'NIE': [[78.0, 99.0, 0.0, 30.0]], 'NIW': [[37.0, 78.0, 0.0, 30.0]], 'SA': [[-65.0, 20.0, -60.0, 0.0]], 'SI': [[20.0, 135.0, -50.0, 0.0]], 'SIE': [[75.0, 135.0, -50.0, 0.0]], 'SIW': [[20.0, 75.0, -50.0, 0.0]], 'SP': [[135.0, 180.01, -50.0, 0.0], [-180.0, -68.0, -50.0, 0.0]], 'SPE': [[172.0, 180.01, -50.0, 0.0], [-180.0, -68.0, -50.0, 0.0]], 'SPW': [[135.0, 172.0, -50.0, 0.0]], 'WP': [[99.0, 180.0, 0.0, 60.0]], 'WPN': [[99.0, 180.0, 20.0, 60.0]], 'WPS': [[99.0, 180.0, 0.0, 20.0]]}**

Boundaries of TC (sub-)basins (lon_min, lon_max, lat_min, lat_max)

climada_petals.hazard.emulator.const.**TC_BASIN_GEOM_SIMPL = {'EP': [[-180.0, -75.0, 0.0, 60.0]], 'EPE': [[-135.0, -75.0, 0.0, 60.0]], 'EPW': [[-180.0, -135.0, 0.0, 60.0]], 'NA': [[-105.0, -30.0, 0.0, 60.0]], 'NAN': [[-105.0, -30.0, 31.0, 60.0]], 'NAS': [[-105.0, -30.0, 0.0, 31.0]], 'NI': [[37.0, 99.0, 0.0, 35.0]], 'NIE': [[78.0, 99.0, 0.0, 35.0]], 'NIW': [[37.0, 78.0, 0.0, 35.0]], 'SI': [[20.0, 135.0, -50.0, 0.0]], 'SIE': [[75.0, 135.0, -50.0, 0.0]], 'SIW': [[20.0, 75.0, -50.0, 0.0]], 'SP': [[135.0, -60.0, -50.0, 0.0]], 'SPE': [[172.0, -60.0, -50.0, 0.0]], 'SPW': [[135.0, 172.0, -50.0, 0.0]], 'WP': [[99.0, 180.0, 0.0, 60.0]], 'WPN': [[99.0, 180.0, 20.0, 60.0]], 'WPS': [[99.0, 180.0, 0.0, 20.0]]}**

Simplified boundaries of TC (sub-)basins (lon_min, lon_max, lat_min, lat_max)

climada_petals.hazard.emulator.const.**TC_SUBBASINS = {'EP': ['EPW', 'EPE'], 'NA': ['NAN', 'NAS'], 'NI': ['NIW', 'NIE'], 'SA': ['SA'], 'SI': ['SIW', 'SIE'], 'SP': ['SPW', 'SPE'], 'WP': ['WPN', 'WPS']}**

Abbreviated names of TC subbasins for each basin

climada_petals.hazard.emulator.const.**TC_BASIN_SEASONS = {'EP': [7, 12], 'NA': [6, 11], 'NI': [5, 12], 'SA': [1, 4], 'SI': [11, 4], 'SP': [11, 5], 'WP': [5, 12]}**

Start/end months of hazard seasons in different basins

climada_petals.hazard.emulator.const.**TC_BASIN_NORM_PERIOD = {'EP': (1950, 2015), 'NA': (1950, 2015), 'NI': (1980, 2015), 'SA': (1980, 2015), 'SI': (1980, 2015), 'SP': (1980, 2015), 'WP': (1950, 2015)}**

TC basin-specific start/end year of norm period (according to IBTrACS data availability)

climada_petals.hazard.emulator.const.**PDO_SEASON = [11, 3]**

Start/end months of PDO activity

climada_petals.hazard.emulator.emulator module

class climada_petals.hazard.emulator.emulator.**HazardEmulator**(*haz_events*, *haz_events_obs*, *region*, *freq_norm*, *pool=None*)

Bases: `object`

Draw samples for a time period driven by climate forcing

Draw samples from the given pool of hazard events while making sure that the frequency and intensity are as predicted according to given climate indices.

explaineds = ['intensity_mean', 'eventcount']

__init__(*haz_events*, *haz_events_obs*, *region*, *freq_norm*, *pool=None*)

Initialize HazardEmulator

Parameters

- **haz_events** (*DataFrame*) – Output of *stats.haz_max_events*.
- **haz_events_obs** (*DataFrame*) – Observed events for normalization. Output of *stats.haz_max_events*.
- **region** (*HazRegion object*) – The geographical region for which to run emulations.
- **freq_norm** (*DataFrame { year, freq }*) – Information about the relative surplus of events in *tracks*, i.e., if *freq_norm* specifies the value 0.2 in some year, then it is assumed that the number of events given for that year is 5 times as large as it is predicted to be. Usually, the value will be smaller than 1 because the event set should be a good representation of TC distribution, but this is not necessary.
- **pool** (*EventPool object, optional*) – If omitted, draws are made from *haz_events*.

calibrate_statistics(*climate_indices*)

Statistically fit hazard data to given climate indices

The internal statistics are truncated to fit the temporal range of the climate indices.

Parameters

climate_indices (*list of DataFrames { year, month, … }*) – Yearly or monthly time series of GMT, ESOI etc.

predict_statistics(*climate_indices=None*)

Predict hypothetical hazard statistics according to climate indices

The statistical fit from *calibrate_statistics* is used to predict the frequency and intensity of hazard events. The standard deviation of yearly residuals is used to define the yearly acceptable deviation of sample intensity.

Without calibration, the prediction is done according to the (bias-corrected) within-year statistics of the event pool. In this case, the within-year standard deviation of intensity is taken as the acceptable deviation of samples for that year.

Parameters

climate_indices (*list of DataFrames { year, month, … }*) – Yearly or monthly time series of GMT, ESOI etc. including at least those passed to *calibrate_statistics*. If omitted, and if *calibrate_statistics* has been called before, the climate indices from calibration are reused for prediction. Otherwise, the internal (within-year) statistics of the data set are used to predict frequency and intensity.

draw_realizations (*nrealizations*, *period*)

Draw samples for given time period according to calibration

Draws for a specific year in the given period are not necessarily restricted to events in the pool that are explicitly assigned to that year because the pool might be too small to allow for draws of the expected sample size and mean intensity.

Parameters

- **nrealizations** (*int*) – Number of samples to draw.
- **period** (*pair of ints [minyear, maxyear]*) – Period for which to make draws.

Returns

draws – Each entry is a sample for the whole period, given as a DataFrame with columns as in *self.pool.events*. The *year* column is set to the respective year and columns for the driving climate indices are added for reference.

Return type

list of DataFrames, length *nrealizations*

class climada_petals.hazard.emulator.emulator.**EventPool** (*haz_events*)

Bases: object

Make draws from a hazard event pool according to given statistics

The event pool might cover an arbitrary number of years and an arbitrary geographical region since the time and geo information fields are ignored when making draws.

No assumptions are made about where the statistics come from that are used in making the draw.

Example

Let *haz_events* be a given dataset of all TC events making landfall in Belize between 1980 and 2050, together with their respective maximum wind speeds on land. Assume that we expect (from some other statistical model) 5 events of annual mean maximum wind speed 30 ± 10 m/s in the year 2025. Then, we can draw 100 realizations of hypothetical 2025 TC event sets hitting Belize with the following commands:

```
>>> pool = EventPool(haz_events)
>>> draws = pool.draw_realizations(100, 5, 30, 10)
```

The realization *draw[i]* might contain events from any year between 1980 and 2050, but the size of the realization and the mean maximum wind speed will be according to the given statistics.

__init__ (*haz_events*)

Initialize instance of EventPool

Parameters

haz_events (*DataFrame*) – Output of *stats.haz_max_events*.

init_drop (*norm_period*, *norm_mean*)

Use a drop rule when making draws

With the drop rule, a random choice of entries is dropped from events before the actual drawing is done in order to speed up the process in case of data sets where the acceptable mean is far from the input data mean.

Parameters

- **norm_period** (*pair of ints [minyear, maxyear]*) – Normalization period for which a specific mean intensity is expected.

- **norm_mean** (*float*) – Desired mean intensity of events in the given time period.

draw_realizations (*nrealizations*, *freq_poisson*, *intensity_mean*, *intensity_std*)

Draw samples from the event pool according to given statistics

If *EventPool.init_drop* has been called before, the drop rule is applied.

Parameters

- **nrealizations** (*int*) – Number of samples to draw
- **freq_poisson** (*float*) – Expected sample size ("frequency", Poisson distributed).
- **intensity_mean** (*float*) – Expected sample mean intensity.
- **intensity_std** (*float*) – Acceptable deviation from *intensity_mean*.

Returns
draws – Each entry is a sample, given as a DataFrame with columns as in *self.events*.

Return type
list of DataFrames, length *nrealizations*

climada_petals.hazard.emulator.geo module

class climada_petals.hazard.emulator.geo.**HazRegion** (*extent=None*, *geometry=None*, *country=None*, *season=(1, 12)*)

Bases: object

Hazard region for given geo information

__init__ (*extent=None*, *geometry=None*, *country=None*, *season=(1, 12)*)

Initialize HazRegion

If several arguments are passed, the spatial intersection is taken.

Parameters

- **extent** (*tuple (lon_min, lon_max, lat_min, lat_max), optional*)
- **geometry** (*GeoPandas DataFrame, optional*)
- **country** (*str or list of str, optional*) – Countries are represented by their ISO 3166-1 alpha-3 identifiers. The keyword "all" chooses all countries (i.e., global land areas).
- **season** (*pair of int, optional*) – First and last month of hazard-specific season within this region

centroids (*latlon=None*, *res_as=360*)

Return centroids in this region

Parameters

- **latlon** (*pair (lat, lon), optional*) – Latitude and longitude of centroids. If not given, values are taken from CLIMADA's base grid (see *res_as*).
- **res_as** (*int, optional*) – One of 150 or 360. When *latlon* is not given, choose coordinates from centroids according to CLIMADA's base grid of given resolution in arc-seconds. Default: 360.

Returns
centroids

Return type
climada.hazard.Centroids object

class climada_petals.hazard.emulator.geo.**TCRegion**(*tc_basin=None*, *season=None*, ***kwargs*)

Bases: *HazRegion*

Hazard region with support for TC ocean basins

__init__(*tc_basin=None*, *season=None*, ***kwargs*)

Initialize TCRegion

The given geo information must be such that everything is contained in a single TC ocean basin.

Parameters

- **tc_basin** (*str*) – TC (sub-)basin abbreviated name, such as "SIW". If not given, automatically determined from geometry and basin bounds.
- ****kwargs** (*see HazRegion.__init__*)

climada_petals.hazard.emulator.geo.**get_tc_basin_geometry**(*tc_basin*)

Get TC (sub-)basin geometry

Parameters
tc_basin (*str*) – TC (sub-)basin abbreviated name, such as "SIW" or "NA".

Returns
df

Return type
GeoPandas DataFrame

climada_petals.hazard.emulator.random module

climada_petals.hazard.emulator.random.**estimate_drop**(*events*, *time_col*, *val_col*, *norm_period*, *norm_fact=None*, *norm_mean=None*)

Determine fraction of outlying events to be dropped

If the mean intensity of events in the given time period *norm_period* is far from the desired mean *norm_mean*, sampling from *events* will usually yield draws whose mean is far from the desired mean, so that many resamplings will be necessary in order to get an acceptable draw.

Dropping events off the desired mean before sampling can reduce the necessary number of samplings.

This function estimates which portion of the events should be dropped.

Parameters

- **events** (*DataFrame*) – Each row describes one event. The dataset should contain at least the columns *time_col* and *val_col*.
- **time_col** (*str*) – Name of time column in *events*.
- **val_col** (*str*) – Name of value column in *events*.
- **norm_period** (*pair of timestamps (e.g. floats or ints)*) – Normalization period for which a specific mean intensity is expected.
- **norm_mean** (*float*) – Desired mean intensity of events in the given time period.
- **norm_fact** (*float*) – Instead of *norm_mean*, the ratio between desired and observed intensity in the given time period can be given.

Returns
drop – Only events satisfying the pandas query expression *expr* should be eligible for dropping. *frac* specifies the fraction of these events that are to be dropped.

Return type
pair [expr, frac]

climada_petals.hazard.emulator.random.**draw_poisson_events**(*poisson*, *events*, *val_col*, *val_accept*, *drop=None*)

Draw poisson distributed events with acceptable value statistics

The size of the draw is poisson distributed. Redraws are made until the draw mean is within the range specified by *val_accept*.

If *drop* is specified, a random choice of entries is dropped from *events* before the actual drawing is done in order to speed up the process in case of data sets where the acceptable mean is far from the input data mean.

Parameters

- **poisson** (*float*) – Poisson parameter.
- **events** (*DataFrame*) – Each row describes one event. The dataset should contain at least the column *val_col*.
- **val_col** (*str*) – Name of value column in *events*.
- **val_accept** (*pair of floats*) – Acceptable range of draw means.
- **drop** (*pair [expr, frac] or None*) – If given, only events satisfying the pandas query expression *expr* are dropped. *frac* specifies the fraction of these events that is dropped.

Returns
draw_idx – Indices into *events*. If no acceptable draw was among the first 10,000 attempts, the return value is None.

Return type
Series or None

climada_petals.hazard.emulator.stats module

climada_petals.hazard.emulator.stats.**seasonal_average**(*data*, *season*)

Compute seasonal average from monthly-time series.

For seasons that are across newyear, the months after June are attributed to the following year's season. For example: The 6-month season from November 1980 till April 1981 is attributed to the year 1981.

The two seasons that are truncated at the beginning/end of the dataset's time period are discarded. When the input data is 1980-2010, the output data will be 1981-2010, where 2010 corresponds to the 2009/2010 season and 1981 corresponds to the 1980/1981 season.

Parameters

- **data** (*DataFrame { year, month, … }*) – All further columns will be averaged over.
- **season** (*pair of ints*) – Start/end month of season.

Returns
averaged_data – Same format as input, but with month column removed.

Return type
DataFrame { year, … }

climada_petals.hazard.emulator.stats.**seasonal_statistics**(*events*, *season*)

Compute seasonal statistics from given hazard event data

Parameters

- **events** (*DataFrame { year, month, intensity, … }*) – Events outside of the given season are ignored.
- **season** (*pair of ints*) – Start/end month of season.

Returns

haz_stats – For seasons that are across newyear, this might cover one year less than the input data since truncated seasons are discarded.

Return type

DataFrame { year, events, intensity_mean, intensity_std, intensity_max }

climada_petals.hazard.emulator.stats.**haz_max_events**(*hazard*, *min_thresh=0*)

Table of max intensity events for given hazard

Parameters

- **hazard** (*climada.hazard.Hazard object*)
- **min_thresh** (*float*) – Minimum intensity for event to be registered.

Returns

events – The integer value in column *id* refers to the internal order of events in the given *hazard* object. *lat*, *lon* and *intensity* specify location and intensity of the maximum intensity registered.

Return type

DataFrame { id, name, year, month, day, lat, lon, intensity }

climada_petals.hazard.emulator.stats.**normalize_seasonal_statistics**(*haz_stats*, *haz_stats_obs*, *freq_norm*)

Bias-corrected annual hazard statistics

Parameters

- **haz_stats** (*DataFrame { … }*) – Output of *seasonal_statistics*.
- **haz_stats_obs** (*DataFrame { … }*) – Output of *seasonal_statistics*.
- **freq_norm** (*DataFrame { year, freq }*) – Information about the relative surplus of hazard events per year, i.e., if *freq_norm* specifies the value 0.2 in some year, then it is assumed that the number of events given for that year is 5 times as large as it is predicted to be.

Returns

statistics – intensity_max_obs, intensity_mean_obs, eventcount_obs } Normalized and observed hazard statistics.

Return type

DataFrame { year, intensity_max, intensity_mean, eventcount,

climada_petals.hazard.emulator.stats.**fit_data**(*data*, *explained*, *explanatory*, *poisson=False*)

Fit a response variable (e.g. intensity) to a list of explanatory variables

The fitting is run twice, restricting to the significant explanatory variables in the second run.

Parameters

- **data** (DataFrame { year, *explained*, *explanatory*, … }) – An intercept column is added automatically.

- **explained** (*str*) – Name of explained variable, e.g. 'intensity'.
- **explanatory** (*list of str*) – Names of explanatory variables, e.g. ['gmt','esoi'].
- **poisson** (*boolean*) – Optionally, use Poisson regression for fitting. If False (default), uses ordinary least squares (OLS) regression.

Returns
sm_results – Results for first and second run.

Return type
pair of statsmodels Results object

climada_petals.hazard.emulator.stats.**fit_significant**(*sm_results*)

List significant variables in *sm_results*

Note: The last variable (usually intercept) is omitted!

climada_petals.hazard.emulator.stats.**fit_significance**(*sm_results*)

Extract and visualize significance of model parameters

1.3.2 climada_petals.hazard.drought module

class climada_petals.hazard.drought.**Drought**

Bases: Hazard

Contains drought events.

SPEI

Standardize Precipitation Evapotraspiration Index

Type
float

vars_opt = {'spei'}

Name of the variables that aren't need to compute the impact.

__init__()

Empty constructor.

set_area(*latmin*, *lonmin*, *latmax*, *lonmax*)

Set the area to analyse

set_file_path(*path*)

Set path of the SPEI data

set_threshold(*threshold*)

Set threshold

set_intensity_def(*intensity_definition*)

Set intensity definition

setup()

Set up the hazard drought

hazard_def(*intensity_matrix*)

return hazard set

Parameters
see intensity_from_spei

Returns

- *Drought, full hazard set*
- *check using new_haz.check()*

plot_intensity_drought(*event=None*)

plot drought intensity

post_processing(*date*)

Date in format '2003-08-01' Sets intensity of events starting after that date to zero

plot_start_end_date(*event=None*)

plot start and end date of the chosen event

1.3.3 climada_petals.hazard.landslide module

class climada_petals.hazard.landslide.**Landslide**

Bases: Hazard

Landslide Hazard set generation.

__init__()

Empty constructor.

classmethod from_hist(*bbox*, *input_gdf*, *res=0.0083333*)

Set historic landslide (ls) raster hazard from historical point records, for example as can be retrieved from the NASA COOLR initiative, which is the largest global ls repository, for a specific geographic extent. Points are assigned to the gridcell they fall into, and the whole grid- cell hence counts as equally affected. Event frequencies from an incomplete dataset are not meaningful and hence aren't set by default. probabilistic calculations! Use the probabilistic method for this!

See tutorial for details; the global ls catalog from NASA COOLR can bedownloaded from https://maps.nccs.nasa.gov/arcgis/apps/webappviewer/index.html?id=824ea5864ec8423fb985b33ee6bc05b7

Note: The grid which is generated has the same projection as the geodataframe with point occurrences. By default, this is EPSG:4326, which is a non- projected, geographic CRS. This means, depending on where on the globe the analysis is performed, the area per gridcell differs vastly. Consider this when setting your resoluton (e.g. at the equator, 1° ~ 111 km). In turn, one can use projected CRS which preserve angles and areas within the reference area for which they are defined. To do this, reproject the input_gdf to the desired projection. For more on projected & geographic CRS, see https://desktop.arcgis.com/en/arcmap/10.3/guide-books/map-projections/about-projected-coordinate-systems.htm

---

Parameters

- **bbox** (*tuple*) – (minx, miny, maxx, maxy) geographic extent of interest
- **input_gdf** (*str or or geopandas geodataframe*) – path to shapefile (.shp) with ls point data or already laoded gdf
- **res** (*float*) – resolution in units of the input_gdf crs of the final grid cells which are created. Whith EPSG:4326, this is degrees. Default is 0.008333.

Returns

Landslide – instance filled with historic LS hazard set for either point hazards or polygons with specified surrounding extent.

Return type
Landslide

set_ls_hist (**args*, ***kwargs*)

This function is deprecated, use Landslide.from_hist instead.

classmethod from_prob (*bbox*, *path_sourcefile*, *corr_fact=10000000.0*, *n_years=500*, *dist='poisson'*)

Set probabilistic landslide hazard (fraction, intensity and frequency) for a defined bounding box and time period from a raster. The hazard data for which this function is explicitly written is readily provided by UNEP & the Norwegian Geotechnical Institute (NGI), and can be downloaded and unzipped from https://preview.grid.unep.ch/index.php?preview=data&events=landslides&evcat=2&lang=eng for precipitation-triggered landslide and from https://preview.grid.unep.ch/index.php?preview=data&events=landslides&evcat=1&lang=eng for earthquake-triggered landslides. It works with any similar raster file. Original data is given in expected annual probability and percentage of pixel of occurrence of a potentially destructive landslide event x 1000000 (so be sure to adjust this by setting the correction factor). More details can be found in the landslide tutorial and under above- mentioned links.

Events are sampled from annual occurrence probabilites via binomial or poisson distribution. An event therefore includes all landslides sampled to occur within a year for the given area. intensity takes a binary value (occurrence or no occurrene of a LS); frequency is set to 1 / n_years.

Impact functions, since they act on the intensity, should hence be in the form of a step function, defining impact for intensity 0 and (close to) 1.

Parameters

- **bbox** (*tuple*) – (minx, miny, maxx, maxy) geographic extent of interest
- **path_sourcefile** (*str*) – path to UNEP/NGI ls hazard file (.tif)
- **corr_fact** (*float or int*) – factor by which to divide the values in the original probability file, in case it is not scaled to [0,1]. Default is 1'000'000
- **n_years** (*int*) – sampling period
- **dist** (*str*) – distribution to sample from. 'poisson' (default) and 'binom'

Returns
haz – probabilistic LS hazard

Return type
climada.hazard.Landslide instance

See also:

`sample_events`

set_ls_prob (**args*, ***kwargs*)

This function is deprecated, use Landslide.from_prob instead.

1.3.4 climada_petals.hazard.low_flow module

class climada_petals.hazard.low_flow.**LowFlow**(*pool=None*)

Bases: Hazard

Contains river low flow events (surface water scarcity). The intensity of the hazard is number of days below a threshold (defined as percentile in reference data). The method set_from_nc can be used to create a LowFlow hazard set populated with data based on gridded hydrological model runs as provided by the ISIMIP project (https://www.isimip.org/), e.g. ISIMIP2a/b. grid cells with a minimum number of days below threshold per month are clustered in space (lat/lon) and time (monthly) to identify and set connected events.

clus_thresh_t

maximum time difference in months to be counted as$ connected points during clustering, default = 1

Type
int

clus_thresh_xy

maximum spatial grid cell distance in number of cells to be counted as connected points during clustering, default = 2

Type
int

min_samples

Minimum amount of data points in one cluster to consider as event, default = 1.

Type
int

date_start

for each event, the date of the first month of the event (ordinal) Note: Hazard attribute 'date' contains the date of maximum event intensity.

Type
np.array(int)

date_end

for each event, the date of the last month of the event (ordinal)

Type
np.array(int)

resolution

spatial resoultion of gridded discharge input data in degree lat/lon, default = 0.5°

Type
float

clus_thresh_t = 1

clus_thresh_xy = 2

min_samples = 1

resolution = 0.5

__init__(*pool=None*)

Empty constructor.

set_from_nc(**args*, ***kwargs*)

This function is deprecated, use LowFlow.from_netcdf instead.

classmethod from_netcdf(*input_dir=None*, *centroids=None*, *countries=None*, *reg=None*, *bbox=None*, *percentile=2.5*, *min_intensity=1*, *min_number_cells=1*, *min_days_per_month=1*, *yearrange=(2001, 2005)*, *yearrange_ref=(1971, 2005)*, *gh_model=None*, *cl_model=None*, *scenario='historical'*, *scenario_ref='historical'*, *soc='histsoc'*, *soc_ref='histsoc'*, *fn_str_var='co2_dis_global_daily'*, *keep_dis_data=False*, *yearchunks='default'*, *mask_threshold=('mean', 1)*)

Wrapper to fill hazard from NetCDF file containing variable dis (daily), e.g. as provided from from ISIMIP Water Sectior (Global): https://esg.pik-potsdam.de/search/isimip/

Parameters

- **input_dir** (*string*) – path to input data directory. In this folder, netCDF files with gridded hydrological model output are required, containing the variable dis (discharge) on a daily temporal resolution as f.i. provided by the ISIMIP project (https://www.isimip.org/)
- **centroids** (*Centroids*) – centroids (area that is considered, reg and country must be None)
- **countries** (*list of countries ISO3*) – selection of countries (reg must be None!) [not yet implemented]
- **reg** (*list of regions*) – can be set with region code if whole areas are considered (if not None, countries and centroids are ignored) [not yet implemented]
- **bbox** (*tuple of four floats*) – bounding box: (lon min, lat min, lon max, lat max)
- **percentile** (*float*) – percentile used to compute threshold, 0.0 < percentile < 100.0
- **min_intensity** (*int*) – minimum intensity (nr of days) in an event event; events with lower max. intensity are dropped
- **min_number_cells** (*int*) – minimum spatial extent (nr of grid cells) in an event event; events with lower geographical extent are dropped
- **min_days_per_month** (*int*) – minimum nr of days below threshold in a month; months with lower nr of days below threshold are not considered for the event creation (clustering)
- **yearrange** (*int tuple*) – year range for hazard set, f.i. (2001, 2005)
- **yearrange_ref** (*int tuple*) – year range for reference (threshold), f.i. (1971, 2000)
- **gh_model** (*str*) – abbrev. hydrological model (only when input_dir is selected) f.i. 'H08', 'CLM45', 'ORCHIDEE', 'LPJmL', 'WaterGAP2', 'JULES-W1', 'MATSIRO'
- **cl_model** (*str*) – abbrev. climate model (only when input_dir is selected) f.i. 'gfdl-esm2m', 'hadgem2-es', 'ipsl-cm5a-lr', 'miroc5', 'gswp3', 'wfdei', 'princeton', 'watch'
- **scenario** (*str*) – climate change scenario (only when input_dir is selected) f.i. 'historical', 'rcp26', 'rcp60', 'hist'
- **scenario_ref** (*str*) – climate change scenario for reference (only when input_dir is selected)
- **soc** (*str*) – socio-economic trajectory (only when input_dir is selected) f.i. 'histsoc', # historical trajectory '2005soc', # constant at 2005 level 'rcp26soc', # RCP6.0 trajectory 'rcp60soc', # RCP6.0 trajectory 'pressoc' # constant at pre-industrial socio-economic level
- **soc_ref** (*str*) – csocio-economic trajectory for reference, like soc. (only when input_dir is selected)
- **fn_str_var** (*str*) – FileName STRing depending on VARiable and ISIMIP simuation round

- **keep_dis_data** (*boolean*) – keep monthly data (variable ndays = days below threshold) as dataframe (attribute "data") and save additional field 'relative_dis' (relative discharge compared to the long term)
- **yearchunks** – list of year chunks corresponding to each nc flow file. If set to 'default', uses the chunking corresponding to the scenario.
- **mask_threshold** – tuple with threshold value [1] for criterion [0] for mask: Threshold below which the grid is masked out. e.g.: ('mean', 1.) –> grid cells with a mean discharge below 1 are ignored ('percentile', .3) –> grid cells with a value of the computed percentile discharge values below 0.3 are ignored. default: ('mean', 1}). Set to None for no threshold. Provide a list of tuples for multiple thresholds.

Raises
NameError –

Returns
hazard set with lowflow calculated from netcdf file containing discharge data

Return type
LowFlow

set_intensity_from_clusters (*centroids=None*, *min_intensity=1*, *min_number_cells=1*, *yearrange=(2001, 2005)*, *keep_dis_data=False*)

Build low flow hazards with events from clustering and centroids and (re)set attributes.

events_from_clusters (*centroids*)

Initiate hazard events from connected clusters found in self.lowflow_df

Parameters
centroids (*Centroids*)

identify_clusters (*clus_thresh_xy=None*, *clus_thresh_t=None*, *min_samples=None*)

call clustering functions to identify the clusters inside the dataframe

Parameters

- **clus_thresh_xy** (*int*) – new value of maximum grid cell distance (number of grid cells) to be counted as connected points during clustering
- **clus_thresh_t** (*int*) – new value of maximum timse step difference (months) to be counted as connected points during clustering
- **min_samples** (*int*) – new value or minimum amount of data points in one cluster to retain the cluster as an event, smaller clusters will be ignored

Return type
pandas.DataFrame

filter_events (*min_intensity=1*, *min_number_cells=1*)

Remove events with max intensity below min_intensity or spatial extend below min_number_cells

Parameters

- **min_intensity** (*int or float*) – Minimum criterion for intensity
- **min_number_cells** (*int or float*) – Minimum crietrion for number of grid cell

Return type
Hazard

1.3.5 climada_petals.hazard.relative_cropyield module

class climada_petals.hazard.relative_cropyield.**RelativeCropyield**(*crop: str = ''*, *intensity_def: str = 'Yearly Yield'*, ***kwargs*)

Bases: `Hazard`

Agricultural climate risk: Relative Cropyield (relative to historical mean); Each year corresponds to one hazard event; Based on modelled crop yield, from ISIMIP (www.isimip.org, required input data). Attributes as defined in Hazard and the here defined additional attributes.

crop_type

crop type ('whe' for wheat, 'mai' for maize, 'soy' for soybeans and 'ric' for rice)

Type
str

intensity_def

intensity defined as: 'Yearly Yield' [t/(ha*y)], 'Relative Yield', or 'Percentile'

Type
str

__init__(*crop: str = ''*, *intensity_def: str = 'Yearly Yield'*, ***kwargs*)

Initialize values.

Parameters

- **crop_type** (*str, optional*) – crop type ('whe' for wheat, 'mai' for maize, 'soy' for soybeans and 'ric' for rice). Default: ''
- **intensity_def** (*str, optional*) – intensity defined as: 'Yearly Yield' [t/(ha*y)], 'Relative Yield', or 'Percentile' Default: 'Yearly Yield'
- ****kwargs** (*Hazard properties, optional*) – All other keyword arguments are passed to the Hazard constructor.

set_from_isimip_netcdf(**args*, ***kwargs*)

This function is deprecated, use RelativeCropyield.from_isimip_netcdf instead.

classmethod from_isimip_netcdf(*input_dir=None*, *filename=None*, *bbox=None*, *yearrange=None*, *ag_model=None*, *cl_model=None*, *bias_corr=None*, *scenario=None*, *soc=None*, *co2=None*, *crop=None*, *irr=None*, *fn_str_var=None*)

Wrapper to fill hazard from crop yield NetCDF file. Build and tested for output from ISIMIP2 and ISIMIP3, but might also work for other NetCDF containing gridded crop model output from other sources.

Parameters

- **input_dir** (*Path or str*) – path to input data directory, default: {CONFIG.exposures.crop_production.local_data}/Input/Exposure
- **filename** (*string*) – name of netcdf file in input_dir. If filename is given, the other parameters specifying the model run are not required!
- **bbox** (*list of four floats*) – bounding box: [lon min, lat min, lon max, lat max]
- **yearrange** (*int tuple*) – year range for hazard set, f.i. (1976, 2005)

- **ag_model** (*str*) – abbrev. agricultural model (only when input_dir is selected) f.i. 'clm-crop', 'gepic','lpjml','pepic'
- **cl_model** (*str*) – abbrev. climate model (only when input_dir is selected) f.i. ['gfdl-esm2m', 'hadgem2-es','ipsl-cm5a-lr','miroc5'
- **bias_corr** (*str*) – bias correction of climate forcing, f.i. 'ewembi' (ISIMIP2b, default) or 'w5e5' (ISIMIP3b)
- **scenario** (*str*) – climate change scenario (only when input_dir is selected) f.i. 'historical' or 'rcp60' or 'ISIMIP2a'
- **soc** (*str*) – socio-economic trajectory (only when input_dir is selected) f.i. '2005soc' or 'histsoc'
- **co2** (*str*) – CO2 forcing scenario (only when input_dir is selected) f.i. 'co2' or '2005co2'
- **crop** (*str*) – crop type (only when input_dir is selected) f.i. 'whe', 'mai', 'soy' or 'ric'
- **irr** (*str*) – irrigation type (only when input_dir is selected) f.i 'noirr' or 'irr'
- **fn_str_var** (*str*) – FileName STRing depending on VARiable and ISIMIP simuation round

Return type
RelativeCropyield

Raises
NameError –

calc_mean (*yearrange_mean=None*, *save=False*, *output_dir=None*)

Calculates mean of the hazard for a given reference time period

Parameters

- **yearrange_mean** (*array*) – time period used to calculate the mean intensity default: 1976-2005 (historical)
- **save** (*boolean*) – save mean to file? default: False
- **output_dir** (*str or Path*) – path of output directory, default: {CONFIG.exposures.crop_production.local_data}/Output

Returns
contains mean value over the given reference time period for each centroid

Return type
hist_mean(array)

set_rel_yield_to_int (**args*, ***kwargs*)

This function is deprecated, use function rel_yield_to_int instead.

set_percentile_to_int (**args*, ***kwargs*)

This function is deprecated, use function percentile_to_int instead.

plot_intensity_cp (*event=None*, *dif=False*, *axis=None*, ***kwargs*)

Plots intensity with predefined settings depending on the intensity definition

Parameters

- **event** (*int or str*) – event_id or event_name
- **dif** (*boolean*) – variable signilizing whether absolute values or the difference between future and historic are plotted (False: his/fut values; True: difference = fut-his)
- **axis** (*geoaxes*) – axes to plot on

Return type
axes (geoaxes)

plot_time_series(*event=None*)

Plots a time series of intensities (a series of sub plots)

Parameters
event (*int or str*) – event_id or event_name

Return type
figure

1.3.6 climada_petals.hazard.river_flood module

class climada_petals.hazard.river_flood.**RiverFlood**(**args*, ***kwargs*)

Bases: Hazard

Contains flood events Flood intensities are calculated by means of the CaMa-Flood global hydrodynamic model

fla_event

total flooded area for every event

Type
1d array(n_events)

fla_annual

total flooded area for every year

Type
1d array (n_years)

fla_ann_av

average flooded area per year

Type
float

fla_ev_av

average flooded area per event

Type
float

fla_ann_centr

flooded area in every centroid for every event

Type
2d array(n_years x n_centroids)

fla_ev_centr

flooded area in every centroid for every event

Type
2d array(n_events x n_centroids)

__init__(**args*, ***kwargs*)

Empty constructor

classmethod from_nc(*dph_path=None*, *frc_path=None*, *origin=False*, *centroids=None*, *countries=None*, *reg=None*, *shape=None*, *ISINatIDGrid=False*, *years=None*)

Wrapper to fill hazard from nc_flood file

Parameters

- **dph_path** (*str, optional*) – Flood file to read (depth)
- **frc_path** (*str, optional*) – Flood file to read (fraction)
- **origin** (*bool, optional*) – Historical or probabilistic event. Default: False
- **centroids** (*Centroids, optional*) – centroids to extract
- **countries** (*list of str, optional*) – If *reg* is None, use this selection of countries (ISO3). Default: None
- **reg** (*list of str, optional*) – Use region code to consider whole areas. If not None, countries and centroids are ignored. Default: None
- **shape** (*str or Path, optional*) – If *reg* and *countries* are None, use the first geometry in this shape file to cut out the area of interest. Default: None
- **ISINatIDGrid** (*bool, optional*) – Indicates whether ISIMIP_NatIDGrid is used. Default: False
- **years** (*list of int*) – Years that are considered. Default: None

Returns
haz

Return type
RiverFlood instance

Raises
NameError –

set_from_nc(**args*, ***kwargs*)

This function is deprecated, use RiverFlood.from_nc instead.

exclude_trends(*fld_trend_path*, *dis*)

Function allows to exclude flood impacts that are caused in areas exposed discharge trends other than the selected one. (This function is only needed for very specific applications)

Raises
NameError –

exclude_returnlevel(*frc_path*)

Function allows to exclude flood impacts below a certain return level by manipulating flood fractions in a way that the array flooded more frequently than the treshold value is excluded. (This function is only needed for very specific applications)

Raises
NameError –

set_flooded_area(*save_centr=False*)

Calculates flooded area for hazard. sets yearly flooded area and
flooded area per event

Raises
MemoryError –

set_flood_volume(*save_centr=False*)

Calculates flooded area for hazard. sets yearly flooded area and
flooded area per event

Raises
MemoryError –

1.3.7 climada_petals.hazard.tc_rainfield module

class climada_petals.hazard.tc_rainfield.**TCRain**(*category: ndarray | None = None, basin: List | None = None, rainrates: List[csr_matrix] | None = None, **kwargs*)

Bases: Hazard

Contains rainfall from tropical cyclone events.

category

for every event, the TC category using the Saffir-Simpson scale:

- -1 tropical depression
- 0 tropical storm
- 1 Hurrican category 1
- 2 Hurrican category 2
- 3 Hurrican category 3
- 4 Hurrican category 4
- 5 Hurrican category 5

Type
np.ndarray of ints

basin

Basin where every event starts:

- 'NA' North Atlantic
- 'EP' Eastern North Pacific
- 'WP' Western North Pacific
- 'NI' North Indian
- 'SI' South Indian
- 'SP' Southern Pacific
- 'SA' South Atlantic

Type
list of str

rainrates

For each event, the rain rates (in mm/h) at each centroid and track position in a sparse matrix of shape (npositions, ncentroids).

Type
list of csr_matrix

intensity_thres = 0.1

intensity threshold for storage in mm

vars_opt = {'category'}

Name of the variables that aren't needed to compute the impact.

__init__ (*category: ndarray | None = None*, *basin: List | None = None*, *rainrates: List[csr_matrix] | None = None*, ***kwargs*)

Initialize values.

Parameters

- **category** (*np.ndarray of int, optional*) –

  For every event, the TC category using the Saffir-Simpson scale:
  -1 tropical depression 0 tropical storm 1 Hurrican category 1 2 Hurrican category 2 3 Hurrican category 3 4 Hurrican category 4 5 Hurrican category 5

- **basin** (*list of str, optional*) –

  Basin where every event starts:
  'NA' North Atlantic 'EP' Eastern North Pacific 'WP' Western North Pacific 'NI' North Indian 'SI' South Indian 'SP' Southern Pacific 'SA' South Atlantic

- **rainrates** (*list of csr_matrix, optional*) – For each event, the rain rates (in mm/h) at each centroid and track position in a sparse matrix of shape (npositions, ncentroids).

- ****kwargs** (*Hazard properties, optional*) – All other keyword arguments are passed to the Hazard constructor.

set_from_tracks (**args*, ***kwargs*)

This function is deprecated, use TCRain.from_tracks instead.

classmethod from_tracks (*tracks: TCTracks*, *centroids: Centroids | None = None*, *pool: ProcessPool | None = None*, *model: str = 'R-CLIPER'*, *model_kwargs: dict | None = None*, *ignore_distance_to_coast: bool = False*, *store_rainrates: bool = False*, *metric: str = 'equirect'*, *intensity_thres: float = 0.1*, *max_latitude: float = 61*, *max_dist_inland_km: float = 1000*, *max_dist_eye_km: float = 300*, *max_memory_gb: float = 8*)

Create new TCRain instance that contains rainfields from the specified tracks

This function sets the *intensity* attribute to contain, for each centroid, the total amount of rain experienced over the whole period of each TC event in mm. The amount of rain is set to 0 if it does not exceed the threshold *intensity_thres*.

The *category* attribute is set to the value of the *category*-attribute of each of the given track data sets.

The *basin* attribute is set to the genesis basin for each event, which is the first value of the *basin*-variable in each of the given track data sets.

Optionally, the time-dependent rain rates can be stored using the *store_rainrates* function parameter (see below).

Currently, two models are supported to compute the rain rates: R-CLIPER and TCR. The R-CLIPER model is documented in Tuleya et al. 2007. The TCR model was used by Zhu et al. 2013 and Emanuel 2017 for the

first time and is documented in detail in Lu et al. 2018. This implementation of TCR includes improvements proposed in Feldmann et al. 2019. TCR's accuracy is much higher than R-CLIPER's at the cost of additional computational and data requirements.

When using the TCR model make sure that your TC track data includes the along-track variables "t600" (temperature at 600 hPa) and "u850"/"v850" (wind speed at 850 hPa). Both can be extracted from reanalysis or climate model outputs. For "t600", use the value at the storm center. For "u850"/"v850", use the average over the 200-500 km annulus around the storm center. If "u850"/"v850" is missing, this implementation sets the shear component of the vertical velocity to 0. If "t600" is missing, the saturation specific humidity is set to a universal estimate of 0.01 kg/kg. Both assumptions can have a large effect on the results (see Lu et al. 2018).

Emanuel (2017): Assessing the present and future probability of Hurricane Harvey's rainfall. Proceedings of the National Academy of Sciences 114(48): 12681–12684. https://doi.org/10.1073/pnas.1716222114

Lu et al. (2018): Assessing Hurricane Rainfall Mechanisms Using a Physics-Based Model: Hurricanes Isabel (2003) and Irene (2011). Journal of the Atmospheric Sciences 75(7): 2337–2358. https://doi.org/10.1175/JAS-D-17-0264.1

Feldmann et al. (2019): Estimation of Atlantic Tropical Cyclone Rainfall Frequency in the United States. Journal of Applied Meteorology and Climatology 58(8): 1853–1866. https://doi.org/10.1175/JAMC-D-19-0011.1

Tuleya et al. (2007): Evaluation of GFDL and Simple Statistical Model Rainfall Forecasts for U.S. Landfalling Tropical Storms. Weather and Forecasting 22(1): 56–70. https://doi.org/10.1175/WAF972.1

Zhu et al. (2013): Estimating tropical cyclone precipitation risk in Texas. Geophysical Research Letters 40(23): 6225–6230. https://doi.org/10.1002/2013GL058284

Parameters

- **tracks** (*climada.hazard.TCTracks*) – Tracks of storm events.
- **centroids** (*Centroids, optional*) – Centroids where to model TC. Default: global centroids at 360 arc-seconds resolution.
- **pool** (*pathos.pool, optional*) – Pool that will be used for parallel computation of rain fields. Default: None
- **model** (*str, optional*) – Parametric rain model to use: "R-CLIPER" (faster and requires less inputs, but much less accurate, statistical approach, Tuleya et al. 2007), "TCR" (physics-based approach, requires non-standard along-track variables, Zhu et al. 2013). Default: "R-CLIPER".
- **model_kwargs** (*dict, optional*) – If given, forward these kwargs to the selected model. The implementation of the R-CLIPER model currently does not allow modifications, so that *model_kwargs* is ignored with *model="R-CLIPER"*. While the TCR model can be configured in several ways, it is usually safe to go with the default settings. Here is the complete list of *model_kwargs* and their meaning with *model="TCR"* (in alphabetical order):

  c_drag_tif
  [Path or str, optional] Path to a GeoTIFF file containing gridded drag coefficients (bottom friction). If not specified, an ERA5-based data set provided with CLIMADA is used. Default: None

  e_precip
  [float, optional] Precipitation efficiency (unitless), the fraction of the vapor flux falling to the surface as rainfall (Lu et al. 2018, eq. (14)). Note that we follow the MATLAB reference implementation and use 0.5 as a default value instead of the 0.9 that was proposed in Lu et al. 2018. Default: 0.5

elevation_tif
[Path or str, optional] Path to a GeoTIFF file containing digital elevation model data (in m). If not specified, an SRTM-based topography at 0.1 degree resolution provided with CLIMADA is used. Default: None

matlab_ref_mode
[bool, optional] This implementation is based on a (proprietary) reference implementation in MATLAB. However, some (minor) changes have been applied in the CLIMADA implementation compared to the reference:

- In the computation of horizontal wind speeds, we compute the Coriolis parameter from latitude. The MATLAB code assumes a constant parameter value (5e-5).
- As a rescaling factor from surface to gradient winds, we use a factor from the literature. The factor in MATLAB is very similar, but does not specify a source.
- Instead of the "specific humidity", the (somewhat simpler) formula for the "mixing ratio" is used in the MATLAB code. These quantities are almost the same in practice.
- We use the approximation of the Clausius-Clapeyron equation used by the ECMWF (Buck 1981) instead of the one used in the MATLAB code (Bolton 1980).

Since it might be useful to have a version that replicates the behavior of the reference implementation, this parameter can be set to True to enforce the exact behavior of the reference implementation. Default: False

max_w_foreground
[float, optional] The maximum value (in m/s) at which to clip the vertical velocity w before subtracting the background subsidence velocity w_rad. Default: 7.0

min_c_drag
[float, optional] The drag coefficient is clipped to this minimum value (esp. over ocean). Default: 0.001

q_950
[float, optional] If the track data does not include "t600" values, assume this constant value of saturation specific humidity (in kg/kg) at 950 hPa. Default: 0.01

res_radial_m
[float, optional] Resolution (in m) in radial direction. This is used for the computation of discrete derivatives of the horizontal wind fields and derived quantities. Default: 2000.0

w_rad
[float, optional] Background subsidence velocity (in m/s) under radiative cooling. Default: 0.005

wind_model
[str, optional] Parametric wind field model to use, see the *TropCyclone* class. Default: "ER11".

Default: None

- **ignore_distance_to_coast** (*boolean, optional*) – If True, centroids far from coast are not ignored. Default: False.
- **store_rainrates** (*boolean, optional*) – If True, the Hazard object gets a list *rainrates* of sparse matrices. For each track, the rain rates (in mm/h) at each centroid and track position are stored in a sparse matrix of shape (npositions, ncentroids). Default: False.
- **metric** (*str, optional*) – Specify an approximation method to use for earth distances:

– "equirect": Distance according to sinusoidal projection. Fast, but inaccurate for large distances and high latitudes.

– "geosphere": Exact spherical distance. Much more accurate at all distances, but slow.

Default: "equirect".

- **intensity_thres** (*float, optional*) – Rain amounts (in mm) below this threshold are stored as 0. Default: 0.1
- **max_latitude** (*float, optional*) – No rain calculation is done for centroids with latitude larger than this parameter. Default: 61
- **max_dist_inland_km** (*float, optional*) – No rain calculation is done for centroids with a distance (in km) to the coast larger than this parameter. Default: 1000
- **max_dist_eye_km** (*float, optional*) – No rain calculation is done for centroids with a distance (in km) to the TC center ("eye") larger than this parameter. Default: 300
- **max_memory_gb** (*float, optional*) – To avoid memory issues, the computation is done for chunks of the track sequentially. The chunk size is determined depending on the available memory (in GB). Note that this limit applies to each thread separately if a *pool* is used. Default: 8

Return type
TCRain

1.3.8 climada_petals.hazard.tc_surge_bathtub module

class climada_petals.hazard.tc_surge_bathtub.**TCSurgeBathtub**

Bases: Hazard

TC surge heights in m, a bathtub model with wind-surge relationship and inland decay.

__init__()

Initialize values.

Parameters

- **haz_type** (*str, optional*) – acronym of the hazard type (e.g. 'TC').
- **pool** (*pathos.pool, optional*) – Pool that will be used for parallel computation when applicable. Default: None
- **units** (*str, optional*) – units of the intensity. Defaults to empty string.
- **centroids** (*Centroids, optional*) – centroids of the events. Defaults to empty Centroids object.
- **event_id** (*np.array, optional*) – id (>0) of each event. Defaults to empty array.
- **event_name** (*list(str), optional*) – name of each event (default: event_id). Defaults to empty list.
- **date** (*np.array, optional*) – integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library). Defaults to empty array.
- **orig** (*np.array, optional*) – flags indicating historical events (True) or probabilistic (False). Defaults to empty array.
- **frequency** (*np.array, optional*) – frequency of each event. Defaults to empty array.

- **frequency_unit** (*str, optional*) – unit of the frequency (default: "1/year").
- **intensity** (*sparse.csr_matrix, optional*) – intensity of the events at centroids. Defaults to empty matrix.
- **fraction** (*sparse.csr_matrix, optional*) – fraction of affected exposures for each event at each centroid. Defaults to empty matrix.

Examples

Initialize using keyword arguments:

```
>>> haz = Hazard('TC', intensity=sparse.csr_matrix(np.zeros((2, 2))))
```

Take hazard values from file:

```
>>> haz = Hazard.from_mat(HAZ_DEMO_MAT, 'demo')
```

static from_tc_winds(*wind_haz*, *topo_path*, *inland_decay_rate=0.2*, *add_sea_level_rise=0.0*)

Compute tropical cyclone surge from input winds.

Parameters

- **wind_haz** (*TropCyclone*) – Tropical cyclone wind hazard object.
- **topo_path** (*str*) – Path to a raster file containing gridded elevation data.
- **inland_decay_rate** (*float, optional*) – Decay rate of surge when moving inland in meters per km. Set to 0 to deactivate this effect. The default value of 0.2 is taken from Section 5.2.1 of the monograph Pielke and Pielke (1997): Hurricanes: their nature and impacts on society. https://rogerpielkejr.com/2016/10/10/hurricanes-their-nature-and-impacts-on-society/
- **add_sea_level_rise** (*float, optional*) – Sea level rise effect in meters to be added to surge height.

1.3.9 climada_petals.hazard.tc_tracks_forecast module

class climada_petals.hazard.tc_tracks_forecast.**TCForecast**(*data: List[Dataset] | None = None*, *pool: ProcessPool | None = None*)

Bases: `TCTracks`

An extension of the TCTracks construct adapted to forecast tracks obtained from numerical weather prediction runs.

data

Same as in parent class, adding the following attributes

- ensemble_member (int)
- is_ensemble (bool; if False, the simulation is a high resolution deterministic run)
- run_datetime (numpy.datetime64): timepoint of the initialisation of the numerical weather prediction run

Type
list of xarray.Dataset

fetch_ecmwf (*path=None*, *files=None*, *target_dir=None*, *remote_dir=None*)

Fetch and read latest ECMWF TC track predictions from the FTP dissemination server into instance. Use path or files argument to use local files instead.

Assumes file naming conventions consistent with ECMWF: all files are assumed to have 'tropical_cyclone' and 'ECEP' in their name, denoting tropical cyclone ensemble forecast files.

Parameters

- **path** (*str, list(str), optional*) – A location in the filesystem. Either a path to a single BUFR TC track file, or a folder containing only such files, or a globbing pattern. Passed to climada.util.files_handler.get_file_names
- **files** (*file-like, optional*) – An explicit list of file objects, bypassing get_file_names
- **target_dir** (*str, optional*) – An existing directory in the filesystem. When set, downloaded BUFR files will be saved here, otherwise they will be downloaded as temporary files.
- **remote_dir** (*str, optional*) – If set, search the ECMWF FTP folder for forecast files in the directory; otherwise defaults to the latest. Format: yyyymmddhhmmss, e.g. 20200730120000

static fetch_bufr_ftp (*target_dir=None*, *remote_dir=None*)

Fetch and read latest ECMWF TC track predictions from the FTP dissemination server. If target_dir is set, the files get downloaded persistently to the given location. A list of opened file-like objects gets returned.

Parameters

- **target_dir** (*str*) – An existing directory to write the files to. If None, the files get returned as tempfiles.
- **remote_dir** (*str, optional*) – If set, search this ftp folder for forecast files; defaults to the latest. Format: yyyymmddhhmmss, e.g. 20200730120000

Return type
[filelike]

read_one_bufr_tc (*file*, *id_no=None*)

Read a single BUFR TC track file tailored to the ECMWF TC track predictions format.

Parameters

- **file** (*str, filelike*) – Path object, string, or file-like object
- **id_no** (*int*) – Numerical ID; optional. Else use date + random int.

write_hdf5 (*file_name*, *complevel=5*)

Write TC tracks in NetCDF4-compliant HDF5 format. This method overrides the method of the base class.

Parameters

- **file_name** (*str or Path*) – Path to a new HDF5 file. If it exists already, the file is overwritten.
- **complevel** (*int, optional*) – Specifies a compression level (0-9) for the zlib compression of the data. A value of 0 or None disables compression. Default: 5

classmethod from_hdf5 (*file_name*)

Create new TCTracks object from a NetCDF4-compliant HDF5 file. This method overrides the method of the base class.

Parameters
file_name (*str or Path*) – Path to a file that has been generated with *TCForecast.write_hdf*.

Returns
tracks – TCTracks with data from the given HDF5 file.

Return type
TCForecast

classmethod read_cxml (*cxml_path: str*, *xsl_path: str | None = None*)

Reads a cxml (cyclone xml) file and returns a class instance.

Parameters

- **cxml_path** (*str*) – Path to the cxml file
- **xsl_path** (*str, optional*) – Path to the xsl tranformation file needed to read the cxml data. Default: None

Returns
TCTracks with data from the given cxml file.

Return type
TCForecast

1.3.10 climada_petals.hazard.wildfire module

class climada_petals.hazard.wildfire.**WildFire**

Bases: `Hazard`

Contains wild fire events.

Wildfires comprise the challenge that the definition of an event is unclear. Reporting standards vary accross regions and over time. Hence, to have consistency, we consider an event as a whole fire season. A fire season is defined as a whole year (Jan-Dec in the NHS, Jul-Jun in SHS). This allows consistent risk assessment across the globe and over time. Hazard for which events refer to a fire season have the haz_type 'WFseason'.

In order to perform concrete case studies or calibrate impact functions, events can be displayed as single fires. In that case they have the haz_type 'WFsingle'.

date_end
integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library). Represents last day of a wild fire instance where the fire was still active.

Type
array

n_fires
number of single fires in a fire season

Type
array

__init__()
Empty constructor.

class FirmsParams (*clean_thresh: int = 30*, *days_thres_firms: int = 2*, *clus_thres_firms: int = 15*, *remove_minor_fires_firms: bool = True*, *minor_fire_thres_firms: int = 3*)

Bases: `object`

DataClass as container for firms parameters.

clean_thresh

Minimal confidence value for the data from MODIS instrument to be use as input

Type
int, default = 30

days_thres_firms

Minimum number of days to consider different fires

Type
int, default = 2

clus_thres_firms

Clustering factor which multiplies instrument resolution

Type
int, default = 15

remove_minor_fires_firms

removes FIRMS fires below defined theshold of entries

Type
bool, default = True

minor_fire_thres_firms

number of FIRMS entries required to be considered a fire

Type
int, default = 3

clean_thresh: int = 30

days_thres_firms: int = 2

clus_thres_firms: int = 15

remove_minor_fires_firms: bool = True

minor_fire_thres_firms: int = 3

__init__(*clean_thresh: int = 30*, *days_thres_firms: int = 2*, *clus_thres_firms: int = 15*, *remove_minor_fires_firms: bool = True*, *minor_fire_thres_firms: int = 3*) → None

class ProbaParams(*blurr_steps: int = 4*, *prop_proba: float = 0.21*, *max_it_propa: int = 500000*)

Bases: `object`

Dataclass as container for parameters for generation of probabilistic events.

PLEASE BE AWARE: Parameter values did not undergo any calibration.

blurr_steps

steps with exponential decay for fire propagation matrix

Type
int, default = 4

prop_proba

Type
float, default = 0.21

max_it_propa

Type
int, default = 500000

blurr_steps: int = 4

prop_proba: float = 0.21

max_it_propa: int = 500000

__init__ (*blurr_steps: int = 4*, *prop_proba: float = 0.21*, *max_it_propa: int = 500000*) → None

classmethod from_hist_fire_FIRMS (*df_firms*, *centr_res_factor=1.0*, *centroids=None*)

Parse FIRMS data and generate historical fires by temporal and spatial clustering. Single fire events are defined as a set of data points that are geographically close and/or have consecutive dates. The unique identification is made in two steps. First a temporal clustering is applied to cleaned data obtained from FIRMS. Data points with acquisition dates more than days_thres_firms days apart are in different temporal clusters. Second, for each temporal cluster, unique event are identified by performing a spatial clustering. This is done iteratively until all firms data points are assigned to an event.

This method sets the attributes self.n_fires, self.date_end, in addition to all attributes required by the hazard class.

This method creates a centroids raster if centroids=None with resolution given by centr_res_factor. The centroids can be retrieved from Wildfire.centroids()

Parameters

- **df_firms** (*pd.DataFrame*) – FIRMS data as pd.Dataframe (https://firms.modaps.eosdis.nasa.gov/download/)
- **centr_res_factor** (*float, optional, default=1.0*) – resolution factor with respect to the satellite data to use for centroids creation. Hence, if MODIS data (1 km res) is used and centr_res_factor is set to 0.2, the grid spacing of the generated centroids will equal 5 km (=1/0.2). If centroids are defined, this parameter has no effect.
- **centroids** (*Centroids, optional*) – centroids in degrees to map data, centroids need to be on a regular raster grid in order for the clustrering to work.

Returns
haz

Return type
WildFire instance

set_hist_fire_FIRMS (**args*, ***kwargs*)

This function is deprecated, use WildFire.from_hist_fire_FIRMS instead.

classmethod from_hist_fire_seasons_FIRMS (*df_firms*, *centr_res_factor=1.0*, *centroids=None*, *hemisphere=None*, *year_start=None*, *year_end=None*, *keep_all_fires=False*)

Parse FIRMS data and generate historical fire seasons.

Individual fires are created using temporal and spatial clustering according to the 'set_hist_fire_FIRMS' method. single fires are then summarized to seasons using max intensity at each centroid for each year.

This method sets the attributes self.n_fires, self.date_end, in addition to all attributes required by the hazard class.

This method creates a centroids raster if centroids=None with resolution given by centr_res_factor. The centroids can be retrieved from Wildfire.centroids()

Parameters

- **df_firms** (*pd.DataFrame*) – FIRMS data as pd.Dataframe (https://firms.modaps.eosdis.nasa.gov/download/)
- **centr_res_factor** (*float, optional, default=1.0*) – resolution factor with respect to the satellite data to use for centroids creation
- **centroids** (*Centroids, optional*) – centroids in degrees to map data, centroids need to be on a regular grid in order for the clustrering to work.
- **hemisphere** (*str, optional*) – 'SHS' or 'NHS' to define fire seasons. The hemisphere parameter is only used for the definition of the start of the fire season
- **year_start** (*int, optional*) – start year; FIRMS fires before that are cut; no cut if not specified
- **year_end** (*int, optional*) – end year; FIRMS fires after that are cut; no cut if not specified
- **keep_all_fires** (*bool, optional*) – keep list of all individual fires; default is False to save memory. If set to true, fires are stored in self.hist_fire_seasons

Returns
haz

Return type
WildFire instance

set_hist_fire_seasons_FIRMS (**args*, ***kwargs*)

This function is deprecated, use WildFire.from_hist_fire_seasons_FIRMS instead.

set_proba_fire_seasons (*n_fire_seasons=1*, *n_ignitions=None*, *keep_all_fires=False*)

Generate probabilistic fire seasons.

Fire seasons are created by running n probabilistic fires per year which are then summarized into a probabilistic fire season by calculating the max intensity at each centroid for each probabilistic fire season. Probabilistic fires are created using the logic described in the method '_run_one_bushfire'.

The fire propagation matrix can be assigned separately, if that is not done it will be generated on the available historic fire (seasons).

Intensities are drawn randomly from historic events. Thus, this method requires at least one fire to draw from.

This method modifies self (climada.hazard.WildFire instance) by adding probabilistic wildfire seasons.

Parameters

- **self** (*climada.Hazard.WildFire*) – must have calculated historic fire seasons before
- **n_fire_seasons** (*int, optional*) – number of fire seasons to be generated
- **n_ignitions** (*array, optional*) – [min, max]: min/max of uniform distribution to sample from, in order to determin n_fire per probabilistic year set. If none, min/max is taken from hist.
- **keep_all_fires** (*bool, optional*) – keep detailed list of all fires; default is False to save memory.

combine_fires (*event_id_merge=None*, *remove_rest=False*, *probabilistic=False*)

Combine events that are identified as different fire to one event

Orig fires are removed and a new fire id created; max intensity at overlapping centroids is assigned.

This method modifies self (climada.hazard.WildFire instance) by combining single fires.

Parameters

- **event_id_merge** (*array of int, optional*) – events to be merged
- **remove_rest** (*bool, optional*) – if set to true, only the merged event is returned.
- **probabilistic** (*bool, optional*) – differentiate, because probabilistic events have no date.

summarize_fires_to_seasons(*year_start=None*, *year_end=None*, *hemisphere=None*)

Summarize historic fires into fire seasons.

Fires are summarized by taking the max intensity at each grid point.

This method modifies self (climada.hazard.WildFire instance) by summarizing individual fires into seasons.

Parameters

- **year_start** (*int, optional*) – start year; fires before that are cut; no cut if not specified
- **year_end** (*int, optional*) – end year; fires after that are cut; no cut if not specified
- **hemisphere** (*str, optional*) – 'SHS' or 'NHS' to define fire seasons

plot_fire_prob_matrix()

Plots fire propagation probability matrix as contour plot. At this point just to check the matrix but could easily be improved to normal map.

Parameters
self (*climada.hazard.WildFire instance*)

Returns
contour plot – contour plot of fire_propa_matrix

Return type
plt

1.4 climada_petals.util package

1.4.1 climada_petals.util.config module

1.4.2 climada_petals.util.constants module

climada_petals.util.constants.**HAZ_DEMO_FLDDPH** = **PosixPath('/home/docs/climada/demo/data/flddph_2000_DEMO.nc')**

NetCDF4 Flood depth from isimip simulations

climada_petals.util.constants.**HAZ_DEMO_FLDFRC** = **PosixPath('/home/docs/climada/demo/data/fldfrc_2000_DEMO.nc')**

NetCDF4 Flood fraction from isimip simulations

climada_petals.util.constants.**DEMO_GDP2ASSET** = **PosixPath('/home/docs/climada/demo/data/gdp2asset_CHE_exposure.nc')**

Exposure demo file for GDP2Asset

CHAPTER

TWO

HAZARD TUTORIALS

2.1 Hazard Emulator

Given a database of hazard events, the module `climada.hazard.emulator` provides tools to subsample events (or time series of events) from that event database. The module provides functionality to guard the subsampling, e.g., using bias-corrected statistics according to historical records in a specific georegion, or using calibrated statistics according to a climate scenario.

In more complex cases, the given event database is divided into a (smaller) set of observed hazard events and a (much larger) set of simulated hazard events. The database of observed events is used to statistically fit the frequency and intensity of events in a fixed georegion to (observed) climate indices. Then, given a hypothetical (future) time series of these climate indices (a "climate scenario"), a "hazard emulator" can draw random samples from the larger database of simulated hazard events that mimic the expected occurrence of events under the given climate scenario in the specified georegion.

The concept and algorithm as applied to tropical cyclones is originally due to Tobias Geiger (unpublished as of now) and has been generalized within this package by Thomas Vogt.

This notebook illustrates the functionality through the example of tropical cyclones in the eastern pacific under the RCP 2.6 climate scenario according to the MIROC5 global circulation model (GCM). However, the algorithm can be applied to arbitrary Hazard types given a suitable database of synthetic events.

2.1.1 Load hazard data

The database of hazard events for this tutorial is too large to be provided with it. Instructions on how to obtain or generate the data are provided in the last section of this tutorial. At this point, we load a precomputed `Hazard` object from a file that contains simulated tropical cyclone tracks for an RCP 2.6 climate scenario according to MIROC5. Note, however, that this module will work with any `Hazard` object, preferably containing a lot of events.

[1]:
```
from climada.util.config import CONFIG
DEMO_DIR = CONFIG.local_data.demo.dir(create=False)
EMULATOR_DATA_DIR = DEMO_DIR.joinpath("emulator")
```

[2]:
```
from climada.hazard import TropCyclone
hazard = TropCyclone.from_hdf5(EMULATOR_DATA_DIR.joinpath("hazard_360as_miroc_rcp26.
↪hdf5"))
```

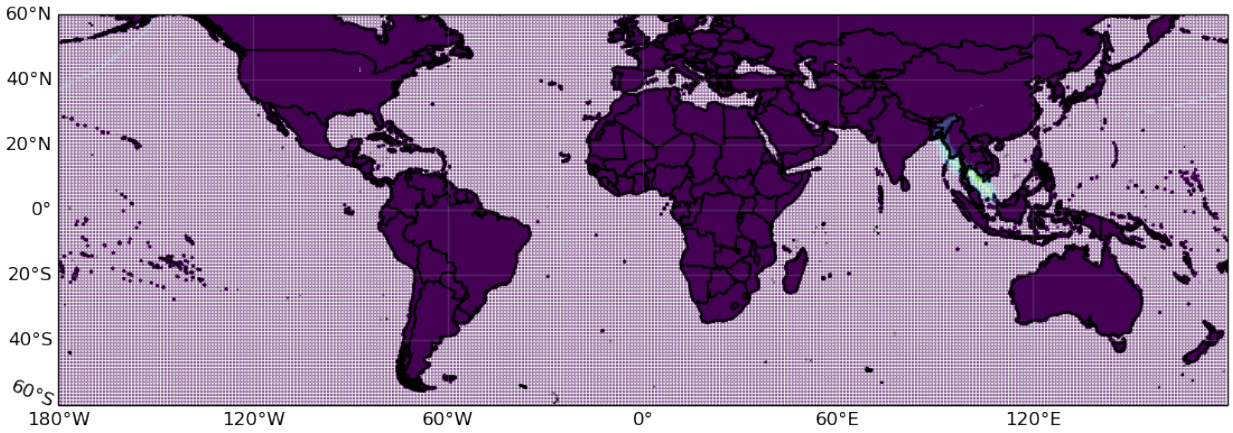
```
2021-03-24 17:46:31,484 - climada.hazard.base - INFO - Reading /home/tovogt/.climada/
↪demo/data/emulator/hazard_360as_miroc_rcp26.hdf5
```

We quickly try to give an impression of what this particular `Hazard` object looks like. It contains `hazard.size = 45300` tropical cyclone wind fields that are computed on a global set of centroids with 360 arc-seconds (0.1 degree)

resolution onland and 1 degree resolution offshore. The physical effects onland are considered more important, the low-resolution information offshore is mostly relevant for plotting purposes:

[3]:
```
import warnings; warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import pyproj
plt.rcParams['figure.dpi'] = 120
hazard.centroids.geometry = hazard.centroids.geometry.to_crs(pyproj.CRS("EPSG:4326"))
hazard.centroids.plot(c=hazard.intensity[16637,:].toarray().ravel(), s=0.1);
```

[3]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x7f74c8bb0a60>
```



2.1.2 Define the region of interest

One basic feature of the `climada.hazard.emulator` module is the `HazRegion` class that not only defines the geographical region of interest for the statistics, but might also contain climatic information about the selected region, such as the cyclone (hurricane) season. For tropical cyclone hazards there already exists a derived class `TCRegion` that defines ocean basins and cyclone seasons.

[4]:
```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from climada_petals.hazard.emulator.geo import TCRegion
# load Eastern Pacific basin, print season (months of year) and plot geometry
region = TCRegion(tc_basin="EP")
print(f"\nThe cyclone season in basin {region.tc_basin} spans "
      f"from month {region.season[0]} until month {region.season[1]}.\n")

ax = plt.gcf().add_axes([0, 0, 1, 1], projection=ccrs.PlateCarree())
ax.add_feature(cfeature.COASTLINE.with_scale('110m'), linewidth=0.75)
ax.add_geometries([region.shape], crs=ccrs.PlateCarree(), alpha=0.8);
```

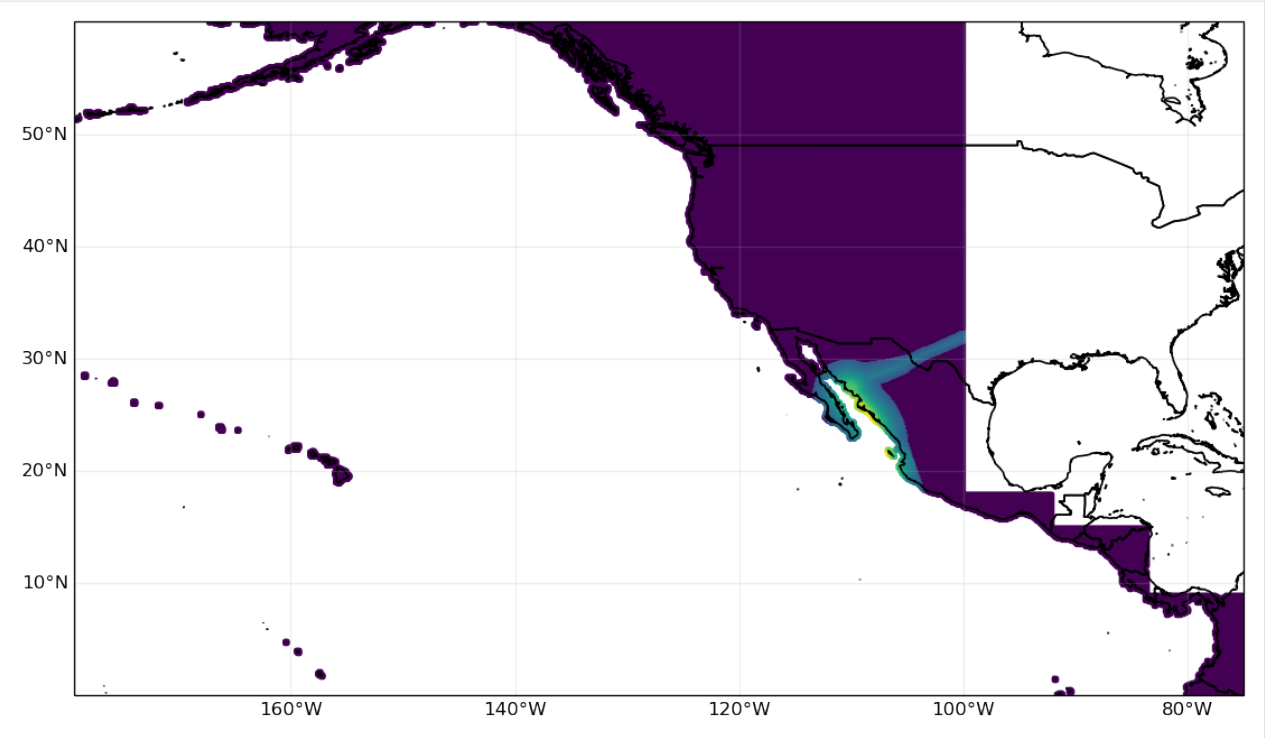
```
The cyclone season in basin EP spans from month 7 until month 12.
```

[4]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f74985581f0>
```

For our analysis, we restrict to the hazard events that affect the land area (defined by the `on_land` property) in this region:

[5]:
```
import shapely
hazard.centroids.region_id = shapely.vectorized.contains(
    region.shape, hazard.centroids.lon, hazard.centroids.lat) & hazard.centroids.on_
↪land.astype(bool)
hazard_EP = hazard.select(reg_id=1)
# Plot one event as an example:
hazard_EP.centroids.plot(c=hazard_EP.intensity[31577,:].toarray().ravel(), s=3);
```

[5]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x7f7498568ee0>
```



2.1.3 Extract events that affect the region of interest

The emulator's statistics and sampling functionality doesn't use the Hazard object directly, but an aggregated version of it (a pandas `DataFrame`). From the Hazard objects, we extract those events that actually "affect" the georegion of interest and store for each the maximum intensity observed within the region (as well as date and location of occurrence):

[6]:
```
from climada_petals.hazard.emulator.stats import haz_max_events

# for this example, we regard grid cells as `affected` if they face at least 34 knots␣
↪wind speeds
KNOTS_2_MS = 0.514444
MIN_WIND_MS = 34 * KNOTS_2_MS

max_events_base = haz_max_events(hazard_EP, min_thresh=MIN_WIND_MS)
max_events_base
```

```
2021-03-24 17:46:52,405 - climada.hazard.emulator.stats - INFO - Condensing 45300␣
↪hazards to 7401 max events …
```

[6]:

	id	name	year	month	day	lat	lon	intensity
0	1	1	1950	8	17	25.1	-112.3	42.654243
1	5	5	1950	9	3	18.7	-104.0	36.720880
2	8	8	1950	9	17	27.2	-114.0	32.730017
3	13	13	1950	11	24	17.2	-100.6	42.861889
4	17	17	1950	10	26	22.9	-106.3	47.887202
…	…	…	…	…	…	…	…	…
7396	45260	28460	2100	11	21	24.1	-107.0	29.222734
7397	45261	28461	2100	10	25	27.4	-114.2	34.721709
7398	45274	28474	2100	10	20	18.1	-102.1	89.034839
7399	45284	28484	2100	10	9	17.8	-101.5	59.966329
7400	45298	28498	2100	10	13	24.2	-107.5	31.401268

(continues on next page)

(continued from previous page)

```
[7401 rows x 8 columns]
```

2.1.4 Making draws from the event pool

The most basic functionality of the emulator module is then to subsample from this event pool according to a desired frequency and intensity *without taking the date of events into account at all*:

[7]:
```
from climada_petals.hazard.emulator.emulator import EventPool
event_pool = EventPool(max_events_base)
draws = event_pool.draw_realizations(nrealizations=10, freq_poisson=10, intensity_
↪mean=30, intensity_std=5)
```

The result `draws` is a list of `nrealizations` subsamples from `tc_events_pool`. The number of events in each sample `DataFrame` is driven by a Poisson distribution (`lambda = 10`) and each DataFrame's mean intensity is 30±5:

[8]:
```
assert len(draws) == 10
display(draws[0])
print("Number of events in each sample:", [d.shape[0] for d in draws])
print("Mean intensity of each sample:", [d['intensity'].mean() for d in draws])
```

	id	name	year	month	day	lat	lon	intensity
754	5285	5285	1967	6	23	18.3	-103.3	30.302091
3458	22096	5296	2023	10	8	15.7	-93.7	47.669055
2357	15416	15416	2001	8	19	22.8	-110.1	30.646163
1329	8892	8892	1979	5	6	19.2	-104.8	39.077807
4153	26512	9712	2038	8	17	31.3	-116.4	18.318382
2131	13949	13949	1996	11	6	18.6	-103.4	34.297652
89	635	635	1952	6	14	21.3	-106.5	51.698398
5106	32250	15450	2057	9	8	21.4	-106.6	37.932184
4703	29808	13008	2049	11	1	59.8	-162.9	20.645396
4844	30510	13710	2051	9	22	26.2	-112.6	26.817070

```
Number of events in each sample: [10, 7, 11, 7, 10, 13, 11, 16, 12, 6]
Mean intensity of each sample: [33.74041981432531, 32.36753130917463, 34.
↪59815875000344, 31.5348526999235, 33.67298356212818, 27.126272794400833, 34.
↪588544482580225, 32.567385754309875, 33.97376593673842, 33.712914894777875]
```

2.1.5 Bias-corrected statistics

Now, as noted above, our database actually attaches date information to the events:

[9]:
```
import datetime as dt
minyear, maxyear = [dt.datetime.fromordinal(d) for d in [hazard.date.min(), hazard.
↪date.max()]]
print(f"The hazard event database covers the period between {minyear} and {maxyear}.")
```

```
The hazard event database covers the period between 1950-01-03 00:00:00 and 2100-12-
↪29 00:00:00.
```

However, the database contains much more events per year than the underlying physics would suggest. That's why the creator of the global database provides information about frequency, from which we created the following CSV-data (see last section for more information about the data source and pre-processing steps taken):

[10]:
```
import pandas as pd
frequency = pd.read_csv(EMULATOR_DATA_DIR.joinpath("freq_miroc_rcp26.csv"))
with pd.option_context("display.max_rows", 5):
    display(frequency)
```

```
     year      freq
0    1950  0.299428
1    1951  0.289091
..    …        …
149  2099  0.348597
150  2100  0.354156

[151 rows x 2 columns]
```

The `freq` column encodes the relative surplus of events for each year: A `freq` value of `0.29` means that the expected number of events for a particular year is `29`, while the actual number of events in the database for that year is `100`. Depending on your data provider, it might be more or less simple to derive this kind of information for your event database.

The database might be subject to regional biases: While the whole global event database might represent global statistics of certain physical properties very well, it can still systematically under- or overestimating regionally aggregated properties. One way to "bias-correct" this kind of effect is by comparing with historical records. That's where a second event database comes in, a database of observed hazard events:

[11]:
```
observed = TropCyclone.from_hdf5(EMULATOR_DATA_DIR.joinpath("hazard_360as_ibtracs_
↪1950-2019.hdf5"))
observed.centroids.region_id = shapely.vectorized.contains(
    region.shape, observed.centroids.lon, observed.centroids.lat) & observed.
↪centroids.on_land.astype(bool)
observed_EP = observed.select(reg_id=1)
```

```
2021-03-24 17:46:52,543 - climada.hazard.base - INFO - Reading /home/tovogt/.climada/
↪demo/data/emulator/hazard_360as_ibtracs_1950-2019.hdf5
```

Since the quality of observed data is known to vary a lot in different world regions, we restrict our dataset to an appropriate norm period. Again we condense the hazard object to a DataFrame.

[12]:
```
from climada_petals.hazard.emulator.const import TC_BASIN_NORM_PERIOD
norm_period = TC_BASIN_NORM_PERIOD[region.tc_basin[:2]]
observed_EP = observed_EP.select(date=(f"{norm_period[0]}-01-01", f"{norm_period[1]}-
↪12-31"))
max_events_observed = haz_max_events(observed_EP, min_thresh=MIN_WIND_MS)
```

```
2021-03-24 17:46:53,013 - climada.hazard.emulator.stats - INFO - Condensing 7335␣
↪hazards to 377 max events …
```

We would now have all the data for bias correction, but we don't have to do this manually. The emulator module takes care of this.

2.1.6 Initialize and calibrate the hazard emulator

The most complex part of the emulator module is the `HazardEmulator` class that automatically applies bias-correction and provides functionality to make draws according to corrected statistics. In the next section, we will see that it can even make draws according to a climate scenario.

[13]:
```
from climada_petals.hazard.emulator.emulator import HazardEmulator
em = HazardEmulator(max_events_base, max_events_observed, region, frequency,␣
↪pool=event_pool)
```

```
2021-03-24 17:46:53,188 - climada.hazard.emulator.random - INFO - Results of␣
↪intensity normalization by subsampling:
2021-03-24 17:46:53,189 - climada.hazard.emulator.random - INFO - - drop 35% of␣
↪entries satisfying 'intensity > 37.60902402823995'
2021-03-24 17:46:53,189 - climada.hazard.emulator.random - INFO - - mean intensity of␣
↪simulated events before dropping is 37.6090
2021-03-24 17:46:53,190 - climada.hazard.emulator.random - INFO - - mean intensity of␣
↪simulated events after dropping is 35.4003
2021-03-24 17:46:53,190 - climada.hazard.emulator.random - INFO - - mean intensity of␣
↪observed events is 34.6826
```

All corrections discussed above have already been applied upon initialization and aggregated in the `stats` attribute (it's mainly for internal use, so don't worry if you don't understand the meaning of all the columns):

[14]:
```
em.stats
```

[14]:

	year	eventcount	intensity_mean	intensity_std	intensity_max	\
0	1950	5.041561	37.560036	10.921193	56.148643	
1	1951	4.056249	32.092965	12.225921	61.743472	
2	1952	5.830183	37.488452	15.000097	60.163892	
3	1953	2.185263	33.317693	9.302551	41.871618	
4	1954	3.123722	30.061603	9.982461	37.396814	
..	…	…	…	…	…	
146	2096	5.641057	39.148745	12.819278	50.586874	
147	2097	7.649071	35.221123	12.889677	55.557459	
148	2098	4.050403	33.088812	12.990415	55.963069	
149	2099	5.217263	34.029102	14.014108	52.617260	
150	2100	7.288148	33.401478	17.953031	68.139491	

	eventcount_obs	intensity_mean_obs	intensity_std_obs	intensity_max_obs
0	3.0	32.444994	4.193251	38.302107
1	6.0	21.487378	2.879905	25.614798
2	NaN	NaN	NaN	NaN
3	3.0	26.853119	2.943347	30.412018
4	8.0	29.232035	8.669075	47.533993
..	…	…	…	…
146	NaN	NaN	NaN	NaN
147	NaN	NaN	NaN	NaN
148	NaN	NaN	NaN	NaN
149	NaN	NaN	NaN	NaN
150	NaN	NaN	NaN	NaN

[151 rows x 9 columns]

We can now draw ensembles of events that adhere to the bias-corrected statistics:

[15]:
```
# the meaning of `predict_statistics` will be explained in the next section
em.predict_statistics()
draws = em.draw_realizations(10, (2030, 2050))
```

```
2021-03-24 17:46:53,222 - climada.hazard.emulator.emulator - INFO - Predicting␣
↪statistics without climate index predictor…
2021-03-24 17:46:53,226 - climada.hazard.emulator.emulator - INFO - Drawing 10␣
↪realizations for period  (2030, 2050)
2030 … 2050 … 2050
```

The returned object `draws` is a `DataFrame` with each row corresponding to a storm event from the hazard pool `hazard_EP` (see above): The column `real_id` assigns one of 100 realizations to each of the events while the columns `id` and `name` are the unique ID and name used in `hazard_EP` to identify this hazard event. The column `year` indicates the year in which the event *would* occur under the hypothetical corrected statistics.

[16]:
```
display(draws[:25:2])
```

	id	name	year	real_id
0	5002	5002	2030	0
2	28485	11685	2030	0
4	8724	8724	2030	2
6	17148	348	2030	2
8	10339	10339	2030	3
10	7026	7026	2030	3
12	27333	10533	2030	5
14	1523	1523	2030	6
16	12183	12183	2030	7
18	20406	3606	2030	8
20	12394	12394	2030	9
22	37101	20301	2031	0
24	17358	558	2031	0

2.1.7 Draw samples according to climate scenario

The emulator can also be used to sample hypothetical events within an arbitrary time period covered by one or several climate index time series:

[17]:
```
climate_indices = [pd.read_csv(EMULATOR_DATA_DIR.joinpath("gmt_miroc_rcp26.csv")),
                   pd.read_csv(EMULATOR_DATA_DIR.joinpath("esoi_miroc_rcp26.csv"))]
with pd.option_context("display.max_rows", 5):
    display(*[df for df in climate_indices])
```

	gmt	year	month
0	0.040892	1861	7
1	0.053463	1862	7
..	…	…	…
437	1.397416	2298	7
438	1.375225	2299	7

[439 rows x 3 columns]

	esoi	year	month
0	-0.316829	1861	1
1	-0.479121	1861	2
…	…	…	…
5266	-0.569960	2299	11
5267	-0.523571	2299	12

[5268 rows x 3 columns]

An arbitrary number of climate indices can be provided in a list, each as a `DataFrame` with a `year` column and an

additional column (in this example, `gmt` or `esoi`, respectively) containing the actual climate index data. Optionally, each climate index `DataFrame` can be at monthly resolution (like in the `esoi` example), as indicated by an additional `month` column. A constant `month` column (like in the `gmt` example above) will be discarded automatically by the emulator.

Using those climate index time series, we calibrate the emulator, i.e., we determine a statistical connection between climate indices (GMT and ENSO in this example) and `tc_events_pool`:

[18]:
```
em.calibrate_statistics(climate_indices)
```

Now that the emulator is calibrated, we use GMT and ENSO time series to predict TC statistics under the chosen climate scenario:

[19]:
```
em.predict_statistics(climate_indices)
```

```
2021-03-24 17:46:54,572 - climada.hazard.emulator.emulator - INFO - Predicting␣
↪statistics with new climate index time series…
```

The predicted statistics are stored in the `stats_pred` attribute of the emulator:

[20]:
```
em.stats_pred
```

[20]:
```
     year       gmt      esoi  intensity_mean  intensity_mean_residuals  \
0    1861  0.040892  0.170012       34.803000                  2.637305
1    1862  0.053463 -0.813848       34.038959                  2.637305
2    1863  0.063511  1.040317       35.478856                  2.637305
3    1864  0.061416  1.441388       35.790317                  2.637305
4    1865  0.059797  1.129038       35.547754                  2.637305
..    …       …        …               …                        …
434  2295  1.395940 -0.726180       34.107040                  2.637305
435  2296  1.407416  0.648489       35.174572                  2.637305
436  2297  1.410368  0.769933       35.268883                  2.637305
437  2298  1.397416  0.870595       35.347054                  2.637305
438  2299  1.375225 -0.526354       34.262219                  2.637305

     eventcount  eventcount_residuals
0      4.176576              1.272837
1      4.406238              1.272837
2      4.018993              1.272837
3      3.935384              1.272837
4      3.996921              1.272837
..          …                    …
434    6.248324              1.272837
435    5.842586              1.272837
436    5.811002              1.272837
437    5.761524              1.272837
438    6.151251              1.272837

[439 rows x 7 columns]
```

Since the climate index time series covers a larger range than our hazard database (1861-2299 vs. 1950-2100), we can even predict the statistics in years that have not been covered by the hazard database.

[21]:
```
draws = em.draw_realizations(10, (2110, 2130))
display(draws[:25:2])
```

```
2021-03-24 17:46:54,603 - climada.hazard.emulator.emulator - INFO - Drawing 10␣
↪realizations for period  (2110, 2130)
2110 … 2130 … 2130
```

```
       id   name  year  real_id
0   13973  13973  2110        0
2   24253   7453  2110        0
4   14891  14891  2110        0
6   21909   5109  2110        0
8     309    309  2110        1
10  27897  11097  2110        1
12  38751  21951  2110        1
14  31890  15090  2110        1
16  13222  13222  2110        2
18  14662  14662  2110        2
20   9915   9915  2110        2
22  26466   9666  2110        3
24   1523   1523  2110        3
```

2.1.8 Create sample Hazard object from draws

Using the DataFrame `draws`, we can produce a hazard object that contains the sampled events from `hazard_EP`:

[22]:
```
hazard_sample = hazard_EP.select(event_names=draws['name'].tolist())
```

For this event to represent the sample, we need to adjust the date (and, optionally, the frequency according to the number of realizations):

[23]:
```
years = [dt.datetime.fromordinal(d).year for d in hazard_sample.date]
hazard_sample.date += [dt.datetime(y_dst, 1, 1).toordinal() - dt.datetime(y, 1, 1).
↪toordinal()
                       for y_dst, y in zip(draws['year'].values, years)]
```

[24]:
```
dates = [dt.datetime.fromordinal(d) for d in hazard_sample.date]
display(dates[:5])
display(dates[-5:])
```

```
[datetime.datetime(2110, 7, 26, 0, 0),
 datetime.datetime(2110, 12, 11, 0, 0),
 datetime.datetime(2110, 1, 14, 0, 0),
 datetime.datetime(2110, 7, 26, 0, 0),
 datetime.datetime(2110, 8, 29, 0, 0)]
```

```
[datetime.datetime(2130, 7, 29, 0, 0),
 datetime.datetime(2130, 5, 28, 0, 0),
 datetime.datetime(2130, 6, 2, 0, 0),
 datetime.datetime(2130, 9, 13, 0, 0),
 datetime.datetime(2130, 4, 3, 0, 0)]
```

For the resulting `TropCyclone` object to be valid, the event names have to be unique. Since our subsampler makes draws for each year and realization independent of any other year and realization, events might occur in more than one year or realization. One way of dealing with this, is the following renaming step:

[25]:
```
hazard_EP.event_names = [f"{row.name}-{row.year}-{row.real_id}"
                         for index, row in draws.iterrows()]
hazard_EP.check()
```

2.1.9 About the input data used for this notebook

Since a crucial ingredient for this module is a (sufficiently large) database of hazard events, executing this tutorial notebook requires external data that is not provided with the official CLIMADA repository. The data is expected to be located in the subdirectory `{CONFIG.local_data.demo}/emulator/` according to the CLIMADA config (called `EMULATOR_DATA_DIR` above). Since parts of the data are under a proprietary license, the data is only available upon request from Thomas Vogt (PIK). In the following, you find a list of all the files used in this notebook with information about sources and pre-processing steps taken:

- **``hazard_360as_miroc_rcp26.hdf5``**: A `TropCyclone` object (stored as 700 MB hdf5-file) that represents the simulated hazard event database. It has been generated from simulated TC tracks provided by Kerry Emanuel for ISIMIP (version 2b). However, since the track data comes with a proprietary license, they are not listed in the official ISIMIP repositories. Tracks for the period 1950-2100 according to the MIROC5 GCM simulations of the RCP 2.6 scenario have been loaded into CLIMADA using the `TCTracks.from_simulations_emanuel` constructor. The wind fields have been computed from the tracks using CLIMADA's `TropCyclone.from_tracks` function and a global set of centroids with 360 arc-seconds (0.1 degree) resolution onland and 1 degree resolution offshore (plotted above).
- **``hazard_360as_ibtracs_1950-2019.hdf5``**: A `TropCyclone` object (stored as 95 MB hdf5-file) that represents the observed hazard event database. It has been generated from records in the IBTrACS database for the years 1950-2019. The tracks have been loaded into CLIMADA using the `TCTracks.from_ibtracs_netcdf` constructor. Then, wind fields have been computed as for the simulated tracks (see `hazard_360as_miroc_rcp26.hdf5`).
- **``freq_miroc_rcp26.csv``**: A table with `year` and `freq` column covering the years 1950-2100. The `freq` values have been obtained by dividing the `freqyear` field contained in the simulated TC tracks (see `hazard_360as_miroc_rcp26.hdf5`) by 300 (the total number of global tracks per year in the simulated TC track files).
- **``gmt_miroc_rcp26.csv``**: A table with `gmt`, `year` and `month` field (the month is constantly set to `7`). GMT stands for "Global Mean (surface) Temperature". The CMIP5 monthly mean atmopsheric (Amon) `tas` temperature field of MIROC5 RCP 2.6 simulations covering the years 1861-2299 has been averaged globally and annually using the data processing tool CDO. The data is relative to the mean over the 500 year pre-industrial control run of the GCM and the time series has been smoothed by applying a 21-year running mean.
- **``esoi_miroc_rcp26.csv``**: A table with `esoi`, `year` and `month` field. ESOI stands for "Equatorial Southern Oscillation Index". Instructions on how to compute this index from air pressure can be found on the ENSO Monitoring website, hosted by Columbia University. For the air pressure input, we extracted the CMIP5 monthly mean atmopsheric (Amon) `psl` field of MIROC5 RCP 2.6 simulations covering the years 1861-2299.

2.2 Hazard: Landslides

The `landslide` class inherits from the `hazard` class. Other than some of the hazard modules available in `climada`, the landslide module does not run a physical model in its background. Rather, this tutorial is a suggestion of how to handle two different types of hazard source files (in one case, already the finished product of some model output, in the other case just a historic data collection).

We propose 2 different types of landslide hazard datasets that the module's methods work well with: * *historic landslides*: historic event sets based on the NASA COOLR global landslide catalogue, continuously updated. * *probabilistic landslides*: two raster files on probabilistic LS hazard, one for landslides triggered by precipitation and one for landslides triggered by earthquakes, based on data from the Norwegian Geotechnical Institute (NGI) for UNEP GRID, last improved 2018.

The module comes with two main functions, both delivering a raster-hazard set (once of historic occurrences, once with probabilistically sampled occurrences) * `from_hist()` * `from_prob()`

2.2.1 Option 1: historic landslide events: NASA COOLR initiative

Data from the global landslide catalogue is continuously updated as part of the Cooperative Open Online Landslide Repository (https://pmm.nasa.gov/landslides/coolrdata.html#download). The data consists in points representing an approximate occurence location, without spatial extent and any kind of "intensity" (binary events).

The most recent version of the dataset should always be downloaded by going to the link > "Open Landslide Viewer" (takes some time to load) > click "Download the full Landslide Catalog" > selecting the "NASA Global Landslide Catalog Points (Shapefile)" for download.

Download and unzip the up-to-date version.

Put it into the **Important**: The original file has a typo in one of its entries, which messes up the reading of a bounding box. Reading it once into memory with geopandas and re-saving it as shapefile solves this. This has to be done only once.

[7]:
```
# Amending the Landslide catalog by loading and re-saving (only necessary first time!)
import geopandas as gpd

# replace with your path to file nasa_global_landslide_catalog_point.shp if not in ~/
↪climada/data folder
PATH_COOLR = str(CONFIG.local_data.system.dir()) + '/haz/nasa_global_landslide_
↪catalog_point/nasa_global_landslide_catalog_point.shp'
ls_gdf_all = gpd.read_file(PATH_COOLR)
ls_gdf_all.to_file(PATH_COOLR)
```

Now we can start the actual task..

[8]:
```
# Loading packages and constants
%matplotlib inline
from climada_petals.hazard.landslide import Landslide
from climada import CONFIG
# replace with your path to file nasa_global_landslide_catalog_point.shp if not in ~/
↪climada/data folder
PATH_COOLR = str(CONFIG.local_data.system.dir()) + '/haz/nasa_global_landslide_
↪catalog_point/nasa_global_landslide_catalog_point.shp'
```

The historic landslide events are read into a landslide hazard set. Since the events are reported as simple points, we convert the hazard set to a raster file with a certain resolution.

Important note on projections and resolution The resolution is up to your choice and has implications: The grid which is generated has the same projection and units as the input geodataframe with point landslide occurrences. By default, this is EPSG:4326, which is a non-projected, geographic CRS. This means, depending on where on the globe the analysis is performed, the area per gridcell differs vastly. Consider this when setting your resolution (e.g. at the equator, 1° ~ 111 km). In turn, one can use a projected CRS which preserve angles and areas within the reference area for which they are defined. To do this, reproject the input_gdf to the desired projection. For more on projected & geographic CRS, read here

Here, we will stick with the default geographic (non-projected) EPSG 4326 and take a resolution of 0.004° (which is about 450m at the equator). All area within a grid cell that "hosts" an event point is hence affected.

[9]:
```
bbox_taiwan = (120.0, 21.5, 122.0, 25.5) # bbox as (minx, miny, maxx, maxy)
# Example for Taiwan
haz_ls_Taiwan_hist = Landslide.from_hist(bbox=bbox_taiwan, input_gdf=PATH_COOLR,␣
↪res=0.004)
# Visual inspection of the hazard
haz_ls_Taiwan_hist.plot_intensity(0);
```

```
2022-05-09 11:00:42,954 - climada_petals.hazard.landslide - INFO - Reading in gdf␣
↪from source /Users/evelynm/climada/data/nasa_global_landslide_catalog_point/nasa_
```

(continues on next page)

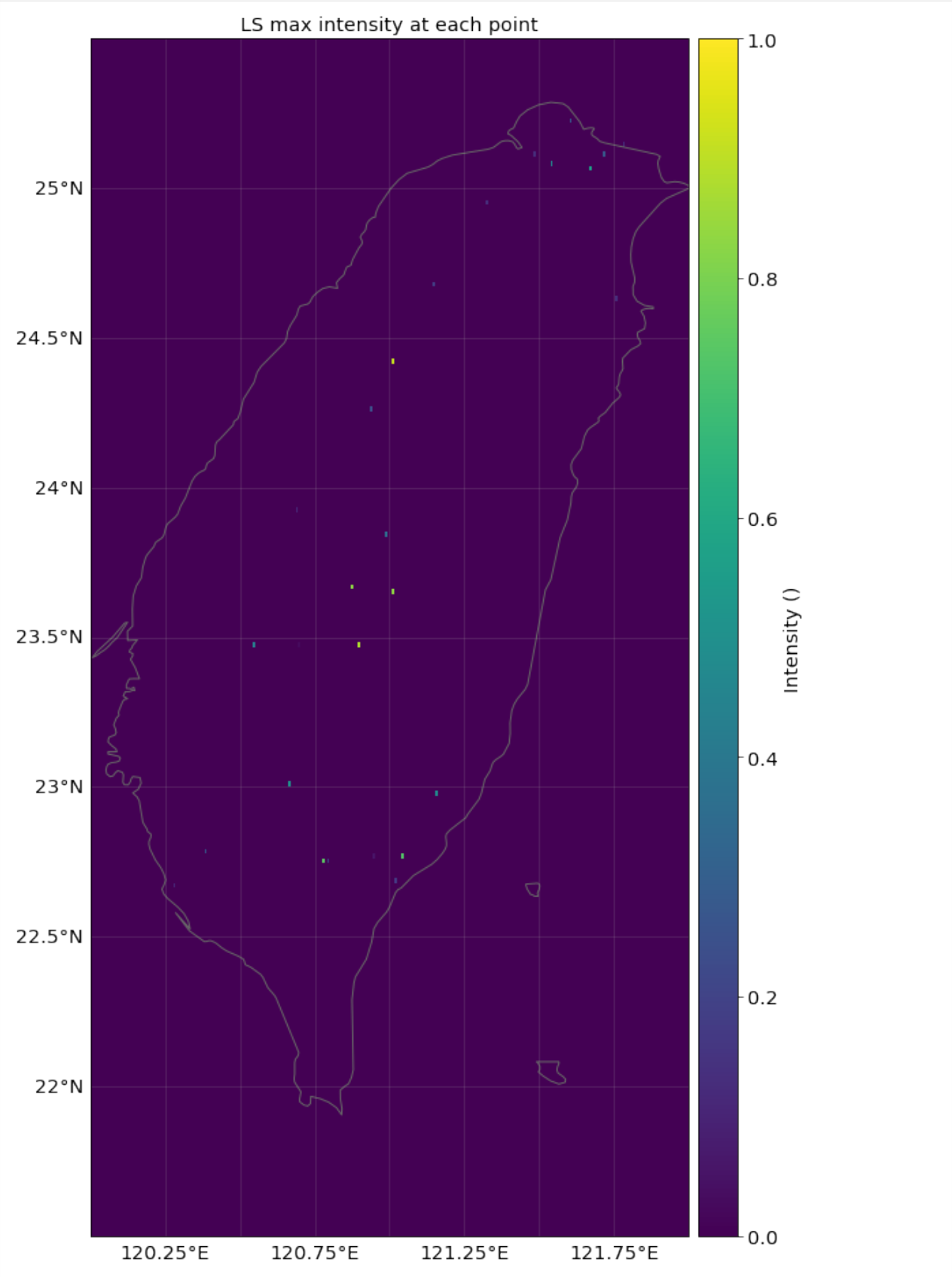
(continued from previous page)

```
↪global_landslide_catalog_point.shp
2022-05-09 11:00:43,058 - climada_petals.hazard.landslide - INFO - Generating a␣
↪raster with resolution 0.004 for box (120.0, 21.5, 122.0, 25.5)
```

```
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/
↪crs.py:1256: UserWarning: You will likely lose important projection information␣
↪when converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  return self._crs.to_proj4(version=version)
```

[9]:
```
<GeoAxesSubplot:title={'center':'LS max intensity at each point'}>
```

```
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated␣
↪and will be removed in Shapely 2.0. Check the length of the `geoms` property␣
↪instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access␣
↪the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated␣
↪and will be removed in Shapely 2.0. Check the length of the `geoms` property␣
↪instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```


LS max intensity at each point
25°N
24.5°N
24°N
23.5°N
23°N
22.5°N
22°N
120.25°E
120.75°E
121.25°E
121.75°E
1.0
0.8
0.6
0.4
0.2
0.0
Intensity ()

2.2.2 Exemplary end-to-end impact calculation using the historic LS option

The steps below follow the normal routine of defining impact functions, getting an exposure, and performing an impact calculation based on the given historic hazard set.

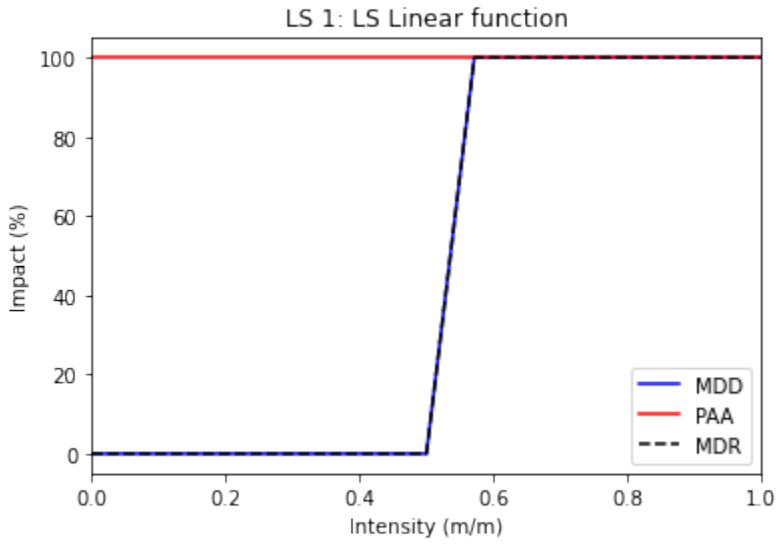
Impact functions relate the hazard intensity to a percentage of damage in the exposure. For a detailed description on impact functions, check out the respective tutorial.

Since the historic landslides are binary (occurrence / non-occurrence), their intensity is simply put to "1" at the respective grid-point where one occurred. A dummy step impact function is created for illustrative purposes, where damage (impact) is simply 100% when intensity is (close to) 1, and 0 else.

[13]:
```
from climada.entity.exposures import LitPop
from climada.entity.entity_def import Entity
from climada.entity import ImpactFuncSet, ImpactFunc
from climada.engine import Impact
import numpy as np
```

[14]:
```
# Set impact function (see tutorial climada_entity_ImpactFuncSet)
impf_LS_hist = ImpactFunc()
impf_LS_hist.haz_type = 'LS'
impf_LS_hist.id = 1
impf_LS_hist.name = 'LS Linear function'
impf_LS_hist.intensity_unit = 'm/m'
impf_LS_hist.intensity = np.linspace(0, 1, num=15)
impf_LS_hist.mdd = np.sort(np.array([0,0,0,0,0,0,0,0,1., 1., 1., 1., 1., 1., 1.]))
impf_LS_hist.paa = np.sort(np.linspace(1, 1, num=15))
impf_LS_hist.check()
impf_LS_hist.plot()
ifset_LS_hist = ImpactFuncSet()
ifset_LS_hist.append(impf_LS_hist)
```

```
2022-04-11 15:11:37,339 - climada.entity.impact_funcs.base - WARNING - For intensity␣
↪= 0, mdd != 0 or paa != 0. Consider shifting the origin of the intensity scale. In␣
↪impact.calc the impact is always null at intensity = 0.
```



For a detailed description of the *Exposure* class, refer to the respective tutorial. This LS tutorial uses the LitPop class, which models countries' gridded asset exposure by disaggregating a macroeconomic indicator (e.g. total asset value or GDP) proportional to the product of night light intensities ("Lit") and gridded population count ("Pop") per country.

[15]:
```
# Set LitPop exposure for Taiwan
exp_LS_hist = LitPop.from_countries(['Taiwan'])
exp_LS_hist.set_geometry_points()
exp_LS_hist.gdf.rename({'impf_': 'impf_LS'}, axis='columns',inplace=True)
exp_LS_hist.set_lat_lon()
exp_LS_hist.check()

# plot the exposure
exp_LS_hist.plot_basemap();
```

```
2022-04-11 15:11:43,471 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: TWN (158)…

2022-04-11 15:11:43,475 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2022-04-11 15:11:43,527 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

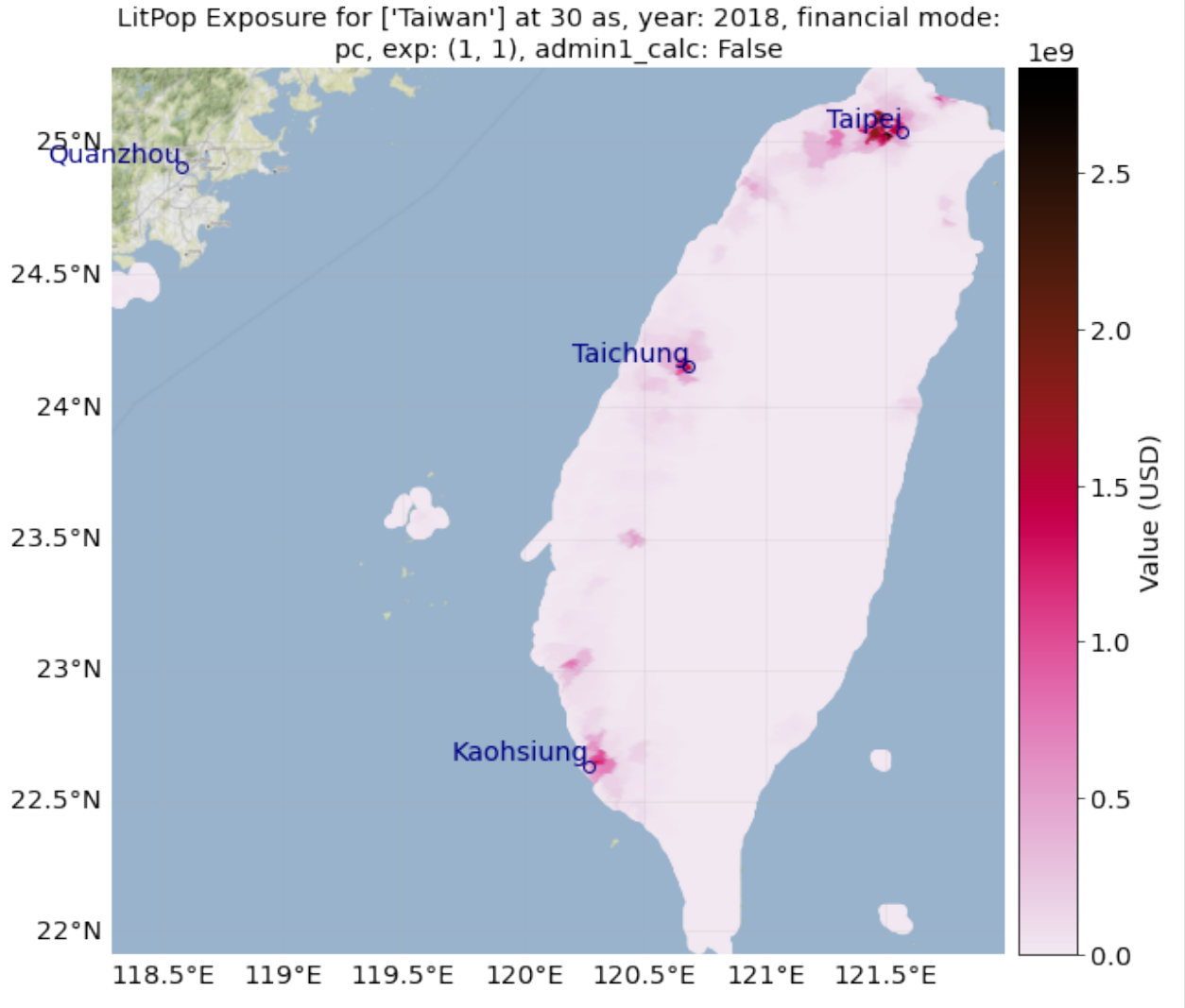
```
/Users/evelynm/climada_python/climada/entity/exposures/litpop/litpop.py:607:␣
↪ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated and will␣
↪be removed in Shapely 2.0. Check the length of the `geoms` property instead to get␣
↪the  number of parts of a multi-part geometry.
  for idx, polygon in enumerate(list(country_geometry)):
/Users/evelynm/climada_python/climada/entity/exposures/litpop/litpop.py:607:␣
↪ShapelyDeprecationWarning: Iteration over multi-part geometries is deprecated and␣
↪will be removed in Shapely 2.0. Use the `geoms` property to access the constituent␣
↪parts of a multi-part geometry.
  for idx, polygon in enumerate(list(country_geometry)):
```

```
2022-04-11 15:11:44,320 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-04-11 15:11:44,321 - climada.util.finance - WARNING - GDP data for TWN is not␣
↪provided by World Bank.                        Instead, IMF data is returned here.
2022-04-11 15:11:44,350 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-04-11 15:11:44,351 - climada.entity.exposures.base - INFO - category_id not set.
2022-04-11 15:11:44,351 - climada.entity.exposures.base - INFO - cover not set.
2022-04-11 15:11:44,352 - climada.entity.exposures.base - INFO - deductible not set.
2022-04-11 15:11:44,352 - climada.entity.exposures.base - INFO - centr_ not set.
2022-04-11 15:11:44,354 - climada.util.coordinates - INFO - Setting geometry points.
2022-04-11 15:11:44,378 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-04-11 15:11:44,389 - climada.entity.exposures.base - INFO - category_id not set.
2022-04-11 15:11:44,390 - climada.entity.exposures.base - INFO - cover not set.
2022-04-11 15:11:44,390 - climada.entity.exposures.base - INFO - deductible not set.
2022-04-11 15:11:44,391 - climada.entity.exposures.base - INFO - centr_ not set.
2022-04-11 15:11:44,434 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/
↪crs.py:1256: UserWarning: You will likely lose important projection information␣
↪when converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  return self._crs.to_proj4(version=version)
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/contextily/
↪tile.py:265: FutureWarning: The url format using 'tileX', 'tileY', 'tileZ' as␣
↪placeholders is deprecated. Please use '{x}', '{y}', '{z}' instead.
  warnings.warn(
```

```
2022-04-11 15:11:59,465 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

[15]:
```
<GeoAxesSubplot:title={'center':"LitPop Exposure for ['Taiwan'] at 30 as, year: 2018,␣
↪financial mode:\npc, exp: (1, 1), admin1_calc: False"}>
```



[16]:
```
# Set Entity
ent_LS_hist = Entity()
ent_LS_hist.exposures = exp_LS_hist
ent_LS_hist.impact_funcs = ifset_LS_hist
```

Important note: Climada allows you to perform damage statistics (such as average annual impact, impact exceedance curves, etc.). Since those reported events have no guarantee of completeness, and cover a relatively short time, it is strongly advised **not** to perform such calculations. Since fr For `impact.at_event` in turn, no frequency correction is made. Apply the probabilistic method (explained below) for such calculations.

[17]:
```
# Impact calculation from historic landslides, with exposure and impact function␣
↪defined as above.
imp_LS_Taiwan_hist = Impact()
imp_LS_Taiwan_hist.calc(ent_LS_hist.exposures, ent_LS_hist.impact_funcs, haz_ls_
```

(continues on next page)

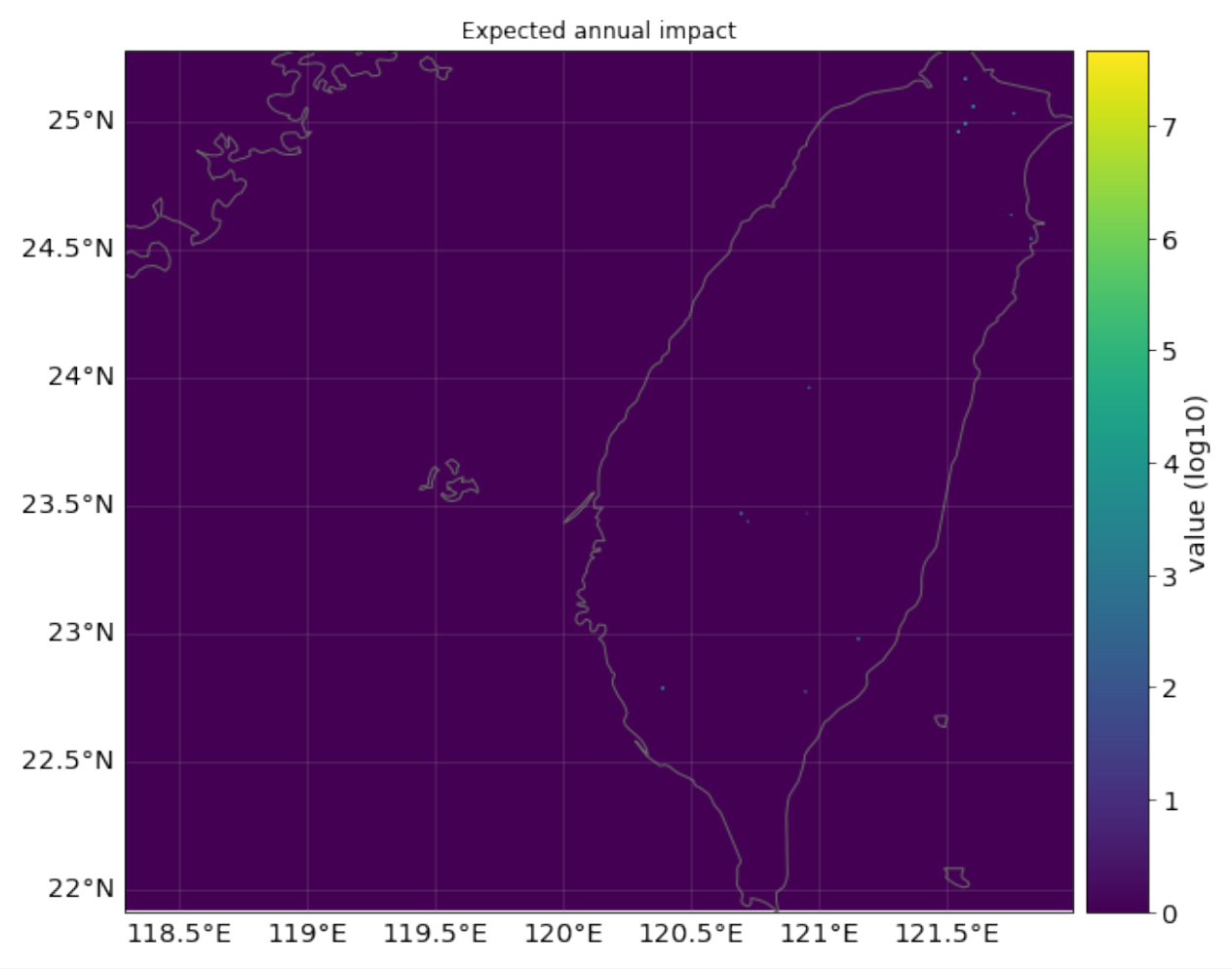
(continued from previous page)

```
↪Taiwan_hist)
imp_LS_Taiwan_hist.plot_raster_eai_exposure()
print(f'The overall estimated impact from all events is {int(imp_LS_Taiwan_hist.aai_
↪agg)} $');
```

```
2022-04-11 15:12:01,829 - climada.entity.exposures.base - INFO - Matching 46167␣
↪exposures with 501501 centroids.
2022-04-11 15:12:01,835 - climada.engine.impact - INFO - Calculating damage for 45512␣
↪assets (>0) and 73 events.
2022-04-11 15:12:01,853 - climada.util.coordinates - INFO - Raster from resolution 0.
↪008333329999956618 to 0.008333329999956618.
```

```
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/
↪crs.py:1256: UserWarning: You will likely lose important projection information␣
↪when converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  return self._crs.to_proj4(version=version)
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated␣
↪and will be removed in Shapely 2.0. Check the length of the `geoms` property␣
↪instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access␣
↪the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated␣
↪and will be removed in Shapely 2.0. Check the length of the `geoms` property␣
↪instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```

```
The overall estimated impact from all events is 115515445 $
```



2.2.3 Option 2: probabilistic landslide hazard (precipitation / earthquake-induced) from UNEP / NGI

The global probabilistic hazardsets are provided publicly by UNEP GRID and were developed by the Norwegian Geotechnical Institute (NGI).

Since the webservices are currently (as of 08/20) not working, please download the geoTIFFs manually:

- Go to https://preview.grid.unep.ch/index.php?preview=data&events=landslides&evcat=2&lang=eng for precipitation-triggered landslides.
- Go to https://preview.grid.unep.ch/index.php?preview=data&events=landslides&evcat=1&lang=eng for earthquake-triggered landslides.
- Unzip the folder and move it to a sensible location

The datasets are in units of expected annual probability and percentage of pixel of occurrence of a potentially destructive landslide event x 1000000 and include an estimate of the annual frequency of landslide triggered by precipitation / earthquakes. It depends on the combination of trigger and susceptibility defined by six parameters: slope factor, lithological (or geological) conditions, soil moisture condition, vegetation cover, precipitation and seismic conditions.

Discrete events are produced by sampling over the occurrence probabilities in each grid cell. The user has the option to choose either a binomial or a Poisson distribution via the `dist` kwarg. Since occurrence probabilities are given annually,

an event represents a year of sampled landslide occurrences in the area.

Read the documentation for details.

[14]:
```
# Loading packages and setting constants
%matplotlib inline
from climada_petals.hazard.landslide import Landslide
# replace with your path to file ls_pr.tif if not in ~/climada/data folder
PATH_LSPROB = str(CONFIG.local_data.system.dir()) + '/haz/ls_pr/ls_pr.tif'
```

Let's produce a 500 year probabilistic event set for the same geographic extent as above. Be aware that the resolution of the grid cells is 0.00833° (about 900m at the equator), so events tend to be quite large, if they occur.

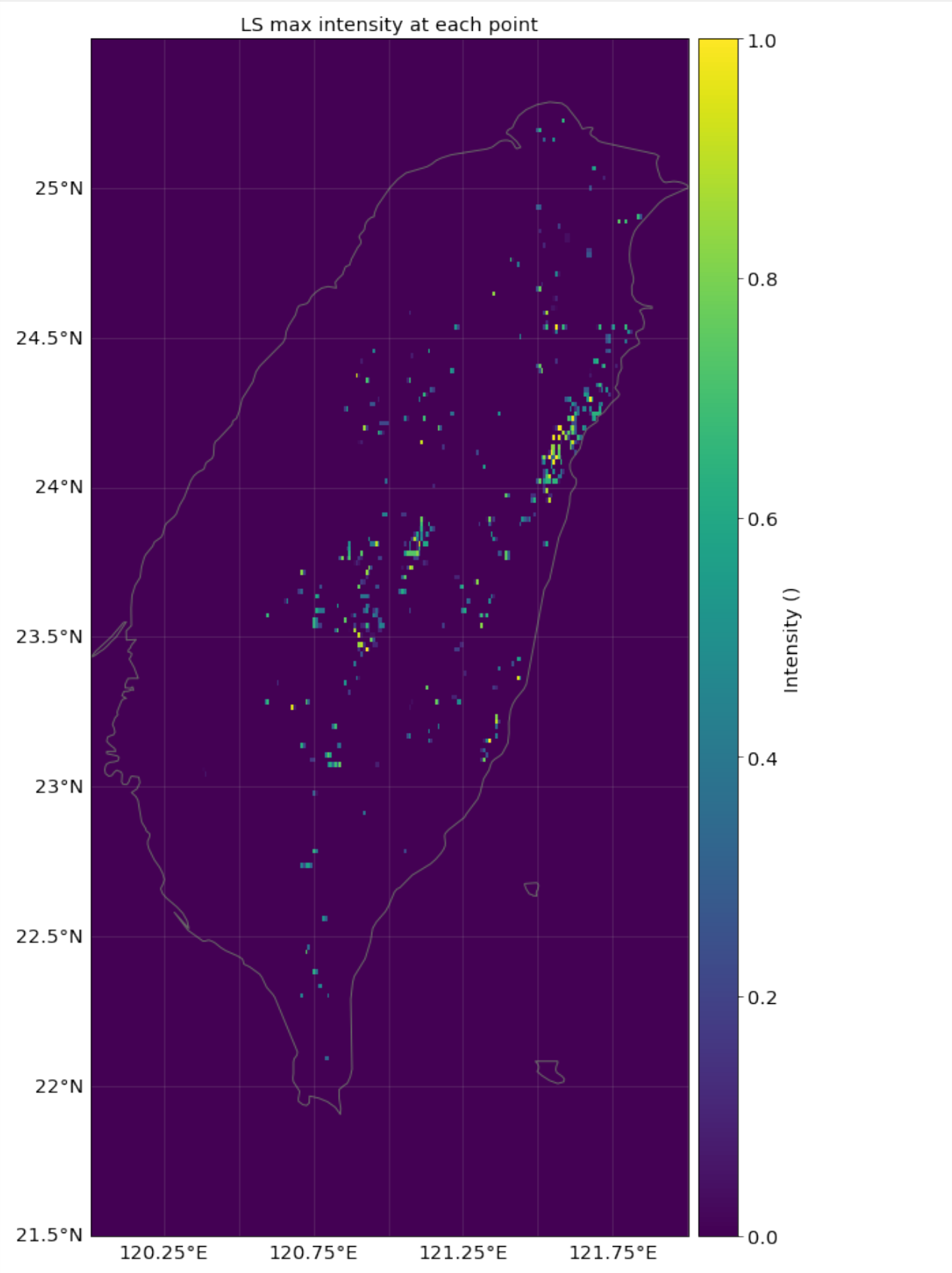
[15]:
```
# Setting precipitation-triggered landslide hazard for Taiwan
bbox_taiwan = (120.0, 21.5, 122.0, 25.5)

# The check-plots produce intensity (image 1) and fraction (image 2) plots
# a correction factor of 1 million is applied since probs are reportet as x10e6 in␣
↪the original data:
haz_ls_taiwan_prob =  Landslide.from_prob(bbox=bbox_taiwan, path_sourcefile=PATH_
↪LSPROB,
                                          n_years=500, corr_fact=10e6, dist='poisson
↪')
# Visual inspection of the hazard
haz_ls_taiwan_prob.plot_intensity(0);
```

```
2022-05-09 11:03:36,319 - climada.util.coordinates - INFO - Reading /Users/evelynm/
↪climada/data/haz/ls_pr/ls_pr.tif
2022-05-09 11:03:38,259 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
```

```
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/
↪crs.py:1256: UserWarning: You will likely lose important projection information␣
↪when converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  return self._crs.to_proj4(version=version)
```

[15]:
```
<GeoAxesSubplot:title={'center':'LS max intensity at each point'}>
```

```
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated␣
↪and will be removed in Shapely 2.0. Check the length of the `geoms` property␣
↪instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access␣
↪the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated␣
↪and will be removed in Shapely 2.0. Check the length of the `geoms` property␣
↪instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```


LS max intensity at each point
25°N
24.5°N
24°N
23.5°N
23°N
22.5°N
22°N
21.5°N
120.25°E
120.75°E
121.25°E
121.75°E
1.0
0.8
0.6
0.4
0.2
0.0
Intensity ()

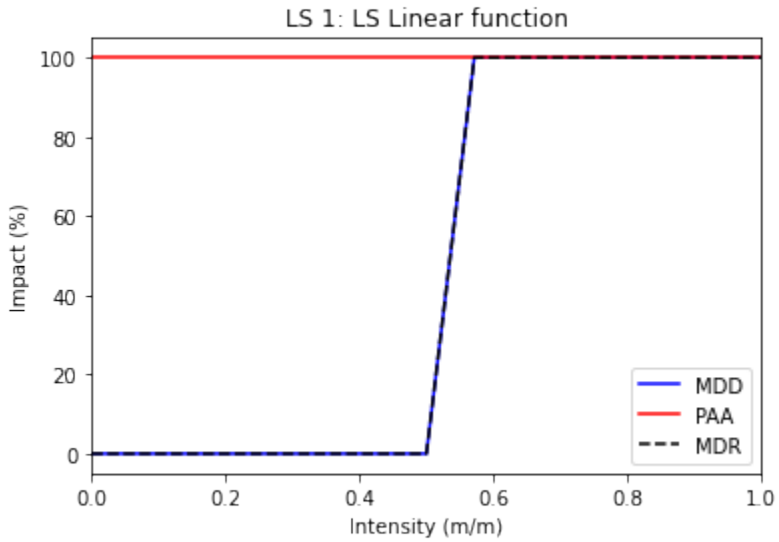
With the hazard set loaded, it is now possible to calculate the expected damage for the simulated period:

[16]:
```
from climada.entity import LitPop
from climada.entity.entity_def import Entity
from climada.entity import ImpactFuncSet, ImpactFunc
from climada.engine import Impact
import numpy as np
```

Important for impact calculations: Since impact functions act on the intensity of a hazard, an our hazard takes on binary intensity values (0 = no LS prob, 1 = >0 LS prob), it makes sense to define some step-function around those two values. The impact calculation accounts for the fractions (% of affected pixel and actual annual probability) by multiplying them into the end-result, anyways, under the hood.

[17]:
```
# Set impact function
impf_LS_prob = ImpactFunc()
impf_LS_prob.haz_type = 'LS'
impf_LS_prob.id = 1
impf_LS_prob.name = 'LS Linear function'
impf_LS_prob.intensity_unit = 'm/m'
impf_LS_prob.intensity = np.linspace(0, 1, num=15)
impf_LS_prob.mdd = np.sort(np.array([0,0,0,0,0,0,0,0,1., 1., 1., 1., 1., 1., 1.]))
impf_LS_prob.paa = np.sort(np.linspace(1, 1, num=15))
impf_LS_prob.check()
impf_LS_prob.plot()
impf_set_LS_prob = ImpactFuncSet()
impf_set_LS_prob.append(impf_LS_prob)
```

```
2022-05-09 11:04:17,672 - climada.entity.impact_funcs.base - WARNING - For intensity␣
↪= 0, mdd != 0 or paa != 0. Consider shifting the origin of the intensity scale. In␣
↪impact.calc the impact is always null at intensity = 0.
```



[22]:
```
# Set exposure for Taiwan:
exp_LS_prob = LitPop.from_countries(['Taiwan'])
exp_LS_prob.set_geometry_points()
exp_LS_prob.gdf.rename({'impf_': 'impf_LS'}, axis='columns', inplace=True)
exp_LS_prob.set_lat_lon()
exp_LS_prob.check()
```

```
2022-05-09 11:04:47,143 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: TWN (158)…

2022-05-09 11:04:47,144 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2022-05-09 11:04:47,146 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

```
/Users/evelynm/climada_python/climada/entity/exposures/litpop/litpop.py:607:␣
↪ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated and will␣
↪be removed in Shapely 2.0. Check the length of the `geoms` property instead to get␣
↪the  number of parts of a multi-part geometry.
  for idx, polygon in enumerate(list(country_geometry)):
/Users/evelynm/climada_python/climada/entity/exposures/litpop/litpop.py:607:␣
↪ShapelyDeprecationWarning: Iteration over multi-part geometries is deprecated and␣
↪will be removed in Shapely 2.0. Use the `geoms` property to access the constituent␣
↪parts of a multi-part geometry.
  for idx, polygon in enumerate(list(country_geometry)):
```

```
2022-05-09 11:04:47,847 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-05-09 11:04:47,848 - climada.util.finance - WARNING - GDP data for TWN is not␣
↪provided by World Bank.                        Instead, IMF data is returned here.
2022-05-09 11:04:47,878 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-05-09 11:04:47,879 - climada.entity.exposures.base - INFO - category_id not set.
2022-05-09 11:04:47,879 - climada.entity.exposures.base - INFO - cover not set.
2022-05-09 11:04:47,880 - climada.entity.exposures.base - INFO - deductible not set.
2022-05-09 11:04:47,881 - climada.entity.exposures.base - INFO - centr_ not set.
2022-05-09 11:04:47,883 - climada.util.coordinates - INFO - Setting geometry points.
2022-05-09 11:04:47,906 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-05-09 11:04:47,917 - climada.entity.exposures.base - INFO - category_id not set.
2022-05-09 11:04:47,918 - climada.entity.exposures.base - INFO - cover not set.
2022-05-09 11:04:47,918 - climada.entity.exposures.base - INFO - deductible not set.
2022-05-09 11:04:47,919 - climada.entity.exposures.base - INFO - centr_ not set.
```

[19]:
```
# Set Entity
ent_LS_prob = Entity()
ent_LS_prob.exposures = exp_LS_prob
ent_LS_prob.impact_funcs = impf_set_LS_prob
```

[23]:
```
# Set impact for probabilistic simulation
imp_LS_Taiwan_prob = Impact()
imp_LS_Taiwan_prob.calc(ent_LS_prob.exposures, ent_LS_prob.impact_funcs, haz_ls_
↪taiwan_prob)
imp_LS_Taiwan_prob.plot_raster_eai_exposure();
```

```
2022-05-09 11:05:29,631 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_LS
2022-05-09 11:05:29,635 - climada.engine.impact - INFO - Calculating damage for 45512␣
↪assets (>0) and 500 events.
2022-05-09 11:05:29,652 - climada.util.coordinates - INFO - Raster from resolution 0.
↪008333329999956618 to 0.008333329999956618.
```

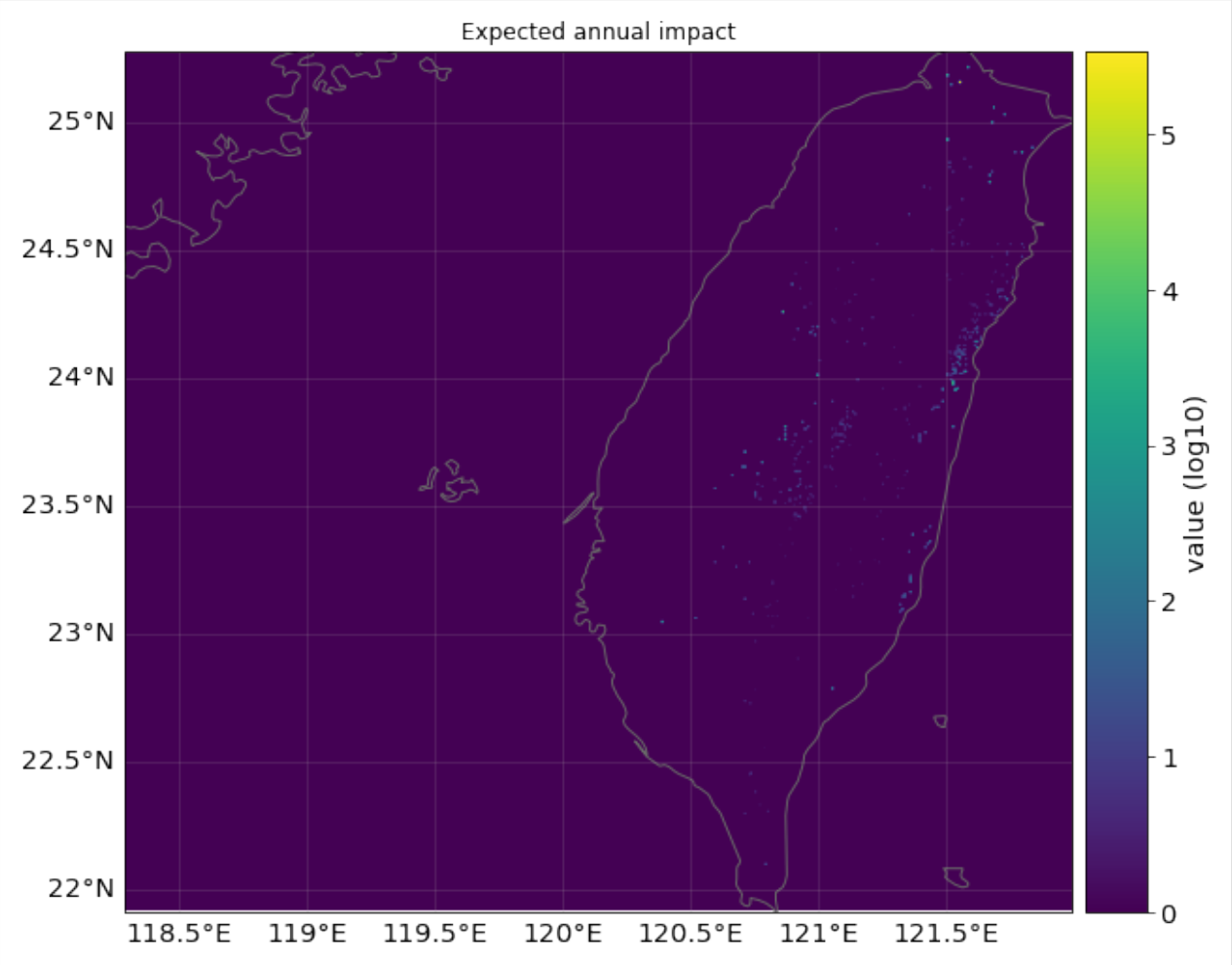
```
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/
↪crs.py:1256: UserWarning: You will likely lose important projection information␣
↪when converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
```

(continues on next page)

(continued from previous page)

```
  return self._crs.to_proj4(version=version)
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated␣
↪and will be removed in Shapely 2.0. Check the length of the `geoms` property␣
↪instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access␣
↪the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/cartopy/crs.
↪py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated␣
↪and will be removed in Shapely 2.0. Check the length of the `geoms` property␣
↪instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```

[23]:
```
<GeoAxesSubplot:title={'center':'Expected annual impact'}>
```



[24]:
```
imp_LS_Taiwan_prob.aai_agg
```

[24]:
```
1508057.2027470088
```

2.3 Hazard: RiverFlood

A river flood hazard is generated by the class RiverFlood() that extracts flood data simulated within the Inter-Sectoral Impact Model Intercomparison Project (ISIMIP, https://data.isimip.org/). The method `from_nc()` generates a data set with flood depth in m and the flooded fraction in each centroid. The data is derived from global hydrological models driven by various climate forcings. A link to the ISIMIP data repository will be provided soon. In this tutorial we show how flood depth and fractions can be translated into socio-economic impacts.

Besides, all other general Hazard Attributes, the class RiverFlood() has further Attributes related to the flooded area and flood volume:

2.3.1 additional Attributes (always calculated)

Name	Data Type	Description
fla_ann_av	float	average flooded area per year
fla_ev_av	float	average flooded area per event
fla_event	1d array(n_events)	average flooded area per year
fla_annual	1d array(n_years)	average flooded area per event

2.3.2 additional Attributes (only calculated if 'save_centr' = True in 'set_flooded_area()')

Name	Data Type	Description
fla_ev_centr	2d array(n_events x n_centroids)	flooded area in every centroid for every event
fla_ann_centr	2d array(n_years x n_centroids)	flooded area in every centroid for every year
fv_ann_centr	2d array(n_years x n_centroids)	flood volume in every centroid for every year

2.3.3 How is this tutorial structured?

Part 1: Data availability and use

Part 2: Generating a RiverFlood Hazard

Part 3: Calculating Flooded Area

Part 4: Generating ISIMIP Exposure

Part 5: Setting JRC damage functions

Part 6: Deriving flood impact with LitPop exposure

Part 1: Data availability and use

To work with the CLIMADA RiverFlood-Module input data containing spatially explicit flood depth and flooded fraction is required.

The input data can be found at https://files.isimip.org/cama-flood/ *.

On this page, data from the ISIMIP2a and ISIMIP2b simulation rounds can be accessed. The simulations contain the output of the river routing model CaMa-Flood for runoff input data generated by various combinations of global hydrological models (GHMs) and climate forcings.

ISIMIP2a

In the ISIMIP2a simulation round, 12 GHMs were driven by 4 climate reanalysis data sets and covers the time period 1971-2010. The runoff was used as input for CaMa-Flood to derive spatially explicit flood depth (flddph) and flooded fraction (fldfrc) of the maximum flood event of each year on 150 arcsec (~ 5 km) and 300 arcsec (~ 10 km) resolution. Data are provided for different protection standards including '0'- no protection, '100'- protection against all events smaller than 100 year return period, and'Flopros'- merged layer in the Flopros data base on global protection standards. File naming conventions follow the scheme:

<indicator_resolution_GHM_ClimateForcingDataset_ProtectionStandard.nc>

ISIMIP2b

In the ISIMIP2b simulation round, 6 GHMs were driven by 4 global circulation models (GCMs) and covers the time period 2005-2100 for RCP 2.6, 6.0 and RCP8.5 (only a smaller ensemble). Additionally, historical and preindustrial control runs are provided. Resolution and protection assumptions are the same as under ISIMIP2a. File naming conventions follow the scheme:

<indicator_resolution_GHM_GCM_ProtectionStandard.nc>

For futher information on flood data generation see also:

Willner, S. N. et al. (2018) 'Adaptation required to preserve future high-end river flood risk at present levels', Science Advances. American Association for the Advancement of Science, 4(1), p. eaao1914. doi:10.1126/sciadv.aao1914.

Willner, S. N., Otto, C. and Levermann, A. (2018) 'Global economic response to river floods', Nature Climate Change. Nature Publishing Group, 8(7), pp. 594–598. doi:10.1038/s41558-018-0173-2.

Sauer, I. et al. (2020) 'Climate Signals in River Flood Damages Emerge under Sound Regional Disaggregation'. doi:10.21203/rs.3.rs-37259/v1.

*Currently, log-in data are required, please contact inga.sauer@pik-potsdam.de to obtain access.

Part 2: Generating a RiverFlood Hazard

A river flood is generated with the method from_nc(). There are different options for choosing centroids. You can set centroids for: - countries - regions - global hazards - with random coordinates - with random shape files (under development)

Countries or regions can either be set with corresponding ISIMIPNatID centroids (ISINatIDGrid = True) or with Natural Earth Multipolygons (default). It is obligatory to set paths for flood depth and flood fraction, here we present example files from floods for the year 2000.

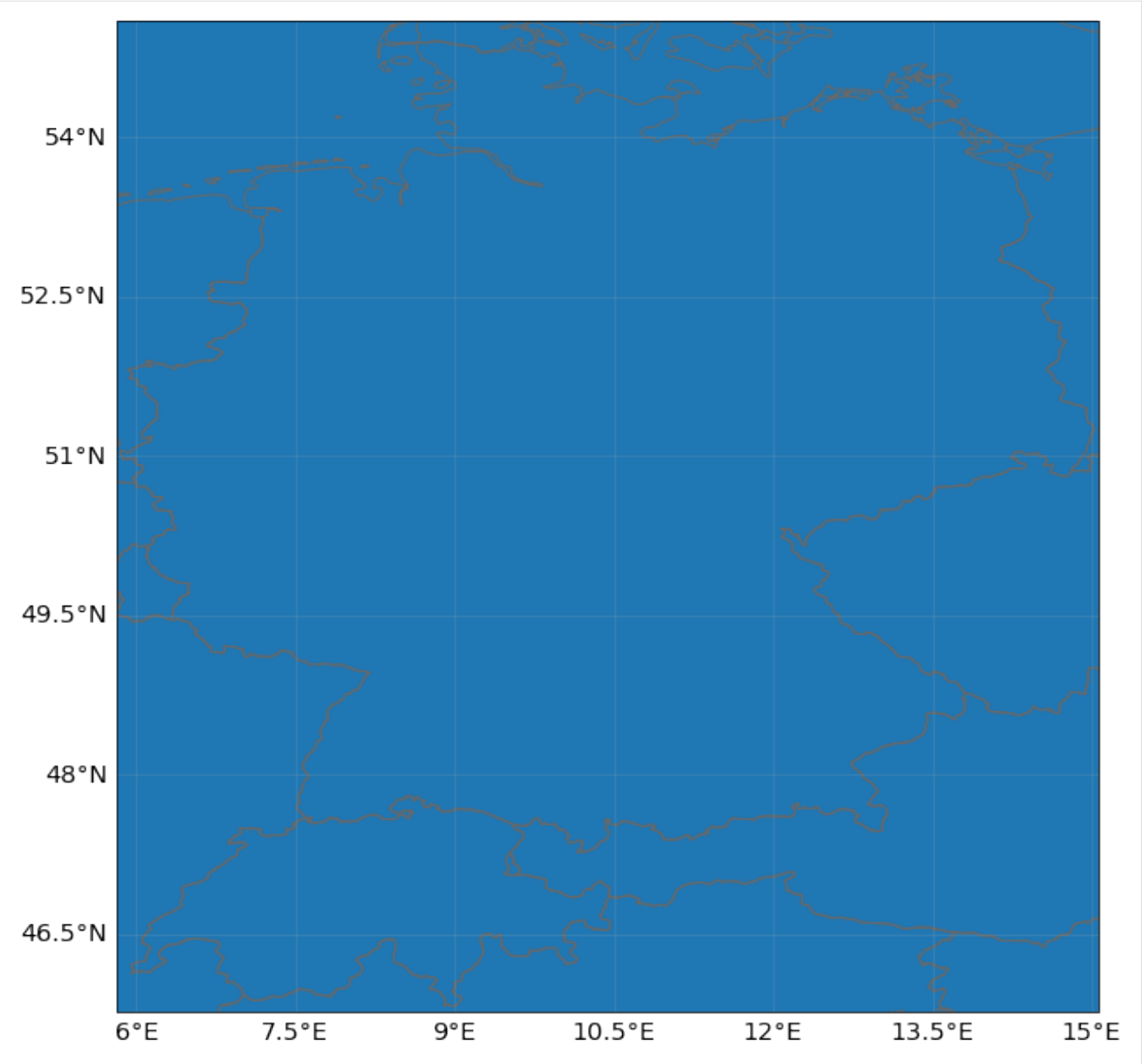
Setting floods for countries with Natural Earth Multipolygons:

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from climada_petals.hazard.river_flood import RiverFlood
from climada.hazard.centroids import Centroids
from climada_petals.util.constants import HAZ_DEMO_FLDDPH, HAZ_DEMO_FLDFRC

years = [2000]
# generating RiverFlood hazard from netCDF file
# uses centroids from Natural Earth Multipolygon for Germany and Switzerland
rf = RiverFlood.from_nc(countries = ['DEU','CHE'], years=years, dph_path=HAZ_DEMO_
↪FLDDPH, frc_path=HAZ_DEMO_FLDFRC)
rf.event_name
```

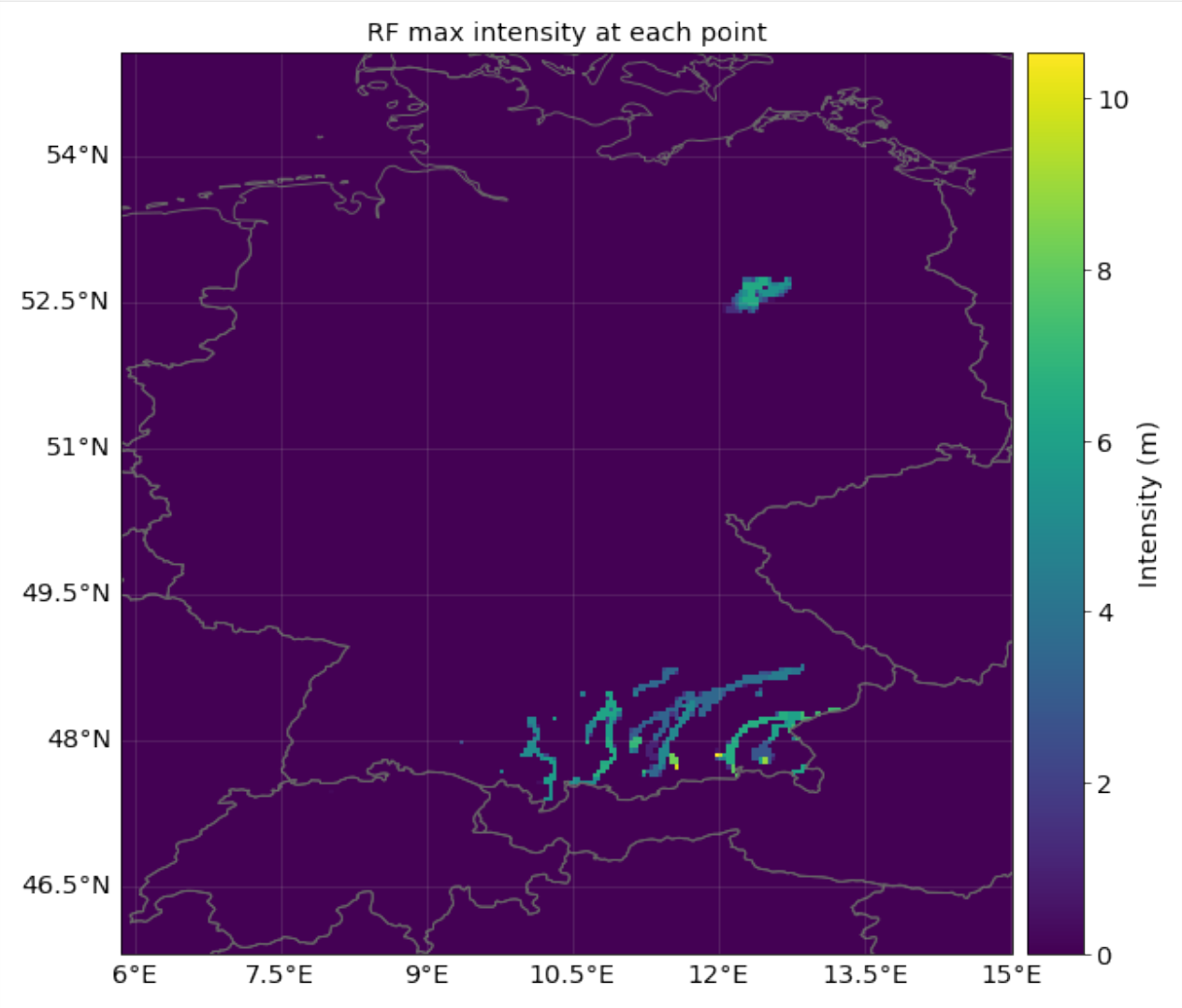
[1]:
```
['2000']
```

[2]:
```
# Note: Points outside the selected countries are masked in further analysis.
# plot centroids:
rf.centroids.plot()
# get resolution
print('resolution:', rf.centroids.meta['transform'][0])
```

```
resolution: 0.04166666666666662
```

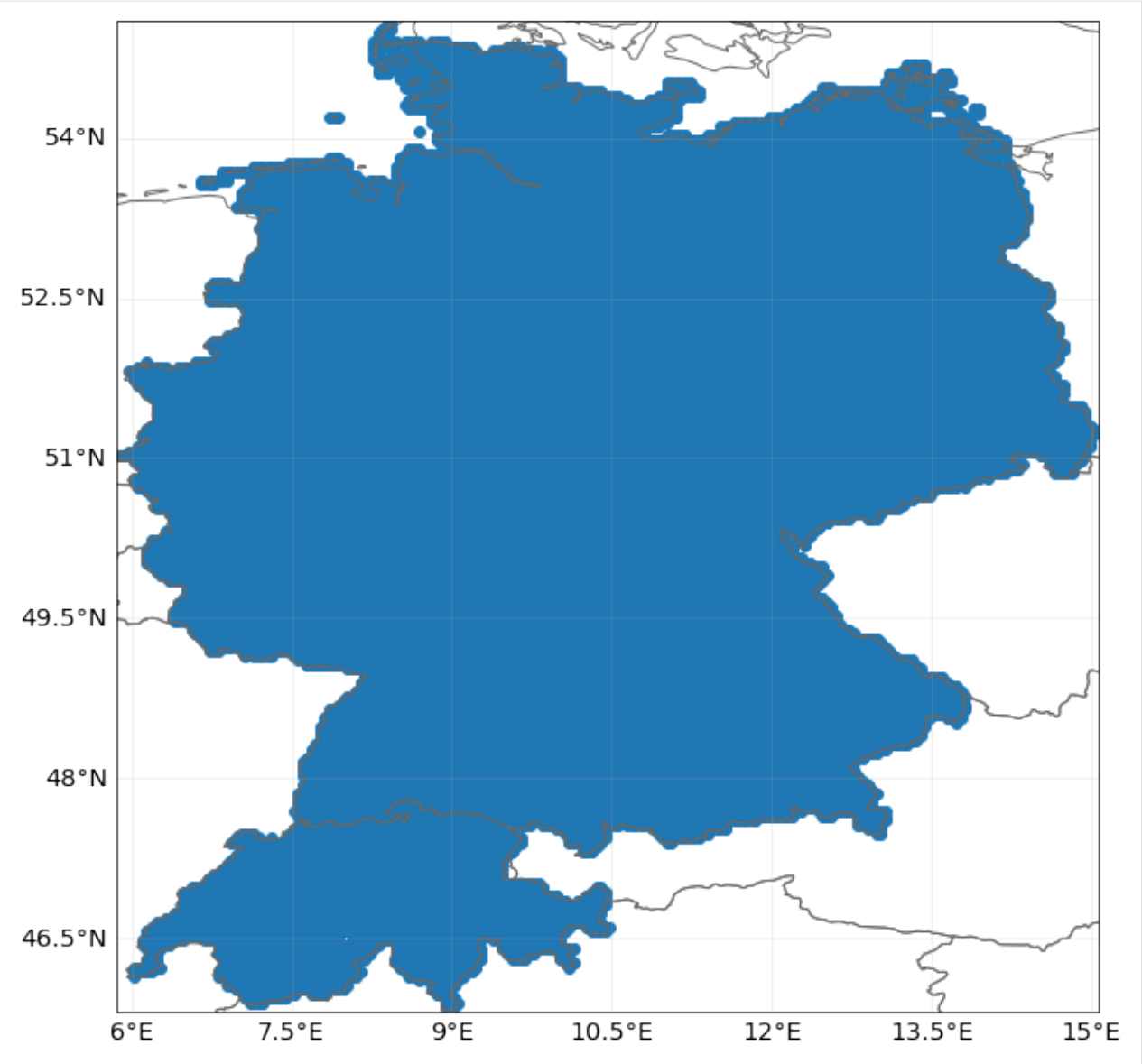


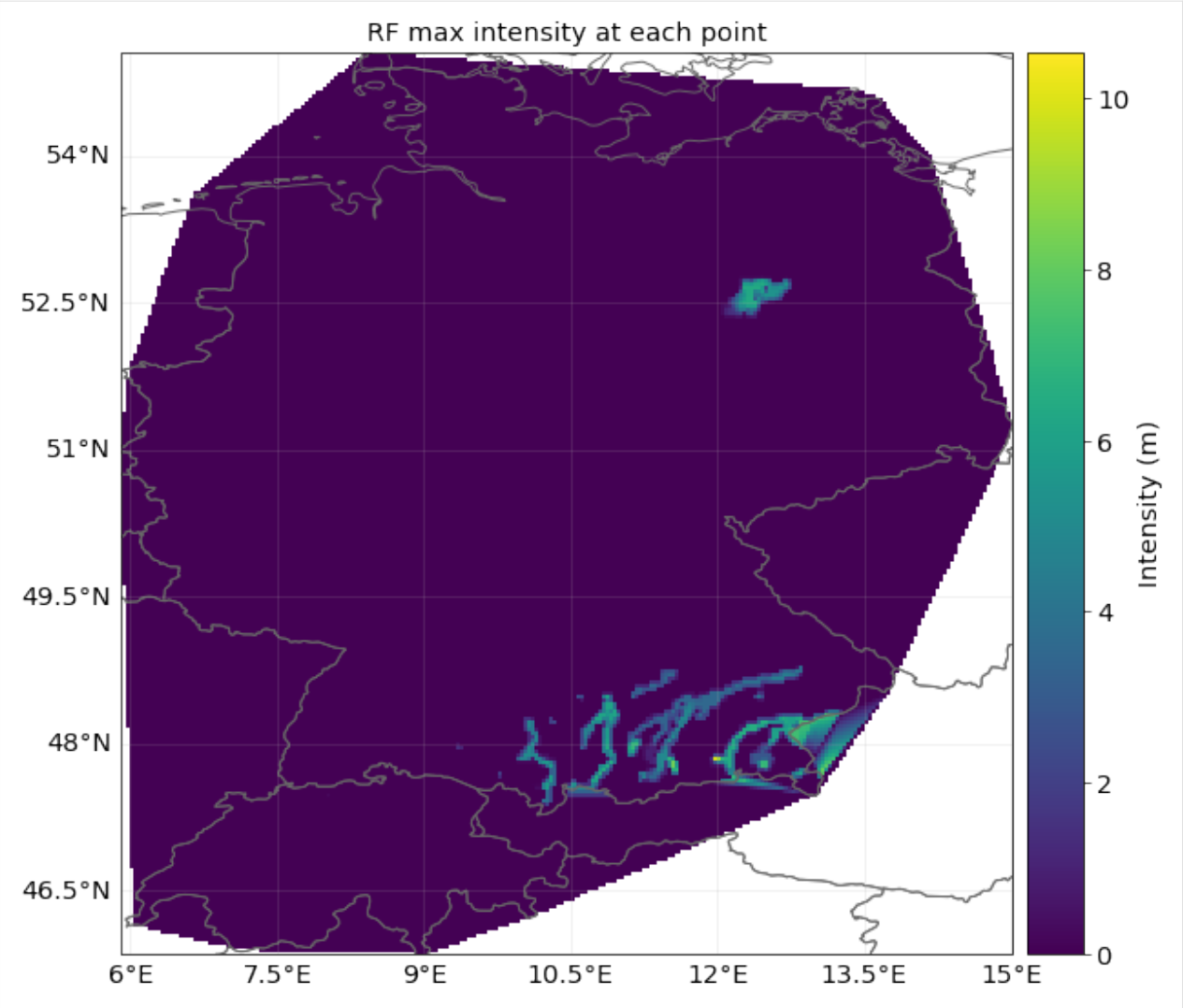
[3]:
```
# plotting intensity (Flood depth in m)
rf.plot_intensity(event=0, smooth = False);
```



Setting flood with ISIMIP NatIDGrid:

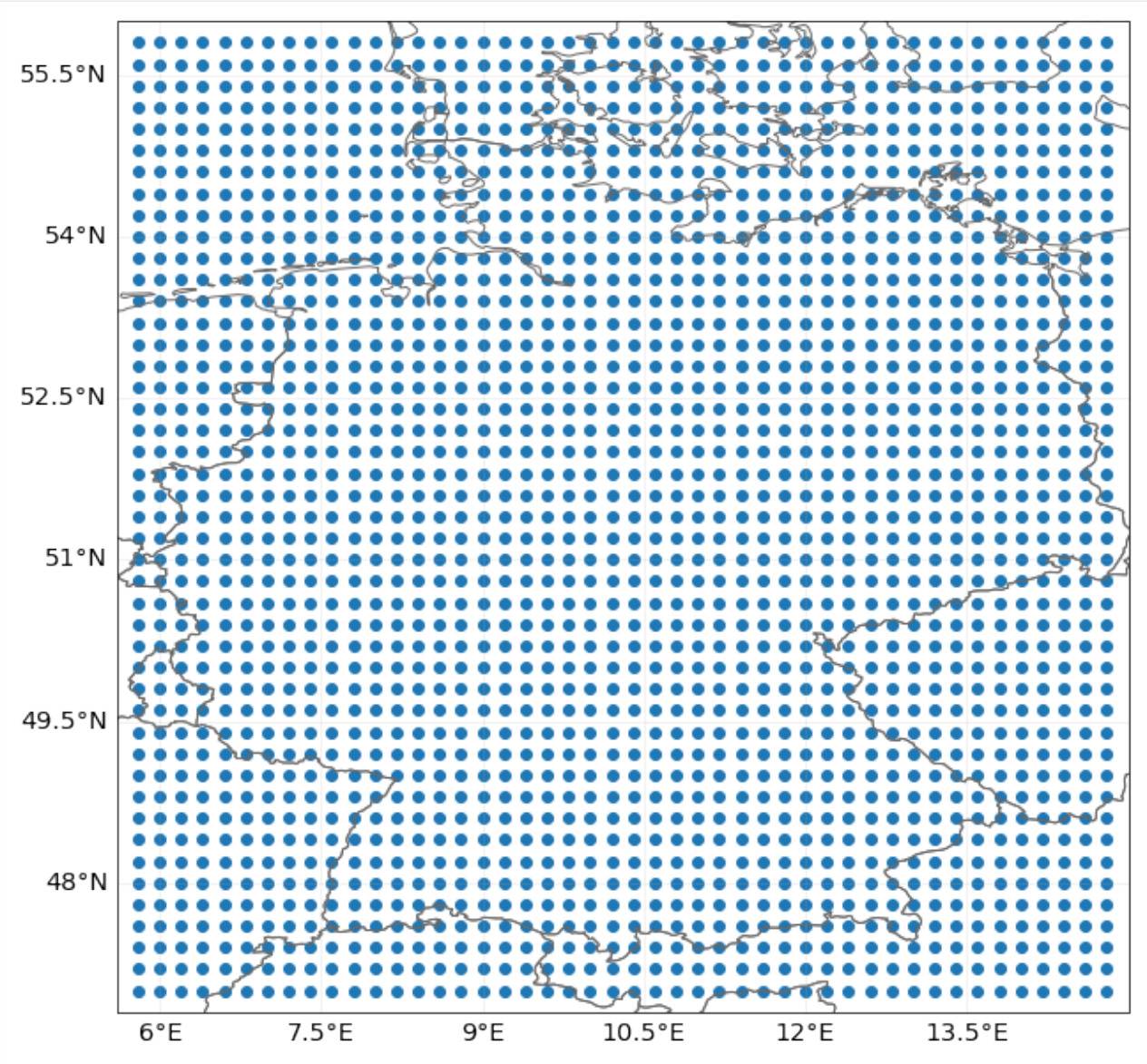
[4]:
```
# generating RiverFlood hazard from netCDF file, using the ISIMIP NatIDGrid␣
↪(according to ISIMIP standards) with a resolution of 150as (aprox 5km)
# setting centroids for a region
rf_isi = RiverFlood.from_nc(countries = ['DEU','CHE'], years=years, dph_path=HAZ_DEMO_
↪FLDDPH, frc_path=HAZ_DEMO_FLDFRC, ISINatIDGrid=True)
rf_isi.centroids.plot()
rf_isi.plot_intensity(event=0, smooth = False);
```

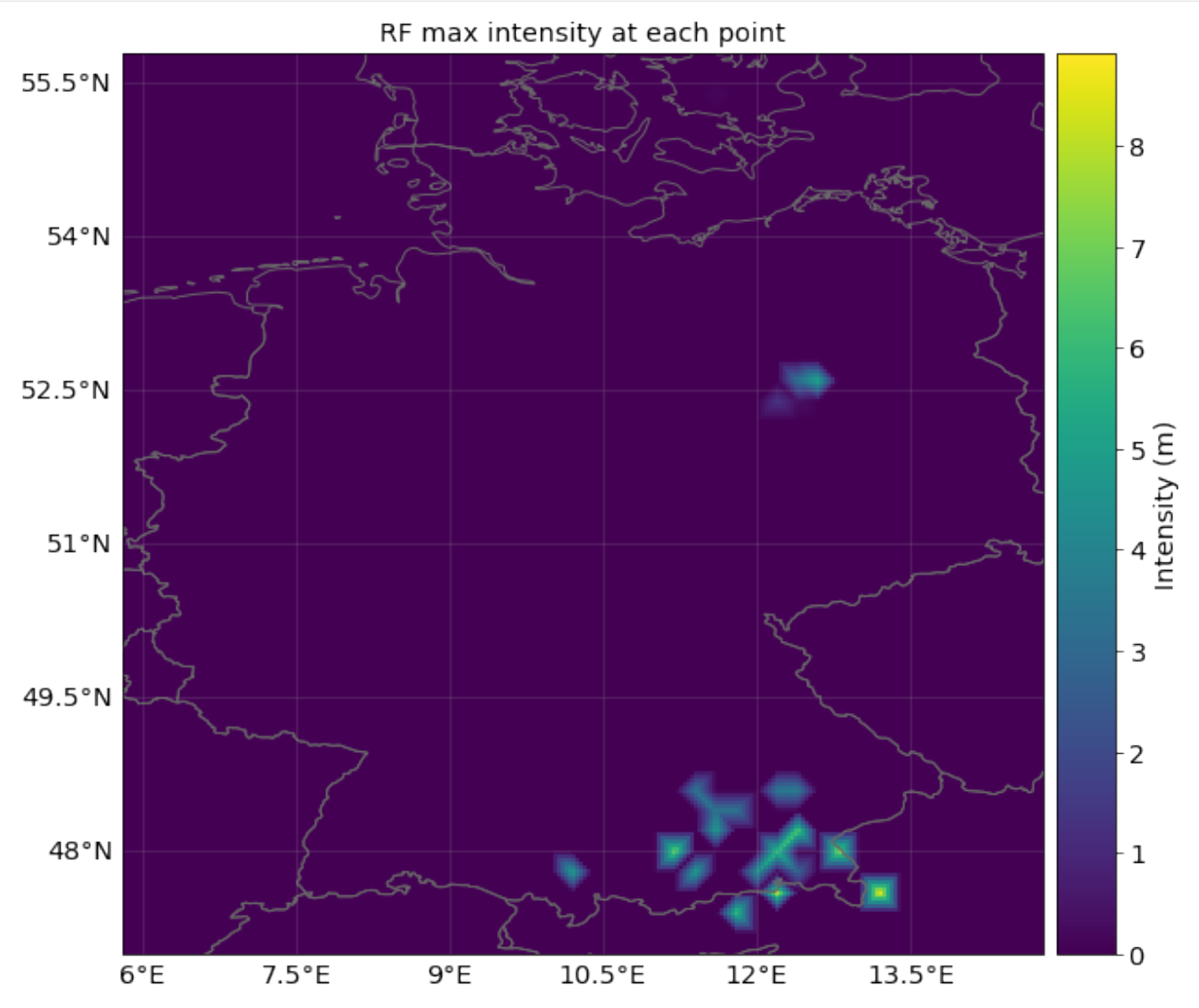

54°N
52.5°N
51°N
49.5°N
48°N
46.5°N
6°E
7.5°E
9°E
10.5°E
12°E
13.5°E
15°E



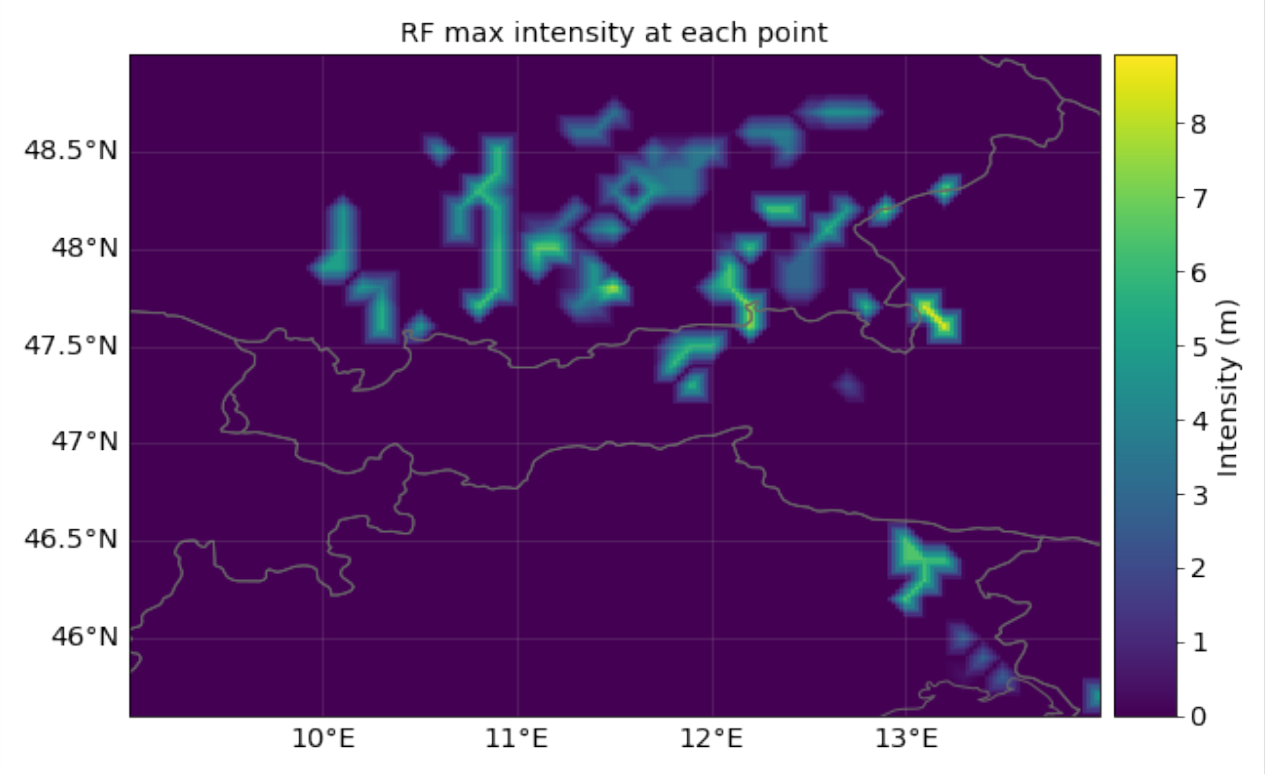
Setting flood with random points as coordinates:

[5]:
```
lat = np.arange(47, 56, 0.2)
lon = np.arange(5.8, 15, 0.2)
lon, lat = np.meshgrid(lon, lat)
rand_centroids = Centroids.from_lat_lon(lat.flatten(), lon.flatten())
rf_rand = RiverFlood.from_nc(dph_path=HAZ_DEMO_FLDDPH, frc_path=HAZ_DEMO_FLDFRC,
                     centroids=rand_centroids, ISINatIDGrid=False)
rf_rand.centroids.plot()
rf_rand.plot_intensity(event = 0);
```


55.5°N
54°N
52.5°N
51°N
49.5°N
48°N
6°E
7.5°E
9°E
10.5°E
12°E
13.5°E



[6]:
```
# setting random poits using raster
min_lat, max_lat, min_lon, max_lon = 45.6 , 49., 9., 14.
cent = Centroids.from_pnt_bounds((min_lon, min_lat, max_lon, max_lat), res=0.1)
rf_rast = RiverFlood.from_nc(dph_path=HAZ_DEMO_FLDDPH, frc_path=HAZ_DEMO_FLDFRC,
                    centroids=cent, ISINatIDGrid=False)
rf_rast.plot_intensity(event=0);
```



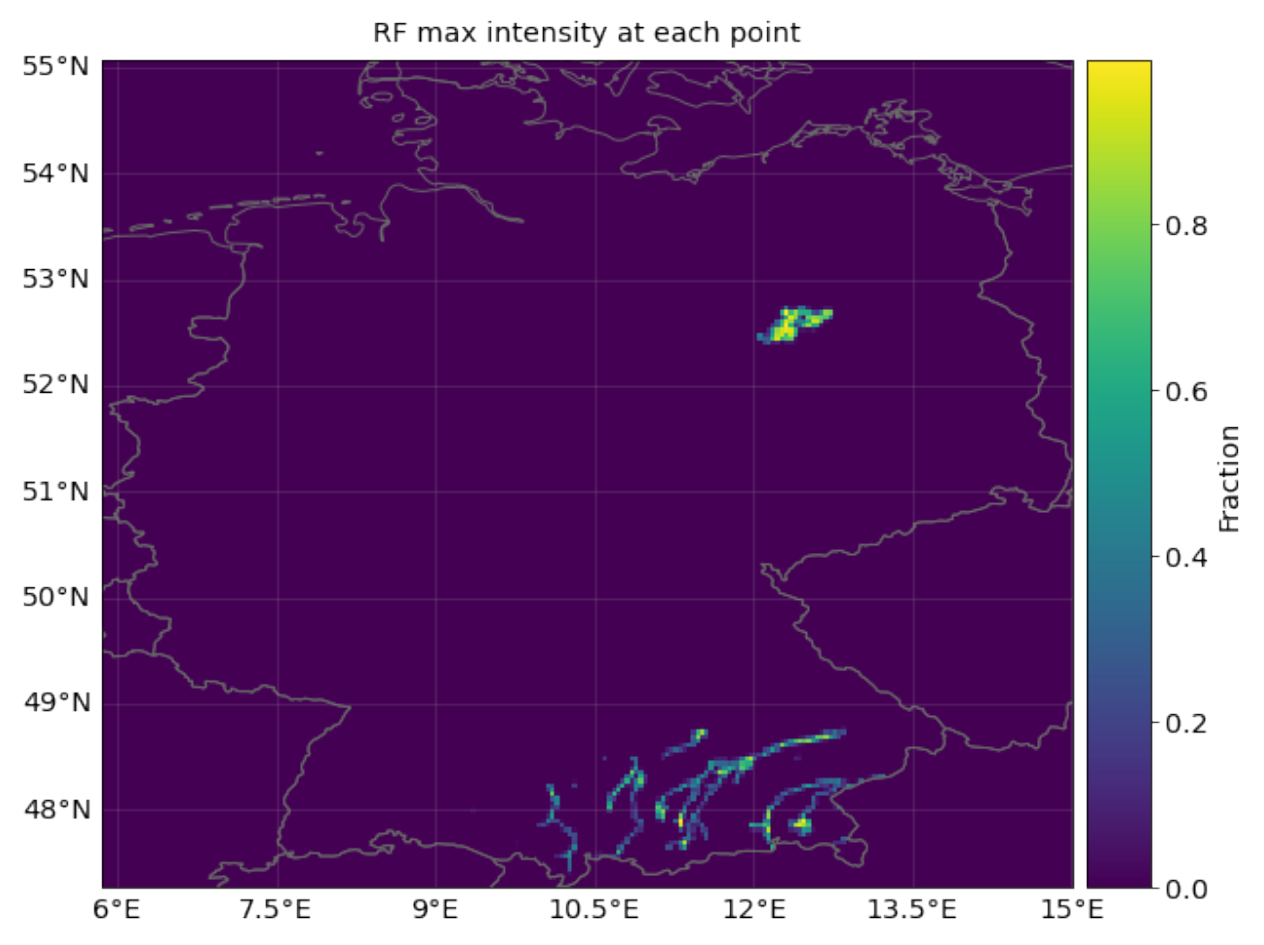
Part 3: Calculating Flooded Area

The fraction indicates the flooded part of a grid cell. It is possible to calculate the flooded area for each grid cell and for the whole area under consideration

As ISIMIP simulations currently provide yearly data with the maximum event, event and yearly flooded area are the same.

[7]:
```
#setting river flood
rf_DEU = RiverFlood.from_nc(countries = ['DEU'], years=years, dph_path=HAZ_DEMO_
↪FLDDPH, frc_path=HAZ_DEMO_FLDFRC)
rf_DEU.plot_fraction(event=0, smooth = False)
# calculating flooded area
rf_DEU.set_flooded_area()
print("Total flooded area for year " + str(years[0]) + " in Germany:")
print(str(rf_DEU.fla_annual[0]) + " m2")

print("Total flooded area at first event in Germany:")
print(str(rf_DEU.fla_event[0]) + " m2");
```

```
Total flooded area for year 2000 in Germany:
2437074832.038133 m2
Total flooded area at first event in Germany:
2437074832.038133 m2
```



[8]:
```
#calculate flooded area
rf_DEU.set_flooded_area(save_centr = True)
print("affected area in each affected centroid and each event:")
rf_DEU.fla_ev_centr.data
```

```
affected area in each affected centroid and each event:
```

[8]:
```
array([  584715.81718075,  5053615.42987214,   584715.81718072,
         772660.21477247,  4635961.19139216,  4844788.31063239,
         877073.72577108,    62648.1284788 ,   793542.93642074,
         710012.09844901,   104512.31458989,  7942935.27615602,
       12980428.9975574 ,  8862643.59587924, 11015597.79960522,
        8151960.31900815,  8026545.7605037 ,  8110155.46617279,
        3428003.77254628,  2550100.30565756,  5748176.99827292,
       10743865.08816197, 10116791.51696033,   167219.69117698,
         167377.62651458,  2426975.61490724,  7866748.53143359,
       12448711.57332617, 10691246.31833798,  3807841.25590698,
         795043.7594347 ,  9958969.25866094, 10858623.66475107,
       10586635.08712338,  9938046.70065339,  9414992.10340605,
        4100752.00183631,    20922.20331432,  4020851.5658597 ,
        8628077.07461108, 12271973.68427333, 11036399.71240725,
        6135986.84213667,  5863741.76936025,  5319251.23373377,
        7936993.24829797, 12104437.71477011, 12041612.40883577,
        9444812.29294032,   628258.03278596,  1907510.51222833,
        3731174.16397245,  9034472.37433945, 13017186.7293233 ,
```

(continues on next page)

(continued from previous page)

```
 8300814.59013924,  2242896.97806211,  4003675.58205338,
 8636201.25119447, 10795251.3687714 ,  7839658.22402078,
 7986389.93703723,   859427.78209519,  1573601.99913549,
 6755998.31491316, 10910307.66297655, 12127226.06552194,
12630778.64271599, 12295077.44566373,  8329600.4117508 ,
  524533.99971186,   629440.81919467,   314720.4095973 ,
  881217.10779167,   251776.33744806,    84004.22287586,
 5649284.31387142,  1722086.62090805,  2142107.81474468,
 7413373.17151569, 11676586.99808064, 12075607.56251821,
 9051455.12336371, 11823594.53327545,  7224363.59517164,
   42041.47990802,  4666604.29120217,  4099044.47915512,
 3951899.15417815,  8975856.20966334, 12255091.63637071,
 8198088.95831024,  9333208.58240435,  9627499.2323574 ,
  588580.71259438,    84161.6504909 ,  2566930.35528144,
 4060799.89490651,   841616.56614469,  1283465.17764072,
   63121.24246085,   778495.28694244,   862656.94355686,
  252484.96984341,    68119.64171431,   340598.1953546 ,
  476837.45234932,  1977089.19581193,  9340041.72726154,
11998885.92035427,  5272237.71440184,    45450.32455399,
  704480.06861677,  3431499.68736199,  2727019.56583407,
 5022260.92439447,  2727019.56583394,  3317873.85448186,
 4526009.59398457, 11417371.18678022,  2001451.49577753,
 2092426.62153581,    90975.06618462,   545850.43682358,
 1182675.89349668,  3047664.70063641,  4662472.38853107,
10484877.04798255, 10507620.35283201, 11758527.53769297,
 7914831.06916955,  5344785.09035629,  3411564.99185211,
  159206.36416826,  3824076.16877868,  5963738.09206057,
   45524.71676818,   751157.84158056,  2412809.97877638,
 6669371.29305688,  9833339.32540515, 11290130.36798235,
10334111.21979271, 11358416.90487568,  8035113.05446658,
 4666283.71550905,  1616127.4966119 ,   637346.02812978,
  159466.56281482,   888456.57183127,  2505903.20571988,
 4715654.29155094,  4077787.82812764,    91123.75112706,
 1412418.21871583,  4419502.17166185, 11322126.22505713,
 7289900.62057462,  3576607.18698346,  2323655.79628766,
  797332.8671151 ,   432837.83608639,  1117175.77323329,
 2439547.17032033,  4559900.9592355 ,  2941136.28432971,
 1117175.77323329,    68398.51916611,    45599.00945964,
 2553544.50319797,  3807517.28822123,  7843030.09154589,
 3579522.19779226,   569987.61990444,   136908.36527585,
  114090.30439655,    22818.05921908,  1300629.43027509,
 4152887.05347047,   912722.43517239,   136908.36527586,
  410725.06926384,  1460355.79002089,  1551628.08666564,
 2624076.8815839 ,  7507142.09570224,  2760985.22029591,
  250998.65639055,   159726.41287331,   319452.82574662,
 2329333.4694195 ,  3950732.208999  ,  4750013.19093546,
 1164666.73470981,   137019.61741205,   936300.67910442,
 1210339.86075792,  1027647.03754179,   137019.61741205,
  685098.06047493,  1415869.35333926,  4932706.12049274,
10459163.73034007, 12423111.55332744,  4658666.93883924,
  411058.82565086,  1278849.65617129,   913464.11608036,
  274261.58936941,  4731012.2436776 ,   137130.7946847 ,
 6925104.95863252,  4822432.61382616,  4548171.0776705 ,
 4593881.26274455,  3771096.65427761,  4868142.79890019,
 4616736.35528191,  9530590.19067662,  8776370.85982062,
 8959211.60011772,   297116.70851342,  1898512.78921767,
 7502556.71861495,   343104.72928439,   548967.58815774,
```

(continues on next page)

(continued from previous page)

```
       7273820.62297533,   160115.5332318 ,  3133689.83594421,
       6427495.36489048,  8874975.84877951,  7479682.68299575,
       4048635.70969342, 10430383.85545637,  7777040.88549149,
       3682657.53061551,    45784.30480911, 10805096.61452431,
       3845881.56399029,  5471224.64954781,    91568.60961822,
        114460.77035089,  5448332.53545232,  1304852.74202514,
       3731420.9935142 , 11995488.25307395, 10667743.07715331,
       4715783.60520665,  2541028.92856497,  4051911.01724683,
         68676.46221054,  4375933.02392118,  7514691.61028689,
        618587.43214996,  8568581.198868  ,  2084868.78441238,
         45821.28892426,  2703456.21656234,  6254606.13986513,
       3642792.77453402,    91642.57784853,  3001294.52289039,
       4673771.75699323,    91642.57784852,  6094231.47026817,
        824783.20730461,    68731.9383873 ,   458212.92258201,
       2909651.88503098,  4765414.18148062,  5819303.77006197,
         68787.37702238,   275149.50808951,  2476345.5194196 ,
       8850642.43013321,  1467463.93637202,  5525919.03388077,
      10547397.32962378,  1467463.93637202,   710802.88255137,
       1903117.31083391,  4838045.18357794,  7612469.24333563,
       1329889.20902028,   321007.73274474,   802519.38524791,
       6213792.8841836 ,   550299.016179  ,  1444534.85073734,
       4539966.64323985,    45858.24801163,  2292912.40725453,
       3714518.2791294 ,  4677541.58413571,  5021478.29574301,
       4654612.49850087,  5663493.97477634,  1811400.75475137,
        389795.12311364,   343936.87176537,  4058454.9907367 ,
       4933732.36289262,  3878142.80500866,  2616025.48722109,
       3786352.59451101,  8743032.51002805,   688427.75417088,
       1032641.57782735,  6677749.35437335,  6471220.95332144,
        206528.32090837,  3740457.48926219,  7526810.08377282,
       3327400.68715837,  6035216.59859321,  4222356.94923925,
       1468645.82569746,  4061723.65343595,  3373296.00612329,
       1973492.94515705,   114737.9634809 ,   160633.13551601,
        481899.40654803,  5507422.0333676 ,  7848076.67537898,
       4773099.06708954,  4658361.30396748,  1858754.96831889,
       2340654.42829594,   688427.75417092,  3465086.43033687,
       4314147.15973668,  2180021.13249286,  7389124.76802673,
       5094365.23126325,  2273638.55111259,  7027610.14485272,
       2296604.5581947 ,  8727097.23558467,   206694.41130696,
        620083.26065688,  6384560.66322575,   987540.01563459,
       1377962.77769441,  4409481.0597325 ,  5833375.63770317,
        160762.31693474,  2664061.31317241,  2916687.81885159,
        183728.36412085,  2503299.04970953,  3031518.06815012,
       1056438.14382492,  2939653.82593355,  6476425.11933014,
       4478379.08097884,  1377962.77769448,   826777.64522785,
       2687027.32025452,  5029564.10650116,    45932.09103021,
        160891.41056458, 10848678.76457667,   436705.2801817 ,
       2206510.95693681,  9354686.58283404,  1333100.28954366,
       3585580.03744784,  1103255.47846846,   436705.2801817 ,
       2183526.29387852,  3585580.03744784,   344767.32364266,
       4527944.15886665,  2987983.50724635,  2022634.93682888,
       3884378.30254839,    22984.48746242,   137906.93480856,
       3930347.62866516,  4665851.28097746,  1195193.38149251,
        551627.73923423,   344767.32364266,  3792440.50655434,
       3861393.85355002,  3585580.03744784,  1103255.47846846,
       1356084.84557202,  4117522.08266347,  6026764.41462543,
       4485568.99342198,  2484315.14800192,  3841487.11382578,
        368046.66974845,   782099.20334174,  7199913.35353265,
```

(continues on next page)

(continued from previous page)

```
  713090.46113229,  2001253.84542028,  4761603.96225989,
  230029.18532961,  4025510.56920494,    23002.91685928,
 2553323.78309566,  2070262.48051402,  4462566.11505762,
  644081.66536506,   552070.04479106,   966896.00520992,
 6353888.18738154,    46042.66739333,  7597040.27065209,
  368341.33914669,  1473365.35658675,  1427322.76624445,
 6468995.09204306,  4028733.5158435 ,   184170.66957333,
   23021.33369667,  1588472.04684531,  3107880.20817732,
 2394218.77145437,   713661.38312219,  1151066.68818345,
 4028733.5158435 ,  3430178.76937919,   851789.36855201,
   69064.00611507,   138238.43785107,  2027496.98119534,
 2741728.95032525,    92158.95186193,  9123736.41537856,
   46079.47593097,  1105907.50280863,  1797099.55795509,
10367882.75836551,  5183941.37918276,  4792265.63092952,
   46079.47593097,  4630987.30591678,    69119.21892554,
 1727980.35244044,  1105907.50280863,  4907464.12797535,
 7142318.83300202,    46079.47593097,  2119655.99340619,
 4354510.48385843,    23039.73796548,   990708.79118845,
 6220729.56918928,   484220.71775385,   322813.81183591,
 6686857.57690364,   184465.03725218,  1867708.59780709,
 1567952.8770407 ,  2582510.49468729,  8854322.43234125,
11690472.35828338,   622569.53260236,  1775476.01878391,
 6940497.00815814,   299755.69392329,  4196579.87598521,
  484220.71775387, 10883437.88237945,   391988.21926018,
 1867708.59780728,  4035172.80900787,    46116.25931304,
 1292284.4771703 ,  7915242.58385565,    69229.53131939,
 1615355.70392125,  3023022.74672283,   207688.58052588,
 6230657.68442242,  9969052.29507587,   138459.06263879,
11261336.34241269,  2099962.43658919,    23076.50876076,
 5561438.76077762,  1476896.56068868,  2076885.75152965,
11953632.16603328,   369224.14017217,  1823044.15012417,
 8007548.46442769,   276918.12527756,   346423.14080041,
 5658244.43590979,  3210187.66745817,  3210187.66745817,
  323328.25040788,  1662831.03282448,  4087792.9646555 ,
  484992.37561182,  3810654.49503272,  4572785.39403927,
14388107.73464195,   577371.88341003,  1570451.57879818,
 4318741.76103709,  3048523.59602618,   415707.7582061 ,
 5450390.9063244 ,  6674418.96791818,  3325662.06564879,
 2632815.89159201, 11131730.17885479, 14226443.44812288,
 8175585.9293106 ,  2218870.16200874,  3258965.45627431,
  901415.93604765,   184905.83338038,  3582550.55369736,
  254245.52594315,   138679.38512554,  2704247.80814296,
  346698.44936018, 13428786.49400053,  1479246.66704298,
 3490097.79172438,  1433020.28605656,  1317454.11833161,
 3883022.46062701,  8806140.64603687,  7350007.4923756 ,
 6818402.8194786 ,  9846235.50978571,  8783027.02502618,
 4691985.84996017, 12481144.39222511, 13729258.40093017,
12527370.77321223, 10239161.03972273,   138679.38512554,
  670815.55181327,  3053367.36157378,  3539130.39008407,
  300710.42442124,   208184.13584106,  2775788.54969063,
 1734867.78969926,    69394.71643513,  3770446.13846321,
  532026.14587166,  3284683.10995292,  1919920.36685962,
 2752657.01793862,  2382551.86361778,   693947.13742262,
 1387894.27484531,  1434157.4460641 ,  3354077.70520894,
12699232.90566386,  1272236.40065568,  2081841.3045532 ,
 2428814.92712191,  7656550.36654593,  6800681.96828465,
  717078.72303205,    23149.8991283 ,  3403035.29818859,
```

(continues on next page)

(continued from previous page)

```
        3310435.64103781,   347248.49871514,  4375330.94367087,
         902846.06431938,  4954078.74696315,   810246.51496873,
         578747.47989191,  4143831.90859385,  4467930.81642188,
        8171914.94645002,  2500189.23386921,  5486526.39827995,
         925996.03250709,    46299.7982566 ,   208349.09383909,
        3266718.48119539,  2919195.18801533,   417027.87629611,
         671878.24814055,  4957998.37854157,  4934829.77060901,
        1876625.52424655,  3336223.01036871,   810887.57620106,
        8386894.52601658,    46336.43032497,  4401960.85052897,
        3730082.87210194,    46373.03711108,   231865.2024258 ,
        2921501.41560192,  1159325.93115107,  3918521.55659509,
          69559.56072774,   347797.79014237,  1669429.3494952 ,
        2156346.28808573,  6469038.64831591, 12798958.41810963,
        1808548.44395802,  3895335.0768415 ,    69559.56072774,
        1669429.34949511,  1738988.89672652,  2133159.80833214,
          23186.51855554,    23204.80929843,   116024.05493402,
         835373.14149704,  6195684.08774647,    69614.43296041,
        3202263.7270813 ,  5105058.14695738,    46409.61859686,
        3619950.24380173,  3411107.09349742,   139228.86592083,
        3643155.01426779,  7843225.38416266,   348072.15129509,
          46409.61859686,  9142695.12359154,   696144.30259014,
         348346.32255877,  1254046.78283975,  6618580.07454613,
        1045038.91360584,   441238.6788458 ,   464461.78143516,
        9266012.36390282,  2090077.82721168,  1394481.2151967 ,
        2788962.43039341,  5229304.71932662,  7111854.17585799,
         418344.35373641,  1301515.74312973,  2672755.64442255,
        2091721.71456896,  3114341.2579497 ,  3230548.04392058,
        1882549.67298334,  5024075.05294284,  4748680.79119586,
         116480.37319707,  4286477.62517144,   163200.01148239,
         536228.64017519])
```

Part 4: Generating ISIMIP Exposure

The exposed assets are calculated by means of national GDP converted to total national wealth as a proxy for asset distribution, downscaled by means of data from spatially explicit GDP distribution. Data for past (1971-2010) and future (2005-2100) periods can be accessed at ISIMIP, https://www.isimip.org/ .

More information on spatially explicit GDP time series:

Geiger, T. (2018) 'Continuous national gross domestic product (GDP) time series for 195 countries: Past observations (1850-2005) harmonized with future projections according to the Shared Socio-economic Pathways (2006-2100)', Earth System Science Data. Copernicus GmbH, pp. 847–856. doi: 10.5194/essd-10-847-2018.

Murakami, D. and Yamagata, Y. (2019) 'Estimation of gridded population and GDP scenarios with spatially explicit statistical downscaling', Sustainability (Switzerland). Multidisciplinary Digital Publishing Institute, 11(7), p. 2106. doi: 10.3390/su11072106.

[9]:
```
# set exposure for damage calculation
from climada_petals.entity.exposures.gdp_asset import GDP2Asset
from climada_petals.util.constants import DEMO_GDP2ASSET
gdpa = GDP2Asset()
gdpa.set_countries(countries = ['CHE'], ref_year = 2000, path=DEMO_GDP2ASSET)
gdpa.gdf
```

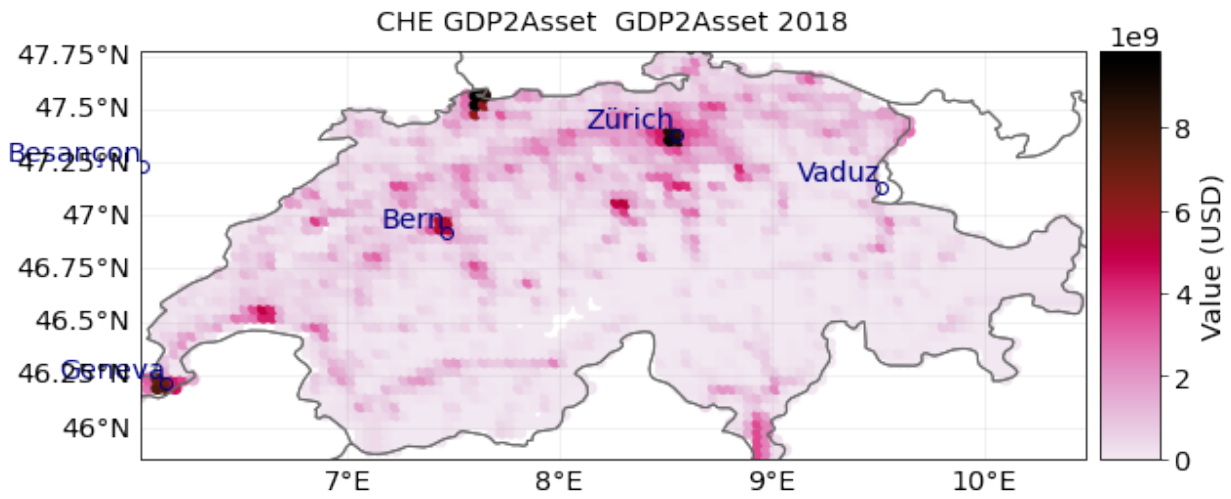
[9]:
```
             value   latitude  longitude  impf_RF  region_id
0     3.556720e+09  45.853916   8.937364      3.0       11.0
```

(continues on next page)

(continued from previous page)

```
1      2.400900e+09  45.853916   8.979031      3.0        11.0
2      2.250146e+08  45.853916   9.020698      3.0        11.0
3      2.939545e+08  45.895583   7.104034      3.0        11.0
4      3.476220e+08  45.895583   7.145701      3.0        11.0
...             ...        ...        ...      ...         ...
2755   1.935802e+08  47.770580   8.520698      3.0        11.0
2756   1.665060e+08  47.770580   8.562365      3.0        11.0
2757   2.254221e+08  47.770580   8.604032      3.0        11.0
2758   4.107628e+08  47.770580   8.645698      3.0        11.0
2759   6.403091e+08  47.770580   8.687365      3.0        11.0

[2760 rows x 5 columns]
```

[10]:
```
from matplotlib import colors
norm=colors.LogNorm(vmin=1.0e2, vmax=1.0e10)
gdpa.plot_scatter();
```



Part 5: Setting JRC damage functions

In CLIMADA we currently calculate damage by translating flood-depth into a damage factors. Damage assessments implemented in CLIMADA base on the residential damage functions basing on an empirical estimate published in the JRC report. Individual damage functions are available for six continents:

RF1: Africa RF2: Asia RF3: Europe RF4: North America RF5: Oceania RF6: South America

For further information on depth-damage functions, see also:

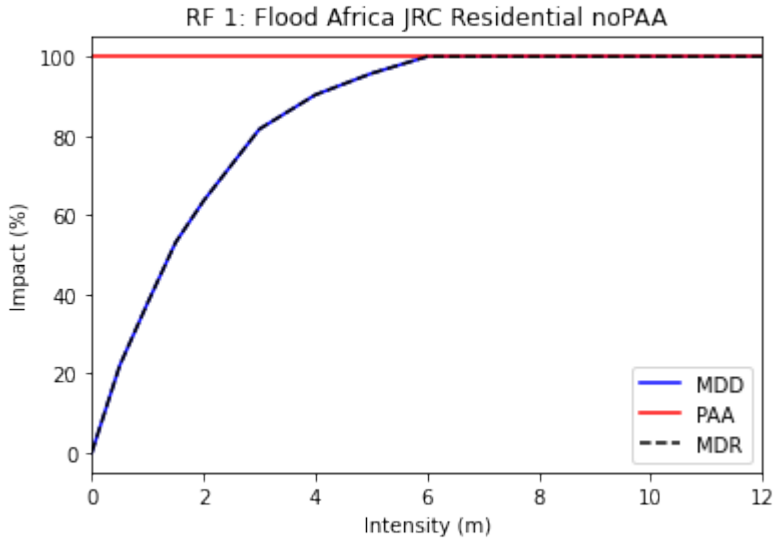
Huizinga, J., Moel, H. de and Szewczyk, W. (2017) Global flood depth-damage functions : Methodology and the Database with Guidelines, Joint Research Centre (JRC). doi: 10.2760/16510.

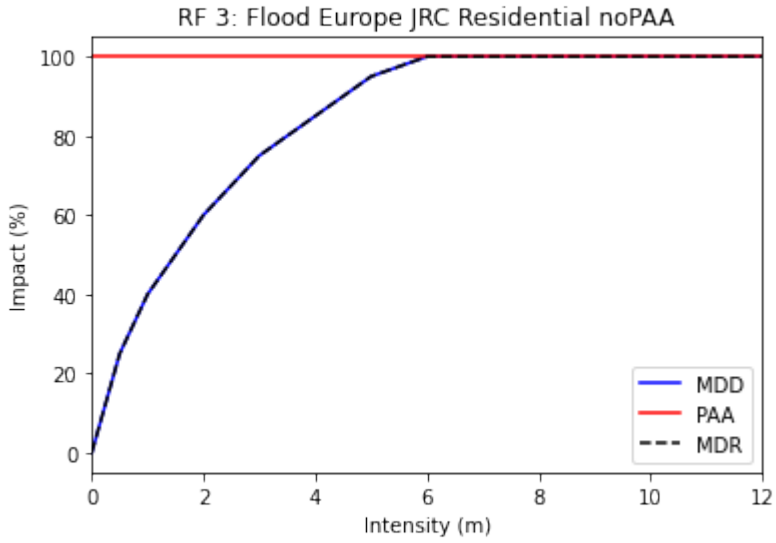
[11]:
```
# import impact function set for RiverFlood using JRC damage functions () for 6␣
↪regions
from climada_petals.entity.impact_funcs.river_flood import ImpfRiverFlood,flood_imp_
↪func_set
impf_set = flood_imp_func_set()
impf_AFR = impf_set.get_func(fun_id=1)
impf_AFR[0].plot()
```

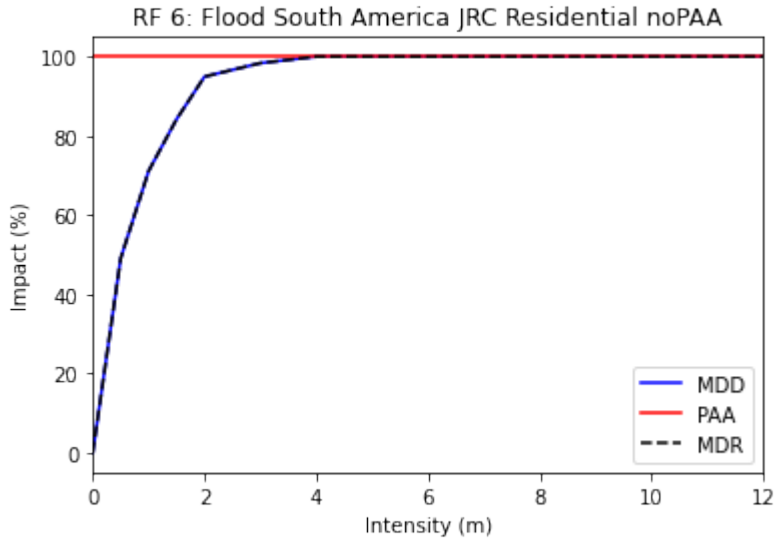
(continues on next page)

(continued from previous page)

```
impf_EUR = impf_set.get_func(fun_id=3)
impf_EUR[0].plot()
impf_OCE = impf_set.get_func(fun_id=6)
impf_OCE[0].plot();
```







The plots illustrate how flood-depth is translated into a damage factor (0%-100%). The damage factor is then multiplied with the exposed asset in each grid cell to derive a local damage.

Linking exposures to the correct impact function

If the ISIMIP exposure presented above is used, the correct impact function ID is automatically provided in the GeoDataFrame:

[12]:
```
gdpa.gdf
```

[12]:

	value	latitude	longitude	impf_RF	region_id
0	3.556720e+09	45.853916	8.937364	3.0	11.0
1	2.400900e+09	45.853916	8.979031	3.0	11.0
2	2.250146e+08	45.853916	9.020698	3.0	11.0
3	2.939545e+08	45.895583	7.104034	3.0	11.0
4	3.476220e+08	45.895583	7.145701	3.0	11.0
…	…	…	…	…	
2755	1.935802e+08	47.770580	8.520698	3.0	11.0
2756	1.665060e+08	47.770580	8.562365	3.0	11.0
2757	2.254221e+08	47.770580	8.604032	3.0	11.0
2758	4.107628e+08	47.770580	8.645698	3.0	11.0
2759	6.403091e+08	47.770580	8.687365	3.0	11.0

[2760 rows x 5 columns]

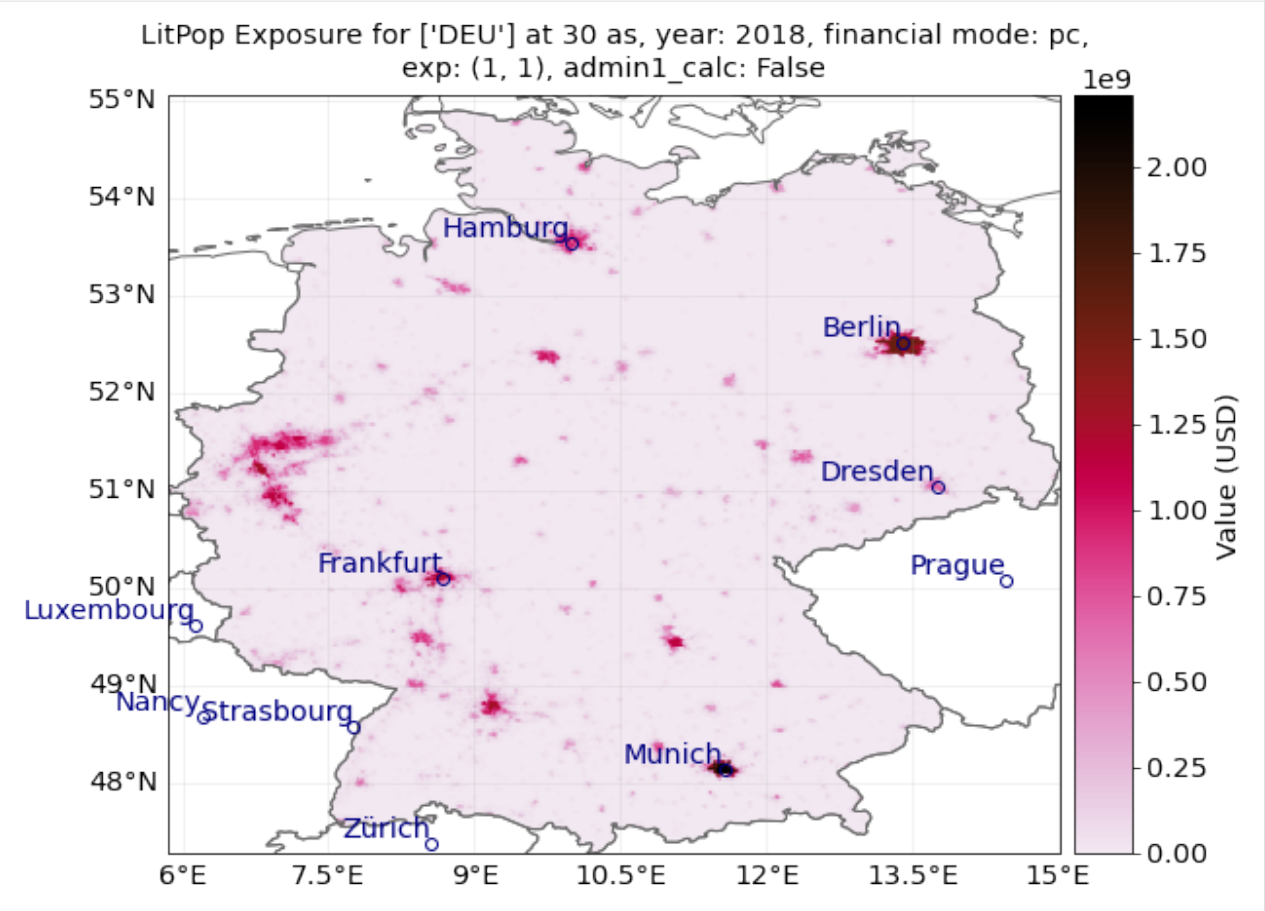
The column 'impf_RF' indicates the ID of the impact function (in this case 3 for Europe). If other Exposure data is used the impact function needs to be set manually.

Part 6: Deriving flood impact with LitPop exposure

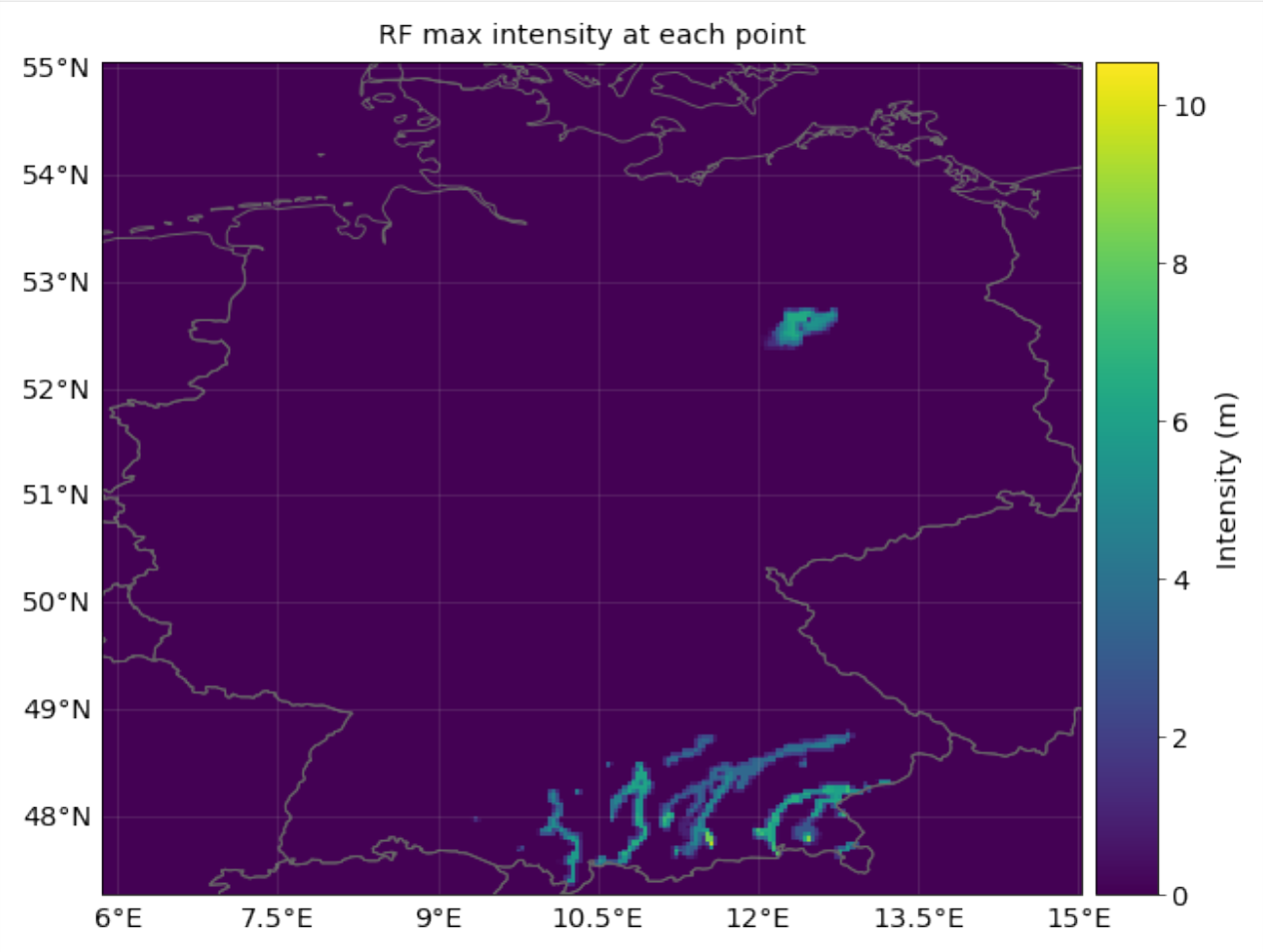
[13]:
```
from climada.entity import LitPop
lp_exp = LitPop.from_countries(['DEU'], fin_mode='pc')
lp_exp.gdf
```

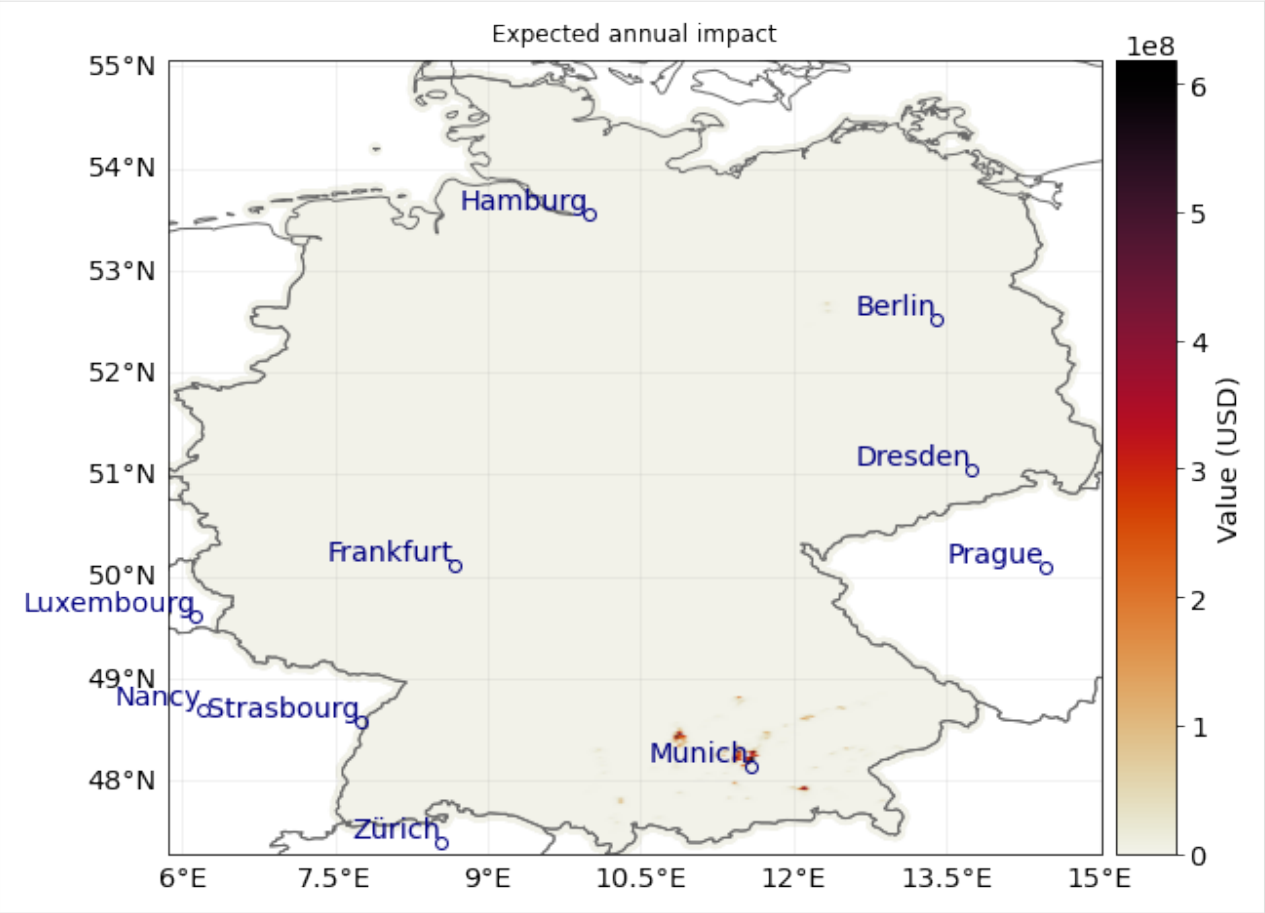
[13]:
```
                value                    geometry   latitude  longitude  \
0         83527.518952  POINT (6.72083 53.61250)  53.612500   6.720833
1        129830.381955  POINT (6.72917 53.61250)  53.612500   6.729167
2        175137.241455  POINT (6.73750 53.61250)  53.612500   6.737500
3        213973.213450  POINT (6.74583 53.61250)  53.612500   6.745833
4        227763.852006  POINT (6.75417 53.61250)  53.612500   6.754167
…                  …                          …         …          …
661391   250520.745603  POINT (8.28750 54.76250)  54.762500   8.287500
661392   217939.745256  POINT (8.29583 54.76250)  54.762500   8.295833
661393    85890.946407  POINT (8.27917 54.75417)  54.754167   8.279167
661394   249873.621559  POINT (8.28750 54.75417)  54.754167   8.287500
661395   121538.038970  POINT (8.28750 54.74583)  54.745833   8.287500

         region_id  impf_
0              276      1
1              276      1
2              276      1
3              276      1
4              276      1
…              …        …
661391         276      1
661392         276      1
661393         276      1
661394         276      1
661395         276      1

[661396 rows x 6 columns]
```

[14]:
```
# In the LitPop exposure the damage function for river floods needs
# to be specified manually.
import pandas as pd
from climada_petals.util.constants import RIVER_FLOOD_REGIONS_CSV

info = pd.read_csv(RIVER_FLOOD_REGIONS_CSV)
lp_exp.gdf['impf_RF'] = info.loc[info['ISO']=='DEU','impf_RF'].values[0]
lp_exp
lp_exp.plot_hexbin(pop_name=True);
```



[15]:
```
from climada.engine import Impact

rf = RiverFlood.from_nc(countries = ['DEU'], years=years, dph_path=HAZ_DEMO_FLDDPH,␣
↪frc_path=HAZ_DEMO_FLDFRC)
imp=Impact()
imp.calc(lp_exp, impf_set,rf,save_mat=True)
rf.plot_intensity(0)
imp.plot_scatter_eai_exposure();
```


RF max intensity at each point
55°N
54°N
53°N
52°N
51°N
50°N
49°N
48°N
6°E
7.5°E
9°E
10.5°E
12°E
13.5°E
15°E
10
8
6
4
2
0
Intensity (m)



2.4 ECMWF OPERATIONAL FORECAST TRACKS

The `TCForecast` class extends the `TCTracks` class with methods to download operational ECMWF ensemble tropical storm track forecasts, read the BUFR files they're contained in and produce a `TCTracks` object that can be used to generate `TropCyclone` hazard footprints.

ECMWF publishes ensemble forecasts for tropical storms multiple times a day. Each is stored in a BUFR file containing encoded information on the storm's location, wind speed, central pressure and other variables (when they exist for that ensemble member). Together, all the ensemble members provide a great probabilistic view of how tropical storms may develop over the coming days.

Downloading the latest ECMWF data and to create a `TCTracks` is straightforward:

[1]:
```
from climada_petals.hazard import TCForecast
import logging
import datetime

logging.getLogger('climada.hazard.tc_tracks_forecast').setLevel(logging.WARNING)
print("Processing ECMWF forecast. Date: " + str(datetime.datetime.now()))

forecast = TCForecast()
forecast.fetch_ecmwf()
```

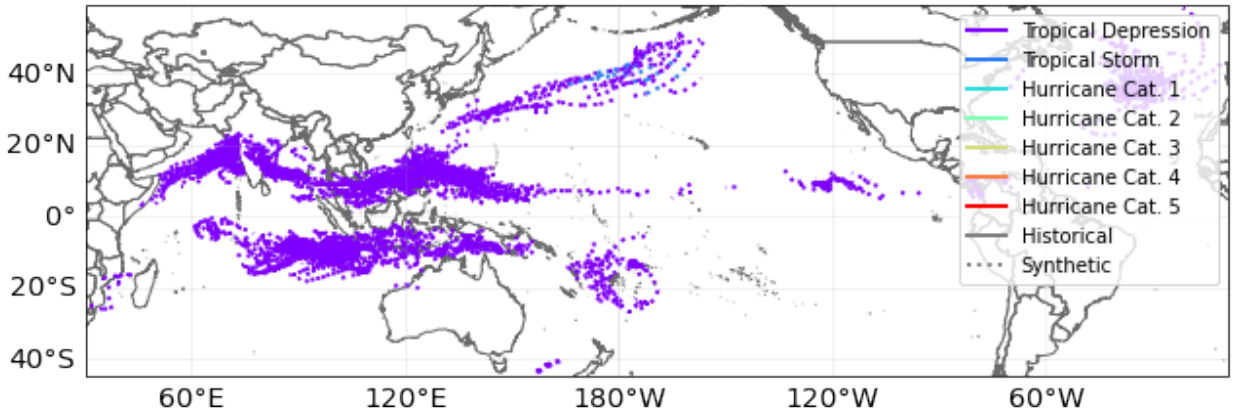
```
Processing ECMWF forecast. Date: 2021-11-16 17:49:27.152261
2021-11-16 17:49:28,816 - climada_petals.hazard.tc_tracks_forecast - INFO - Fetching␣
↪BUFR tracks:
```

```
Download: 100%|██████████| 42/42 [00:07<00:00,  5.45 files/s]
Processing: 100%|██████████| 42/42 [00:04<00:00,  8.92files/s]
```

This has created a `TCTracks` object that works the same way as, for example, tracks read from IBTrACS. We can visualise them:

[2]:
```
forecast.plot()
```

[2]:
```
<GeoAxesSubplot:>
```



The `TCForecast` class can also read from already-downloaded BUFR files, with the `file` and `path` parameters in its `fetch_ecmwf` method. See the documentation for how to use these.

This is especially useful if you're setting up automated cron jobs, where the computing environment may need you to separate the downloads from the file processing.

You can save operational products to the local machine by using the `TCForeacst.fetch_bufr_ftp` method and providing the `target_dir` parameter. Otherwise they are saved as temporary files. See the method documentation for more details.

You can also process past operational products by using the `TCForeacst.fetch_bufr_ftp` method and providing the `remote_dir` parameter with the format `yyyymmddhhmmss`, e.g. `'20200730120000'`. See the method documentation for more details. The ECMWF ftp server keeps operational products going back about six months (though due to changes in BUFR formatting, not all of them are guaranteed to be readable - see the next section).

2.4.1 Working with the BUFR format

This should be enough for you to work with ECMWF tropical storm forecasts. This section is for people who want/need to modify the code.

Unfortunately, the BUFR file format is a nightmare to work with. If you're lucky, you'll never need to interact with it, and these scripts will do all the work for you.

This isn't anyone's fault: it's a very old format that prioritises compact information storage over human readability. BUFR files are binary-encoded, and can be read with the ECMWF ecCodes package (amongst other options). Once decoded they're *still* not human readable, and consist of long strings of alphanumeric codes without the necessary metadata to decode them. See Bufr Format in a Nutshell for a not-very-helpful overview of the format.

2.4.2 Troubleshooting

Usually if something stops working it's because ECMWF have made a change to the BUFR format. (Note that this makes the code here incompatible with older operational forecast files. CLIMADA version 2.2.0 contains code to work with BUFR version 35 if you need it).

Since the code here uses ECMWF's own ecCodes package to read in files, some updates to the BUFR

We've comment the code so that the next time it breaks it'll be easier to fix.

If you're unlucky and have to fix these scripts, there are a few things that helped us last time.

- Information on changes to the BUFR format: go here first if something breaks! It also tells you about upcoming changes: in theory we can prepare for them!
- Sample ECMWF FORTRAN and python code for reading TC tracks from BUFR files: this guided the current code. Check its publication date to make sure it works with the latest BUFR formatting!
- Visualisations of current tropical cyclone forecasts: for each storm you can see which ensemble members have tracks present, see them on a map, and check that the `TCForecast` tracks match.

2.5 Hazard: Tropical cyclone rain from R-CLIPER or TCR model

The `TCRain` class models precipitation generated by tropical cyclones. Given a `TCTracks` instance, it computes the precipitation rates (in mm/h) for each historical and/or synthetic track at every centroid and track position. The precipitation rates are then translated into total amounts of rainfall (in mm) by multiplying with each track's time step sizes and summation over the whole storm life time. `TCRain` inherits from `Hazard` and has an associated hazard type `"TR"`.

2.5.1 Model description: R-CLIPER

The statistical model **R-CLIPER** (Tuleya et al. 2007) assumes a radially symmetric distribution of the precipitation rate around the storm center. The shape of the radial profile increases linearly from the storm center to the boundary of the inner core, and decreases exponentially outside of it:

$$T_0 + (T_m - T_0)(r/r_m) \quad \text{for} \quad r \leq r_m,$$

$$T_m \exp(-(r - r_m)/r_e) \quad \text{for} \quad r > r_m,$$

where

- r_m : the radial extent of the inner-core precipitation rate,
- r_e : a measure for the total radial extent of the tropical rain field,
- T_0 : the precipitation rate at the center of the storm (at $r = 0$),
- T_m : the maximum precipitation rate (at $r = r_m$).

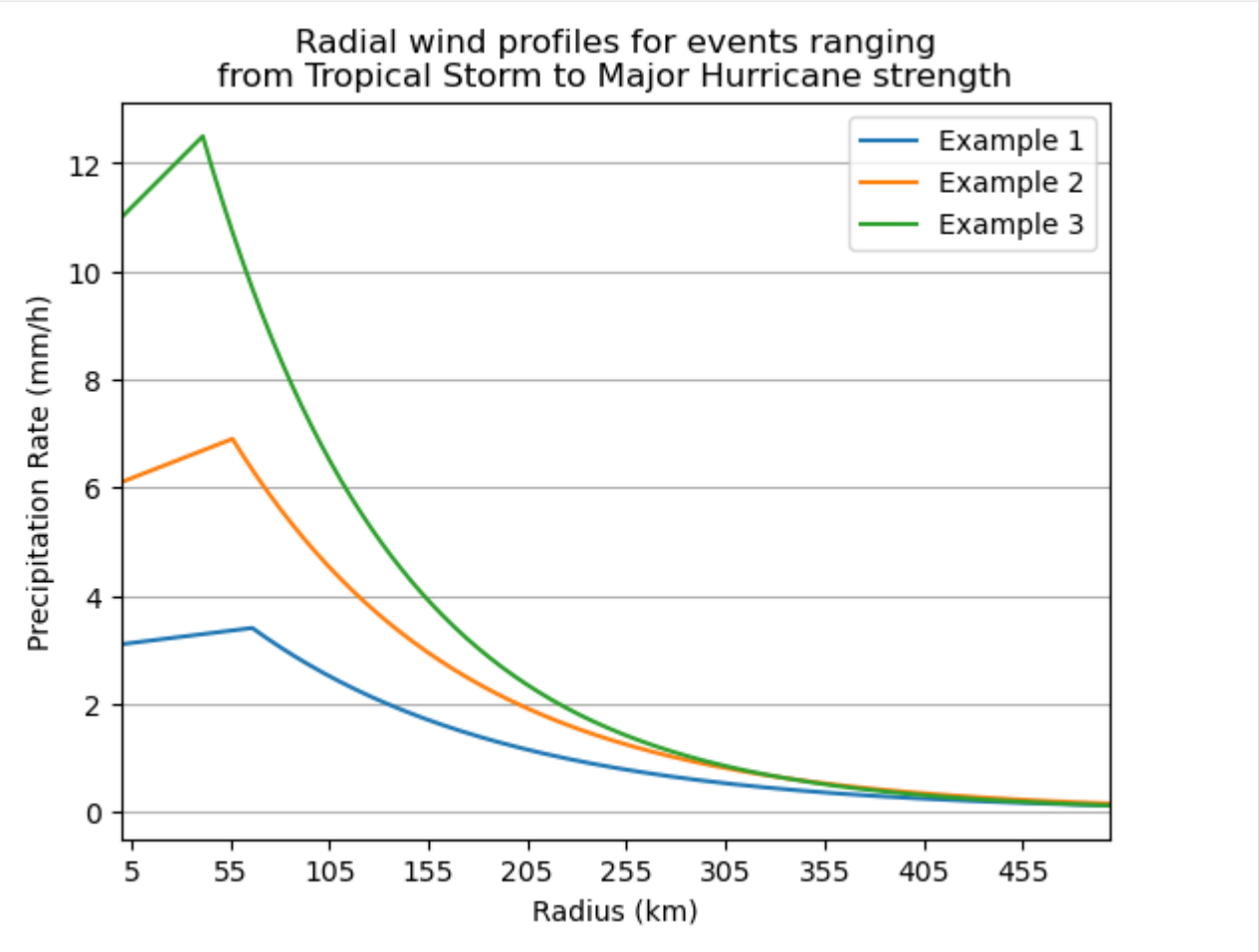
These four quantities are estimated via linear statistical relationships from the maximum wind speed variable of the storm track, which is the only along-track variable that is considered in this model except from the storm's position.

[1]:
```
%matplotlib inline
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 100

import numpy as np
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
l_T0 = np.array([3.1, 6.1, 11.0])
l_Tm = np.array([3.4, 6.9, 12.5])
l_rm = np.array([66.0, 56.0, 41.0])
l_re = np.array([128.0, 116.0, 98.0])

radius = np.arange(0, 500, dtype=float)

for i, (T_0, T_m, r_m, r_e) in enumerate(zip(l_T0, l_Tm, l_rm, l_re)):
    rainrate = np.zeros_like(radius)
    msk = radius <= r_m
    rainrate[msk] = T_0 + (T_m - T_0) * (radius[msk] / r_m)
    rainrate[~msk] = T_m * np.exp(-(radius[~msk] - r_m) / r_e)
    plt.plot(radius, rainrate, label=f"Example {i + 1}")

plt.title(
    "Radial wind profiles for events ranging\n"
    "from Tropical Storm to Major Hurricane strength"
)
plt.xticks(np.arange(5, 500, 50))
plt.xlim(radius.min(), radius.max())
plt.yticks(np.arange(0, 13, 2))
plt.grid(axis="y")
plt.xlabel('Radius (km)')
plt.ylabel('Precipitation Rate (mm/h)')
plt.legend()
plt.show()
```



2.5.2 Model description: TCR

The physics-based model **TCR** (Zhu et al. 2013, Emanuel 2017) assumes that the precipitation rate at each centroid is proportional to the vertical vapor flux $w_q = q_s \cdot w$, where q_s is the saturation specific humidity and w is the vertical velocity. The implementation in CLIMADA follows the description in Lu et al. 2018 and includes the improvements proposed in Feldmann et al. 2019. The saturation specific humidity q_s is assumed to be constant over the rain field and can be estimated from the 600 hPa temperature at the storm center using the Claudius-Clapeyeron relation. The focus of this model is on an accurate description of the vertical velocity w as the sum of the following five components:

- w_f : friction-induced (interaction between horizontal winds and the surface roughness),
- w_h : topographically induced (interaction between horizontal winds and the surface slope),
- w_t : vortex stretching-induced (changes in the storm's vorticity over time),
- w_s : shear-induced (baroclinic interaction of the upper and lower troposphere),
- w_r : radiative cooling-induced (constant subsidence everywhere on Earth is assumed).

The calculation of these components requires horizontal TC wind fields as implemented as part of CLIMADA's `TropCyclone` class. While any TC wind model can be combined with TCR, this implementation chooses `"ER11"` by default since this is the one that is used in most studies that apply TCR. Note that the default wind model in `TropCyclone` is `"H08"`.

In addition to common along-track variables, the TCR model requires four additional variables:

- gridded topography (surface elevation),
- gridded drag coefficients (e.g. derived from surface roughness),
- along-track 600 hPa temperature at the storm center,
- along-track 850 hPa wind speeds averaged over an annulus of 200-500 km around the storm center.

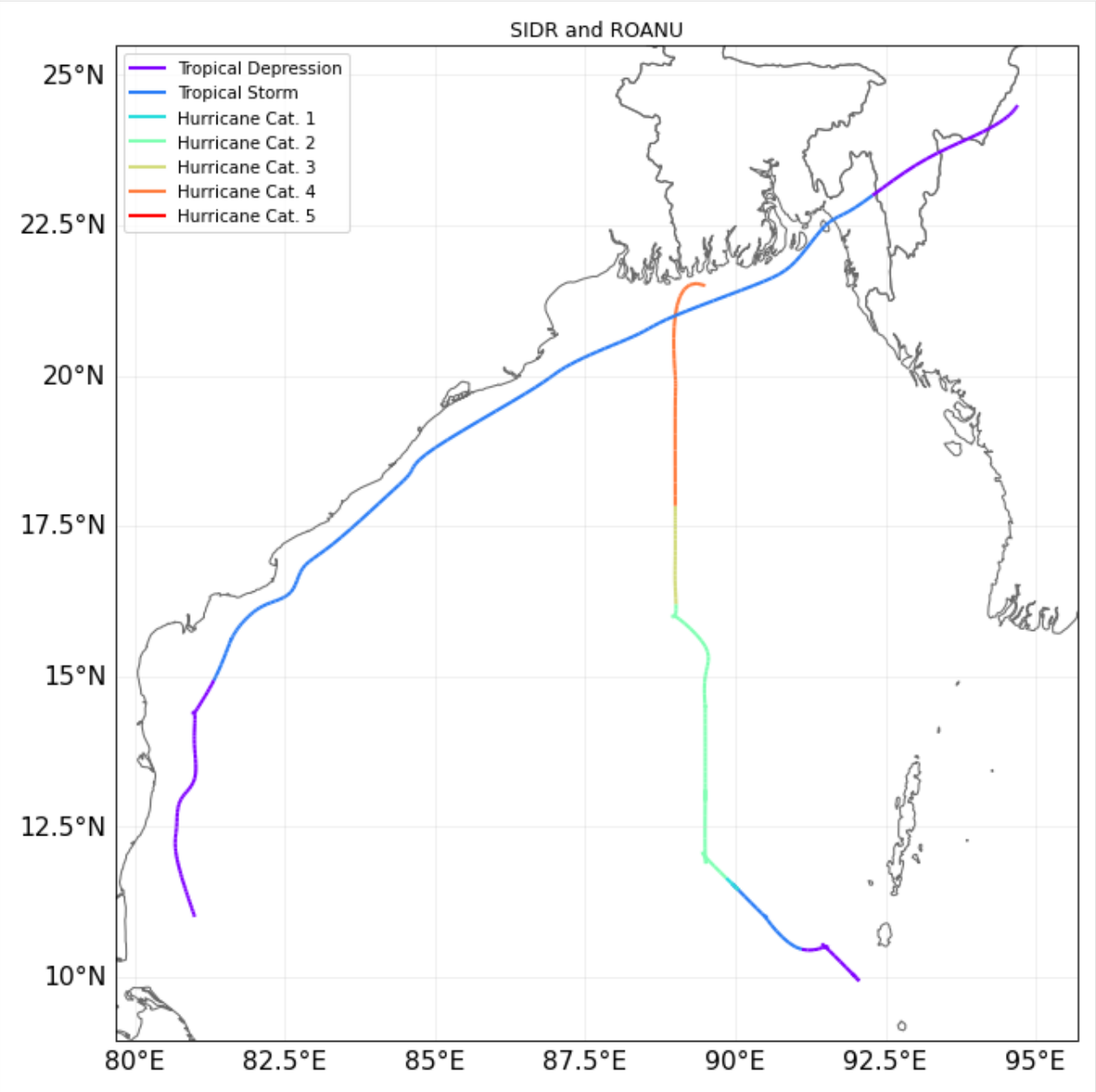
While default data sets for topography and drag coefficients are provided with CLIMADA and do not need further considerations for most applications, the additional temperature and wind speed variables need to be provided by the user. For existing TC track data sets, both variables can be extracted from reanalysis and climate model data. For example, when using historical IBTrACS records, it is recommended to extract the variables from the ERA5 reanalysis which provides global hourly data on pressure levels. On the other hand, providers of synthetic track sets based on climate models (e.g. CHAZ) can often provide both variables (as extracted from the climate model outputs) along with the track data.

2.5.3 Examples

We compute the rain fields of Sidr 2007 and Roanu 2016 over Bangladesh as follows:

[2]:
```
mpl.rcParams['figure.dpi'] = 75

from climada.hazard import TCTracks

tracks = TCTracks.from_ibtracs_netcdf(
    # SIDR 2007 and ROANU 2016
    storm_id=['2007314N10093', '2016138N10081'],
)
tracks.equal_timestep(0.5)
ax = tracks.plot()
ax.get_legend()._loc = 2
ax.set_title('SIDR and ROANU');
```

```
2023-07-17 15:15:35,050 - climada.hazard.tc_tracks - INFO - Progress: 50%
2023-07-17 15:15:35,062 - climada.hazard.tc_tracks - INFO - Progress: 100%
2023-07-17 15:15:35,075 - climada.hazard.tc_tracks - INFO - Interpolating 2 tracks to␣
↪0.5h time steps.
```



[3]:
```
from climada.hazard import Centroids

# define centroids and restrict to points on land
min_lat, max_lat, min_lon, max_lon = 21.0, 24.5, 88.5, 92.5
cent_bang = Centroids.from_pnt_bounds((min_lon, min_lat, max_lon, max_lat), res=0.015)
cent_bang.set_dist_coast(signed=True, precomputed=True)
cent_bang.dist_coast *= -1.0
cent_bang.dist_coast[cent_bang.dist_coast < 0] = 9000e3
cent_bang.check()
```

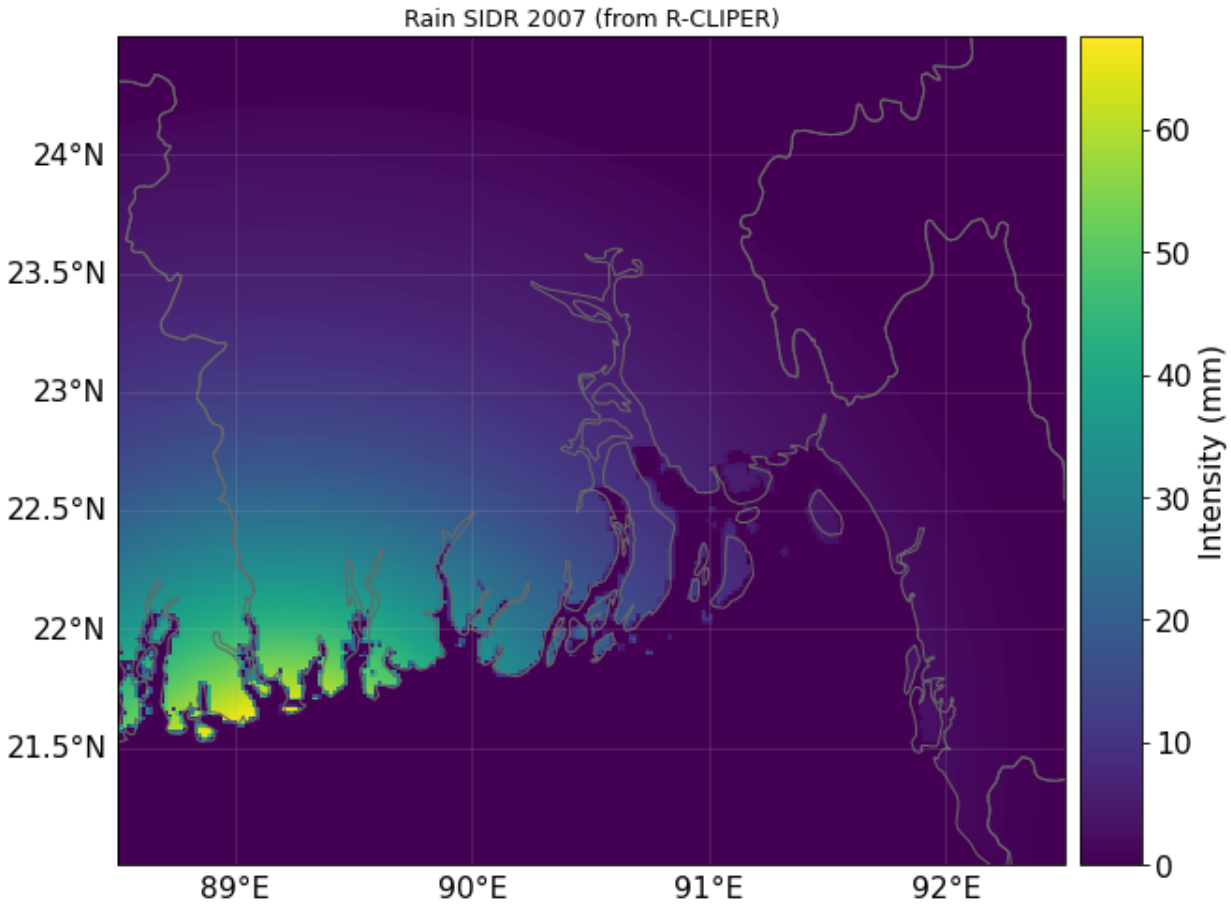
```
2023-07-17 15:15:39,168 - climada.util.coordinates - INFO - Sampling from SYSTEM_DIR/
↪GMT_intermediate_coast_distance_01d.tif
```

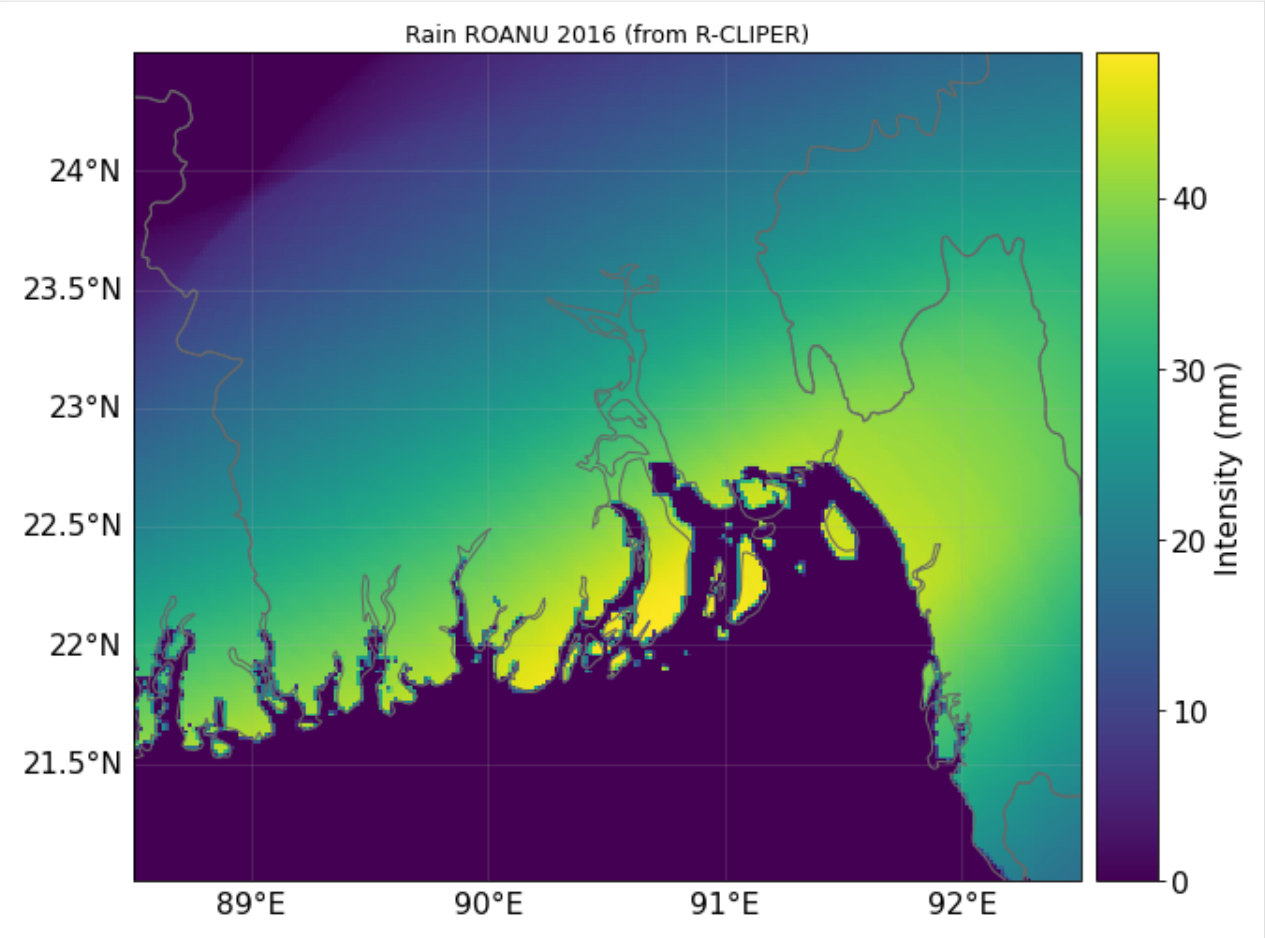
Rain field output with R-CLIPER

[4]:
```
from climada_petals.hazard import TCRain

tr_bang = TCRain.from_tracks(tracks, centroids=cent_bang, model="R-CLIPER")
ax = tr_bang.plot_intensity(1)
ax.set_title('Rain SIDR 2007 (from R-CLIPER)')
ax = tr_bang.plot_intensity(2)
ax.set_title('Rain ROANU 2016 (from R-CLIPER)');
```

```
2023-07-17 15:15:39,266 - climada_petals.hazard.tc_rainfield - INFO - Mapping 2␣
↪tracks to 45839 coastal centroids.
2023-07-17 15:15:40,859 - climada_petals.hazard.tc_rainfield - INFO - Progress: 50%
```

```
2023-07-17 15:15:43,117 - climada_petals.hazard.tc_rainfield - INFO - Progress: 100%
```



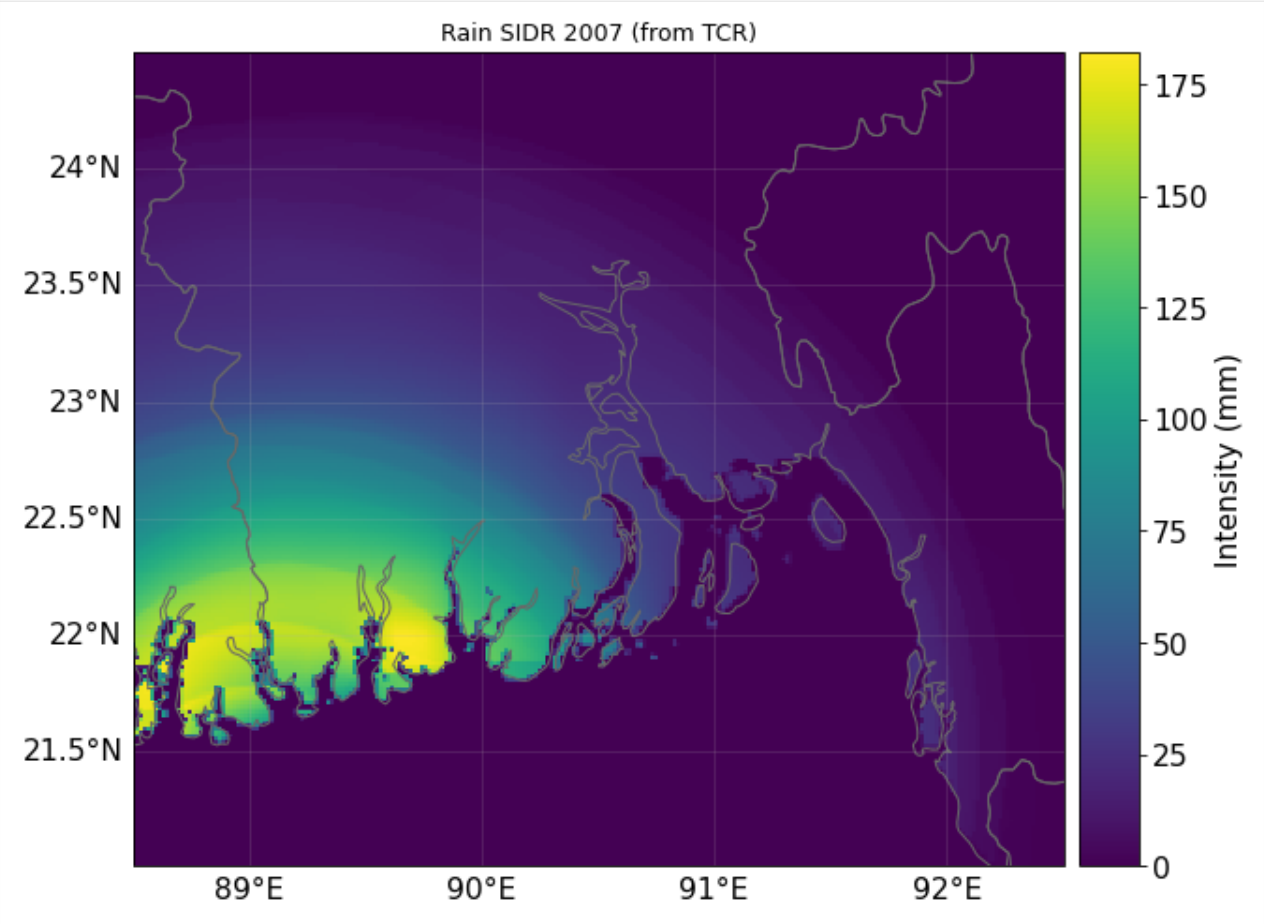


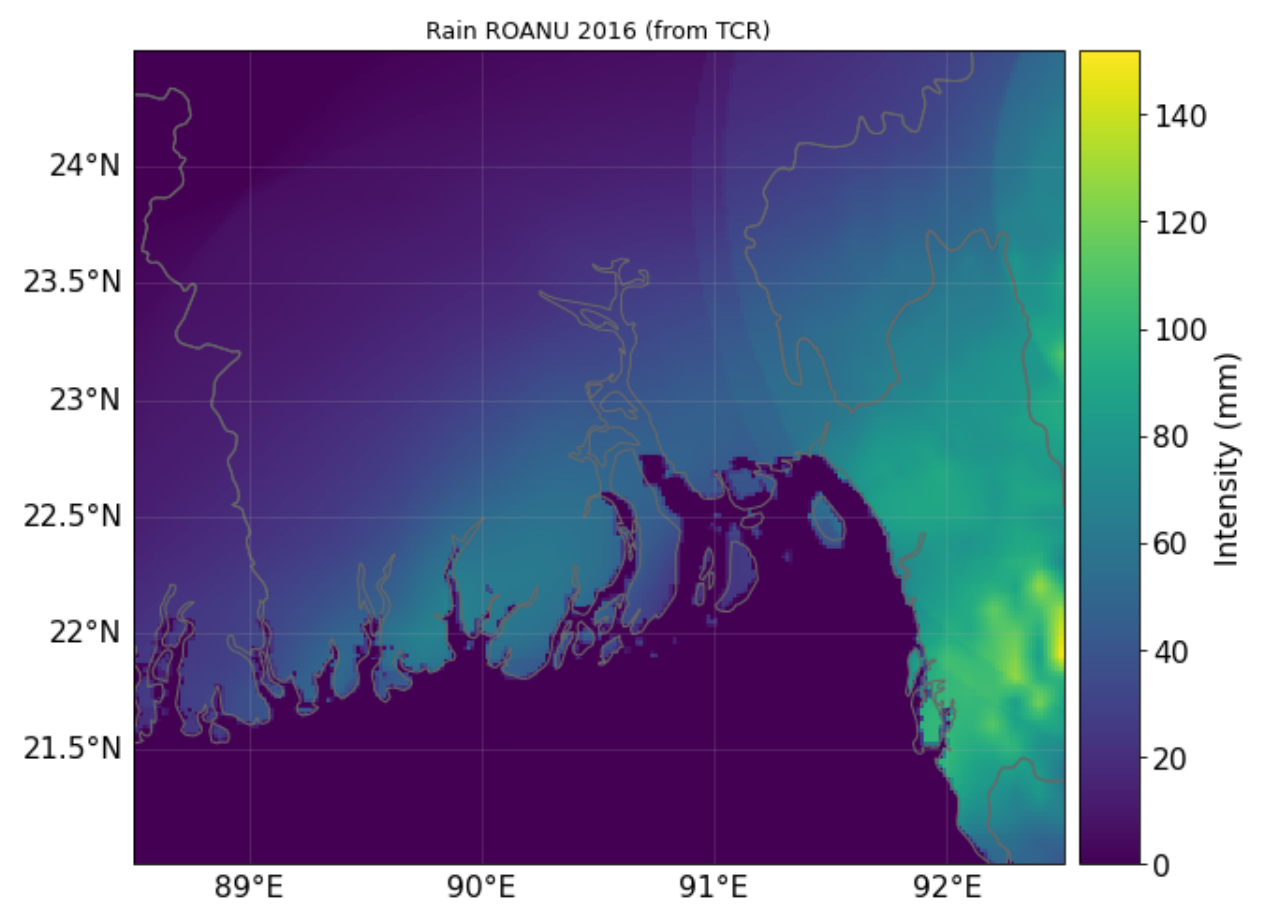
Rain field output with TCR

[5]:
```
tr_bang = TCRain.from_tracks(tracks, centroids=cent_bang, model="TCR")
ax = tr_bang.plot_intensity(1)
ax.set_title('Rain SIDR 2007 (from TCR)')
ax = tr_bang.plot_intensity(2)
ax.set_title('Rain ROANU 2016 (from TCR)');
```

```
2023-07-17 15:15:49,639 - climada_petals.hazard.tc_rainfield - INFO - Mapping 2␣
↪tracks to 45839 coastal centroids.
2023-07-17 15:15:52,918 - climada.util.coordinates - INFO - Sampling from SYSTEM_DIR/
↪c_drag_500.tif
2023-07-17 15:15:54,682 - climada.util.coordinates - INFO - Sampling from SYSTEM_DIR/
↪topography_land_360as.tif
2023-07-17 15:15:55,167 - climada_petals.hazard.tc_rainfield - INFO - Progress: 50%
```

```
2023-07-17 15:16:00,920 - climada.util.coordinates - INFO - Sampling from SYSTEM_DIR/
↪c_drag_500.tif
2023-07-17 15:16:03,552 - climada.util.coordinates - INFO - Sampling from SYSTEM_DIR/
↪topography_land_360as.tif
2023-07-17 15:16:04,354 - climada_petals.hazard.tc_rainfield - INFO - Progress: 100%
```


Rain SIDR 2007 (from TCR)
24°N
23.5°N
23°N
22.5°N
22°N
21.5°N
89°E
90°E
91°E
92°E
175
150
125
100
75
50
25
0
Intensity (mm)



Since TCR has additional requirements on the available along-track variables, the above examples omit or simplify two effects in the model:

- A constant universal estimate is assumed for the specific humidity q_s, which has a large influence on the rain rate as it is directly proportional.
- The shear-induced component of the vertical velocity `w_s` is omitted.

For the two tracks considered in this tutorial, CLIMADA provides a track data set that includes the required along-track variables as extracted from the ERA5 reanalysis:

[6]:
```
from climada.util.api_client import Client
client = Client()
_, [tcrain_examples] = client.download_dataset(client.get_dataset_info(name='tcrain_
↪examples', status='package-data'))

tracks = TCTracks.from_hdf5(tcrain_examples)
tracks.equal_timestep(0.5)

tr_bang = TCRain.from_tracks(tracks, centroids=cent_bang, model="TCR")
ax = tr_bang.plot_intensity(1)
ax.set_title('Rain SIDR 2007 (from TCR, with all required inputs)')
ax = tr_bang.plot_intensity(2)
ax.set_title('Rain ROANU 2016 (from TCR, with all required inputs)');
```

```
2023-07-17 15:16:11,038 - climada.hazard.tc_tracks - INFO - Interpolating 2 tracks to␣
↪0.5h time steps.
```

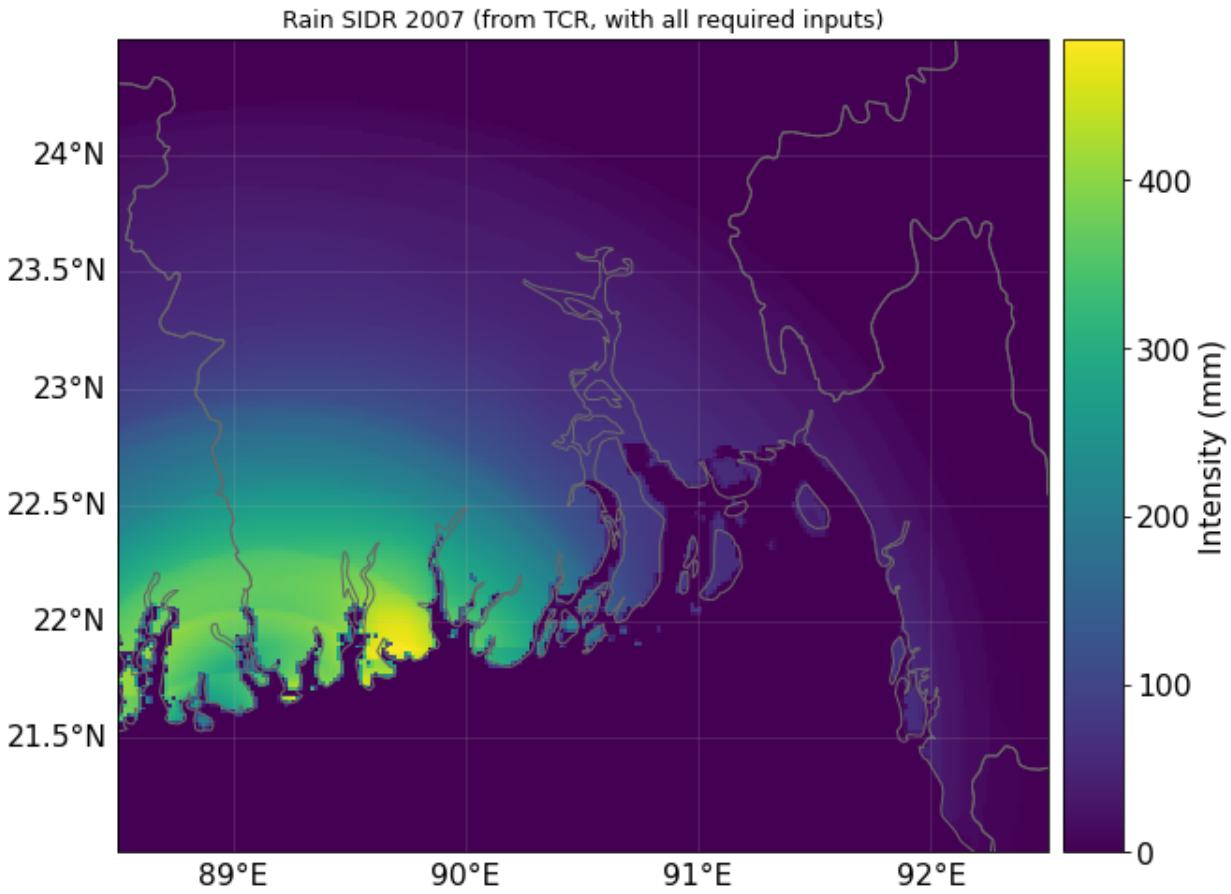
(continues on next page)

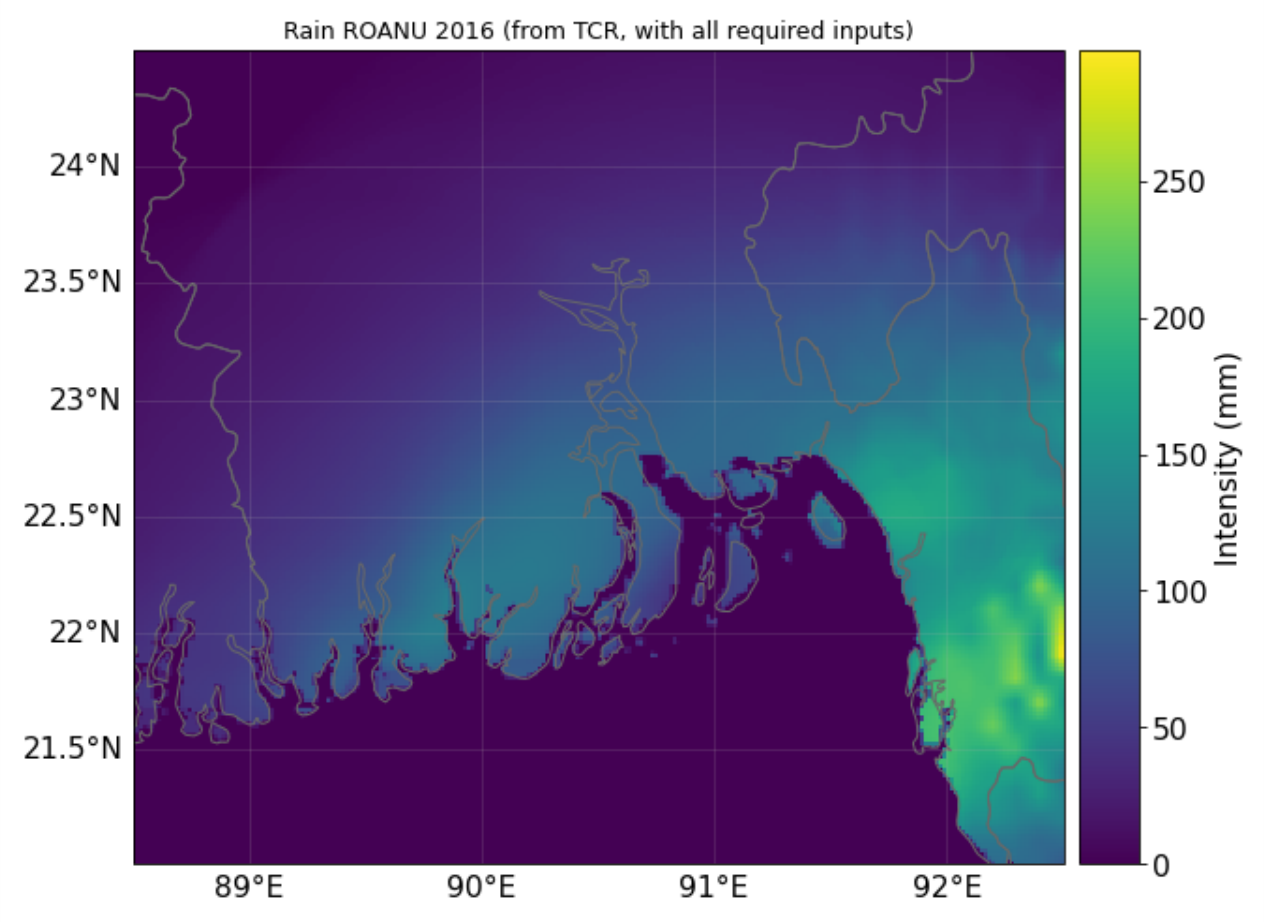
(continued from previous page)

```
2023-07-17 15:16:11,085 - climada_petals.hazard.tc_rainfield - INFO - Mapping 2␣
↪tracks to 45839 coastal centroids.
```

```
2023-07-17 15:16:13,588 - climada.util.coordinates - INFO - Sampling from SYSTEM_DIR/
↪c_drag_500.tif
2023-07-17 15:16:14,437 - climada.util.coordinates - INFO - Sampling from SYSTEM_DIR/
↪topography_land_360as.tif
2023-07-17 15:16:15,162 - climada_petals.hazard.tc_rainfield - INFO - Progress: 50%
```

```
2023-07-17 15:16:20,247 - climada.util.coordinates - INFO - Sampling from SYSTEM_DIR/
↪c_drag_500.tif
2023-07-17 15:16:22,452 - climada.util.coordinates - INFO - Sampling from SYSTEM_DIR/
↪topography_land_360as.tif
2023-07-17 15:16:23,675 - climada_petals.hazard.tc_rainfield - INFO - Progress: 100%
```





2.6 Hazard: Tropical cyclone surge from linear wind-surge relationship and a bathtub model

The `TCSurgeBathtub` class models surges generated by tropical cyclones. Given an elevation data set and a `TropCyclone` instance, it computes the surges for each historical and/or synthetic event at every centroid. `TCSurgeBathtub` inherits from `Hazard` and has an associated hazard type `TCSurgeBathtub`.

2.6.1 Model description

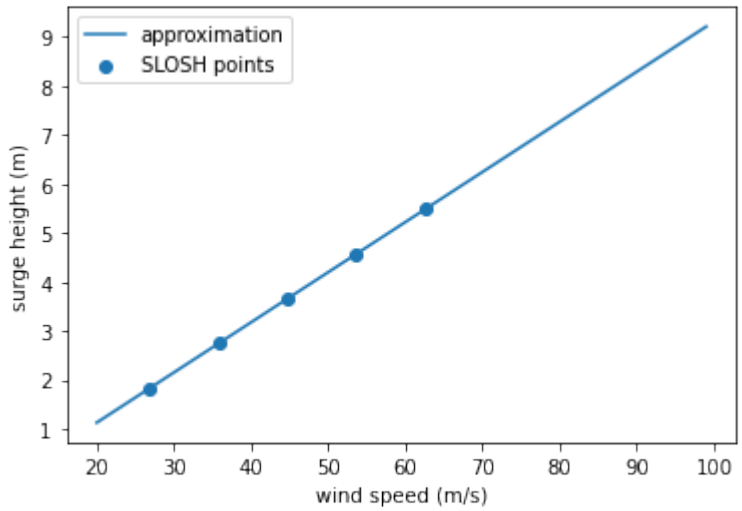
As a first approximation, the tropical cyclone's wind field in each grid cell is used as an input to a simplified version of the **wind-surge relationship** in Xu (2010), which is based on pre-run SLOSH outputs.

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# conversion factors
mph2ms = 0.44704;
f2m = 0.3048;
```

(continues on next page)

(continued from previous page)

```
# the points read from the SLOSH graph
v0 = 60*mph2ms;
v1 = 140*mph2ms;
s0 = 6*f2m;
s1 = 18*f2m;

# the parameters for the linear function: a*(v-v0)+s0
a = (s1-s0)/(v1-v0)

# graphical representation
v = np.arange(20, 100)
vmph = np.arange(60, 141, 20)

plt.plot(v, a*(v-v0)+s0, label='approximation')
plt.scatter(vmph*mph2ms, a*(vmph*mph2ms-v0)+s0, label='SLOSH points')
plt.xlabel('wind speed (m/s)')
plt.ylabel('surge height (m)')
plt.legend()
plt.show()
```



The elevation of the centroids is then subtracted from the surge using the user-specified elevation data set. The elevation data set has to be given as a path to a GeoTIFF grid data file that covers the region affected by the tropical cyclone. A global data set is freely available as SRTM15+V2.0 (a sample of which is used in the example below). The improved-quality CoastalDEM data set is available on request from Climate Central.

In a final step, a decay of the surge height depending on the distance from the coastline by 0.2 meters per kilometer is implemented following Pielke and Pielke (1997).

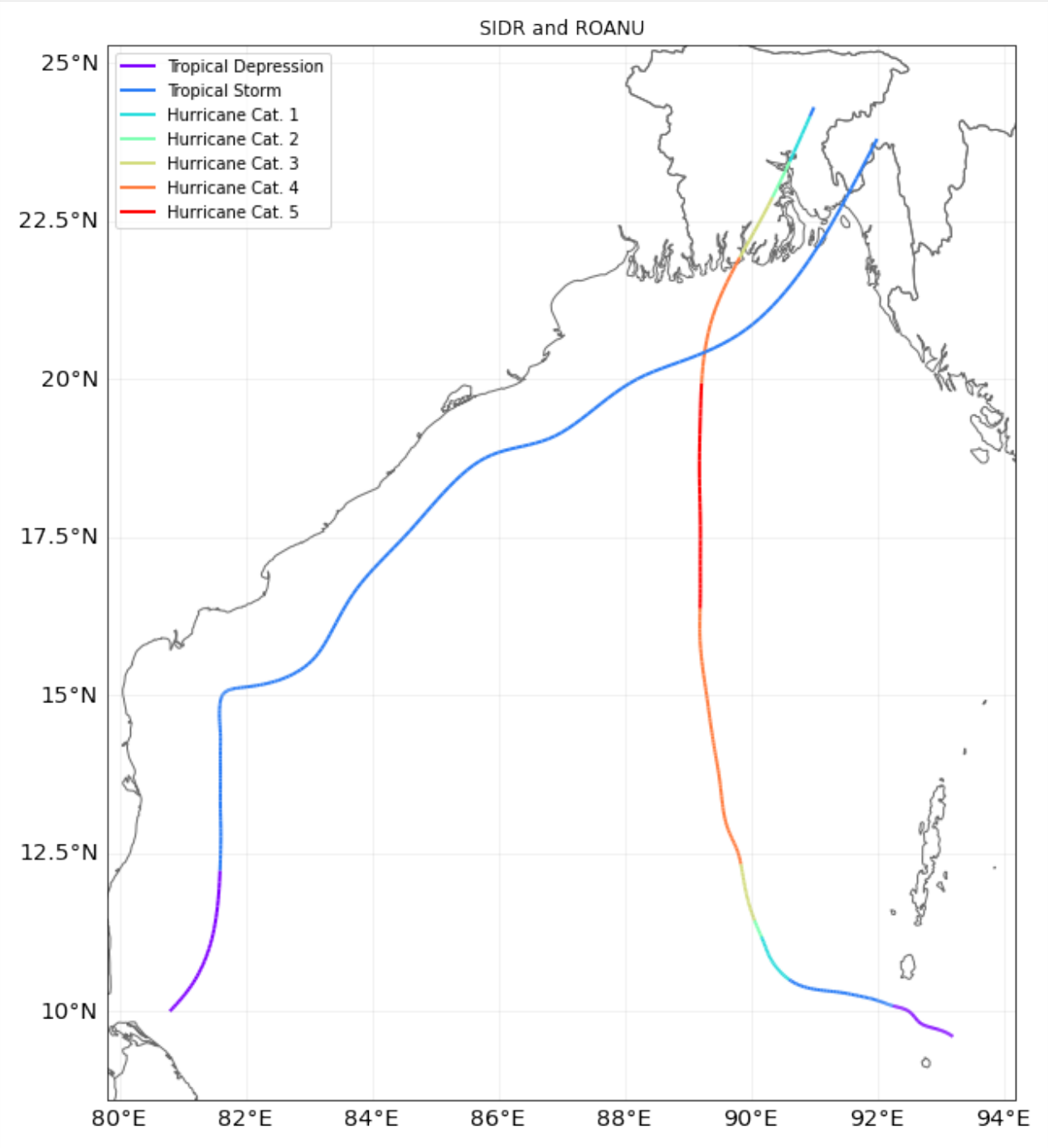
Optionally, a user-specified sea level rise offset is added to the result.

2.6.2 Example

We compute the surges of Sidr 2007 and Roanu 2016 over Bangladesh as follows:

[2]:
```
%matplotlib inline
# 1: tracks retrieval
from climada.hazard import TCTracks

tr_usa = TCTracks.from_ibtracs_netcdf(provider='usa', storm_id=['2007314N10093',
↪'2016138N10081']) # SIDR 2007 and ROANU 2016
tr_usa.equal_timestep(0.5)
ax = tr_usa.plot()
ax.get_legend()._loc = 2 # correct legend location
ax.set_title('SIDR and ROANU'); # set title
```

```
2021-07-09 15:24:31,990 - climada.hazard.tc_tracks - INFO - Progress: 50%
2021-07-09 15:24:32,004 - climada.hazard.tc_tracks - INFO - Progress: 100%
2021-07-09 15:24:32,015 - climada.hazard.tc_tracks - INFO - Interpolating 2 tracks to␣
↪0.5h time steps.
```



[3]:
```
# 2: wind gusts computation
from climada.hazard import TropCyclone, Centroids

# define centroids raster
min_lat, max_lat, min_lon, max_lon = 20, 27, 88.5, 92.5
cent_bang = Centroids.from_pnt_bounds((min_lon, min_lat, max_lon, max_lat), res=0.015)
cent_bang.set_dist_coast(signed=True, precomputed=True)
cent_bang.check()

tc_bang = TropCyclone.from_tracks(tr_usa, centroids=cent_bang)
```

```
2021-07-09 15:24:33,987 - climada.util.coordinates - INFO - Sampling from /home/
↪tovogt/.climada/data/GMT_intermediate_coast_distance_01d.tif
2021-07-09 15:24:34,077 - climada.hazard.trop_cyclone - INFO - Mapping 2 tracks to␣
↪125424 coastal centroids.
2021-07-09 15:24:41,455 - climada.hazard.trop_cyclone - INFO - Progress: 100%
```

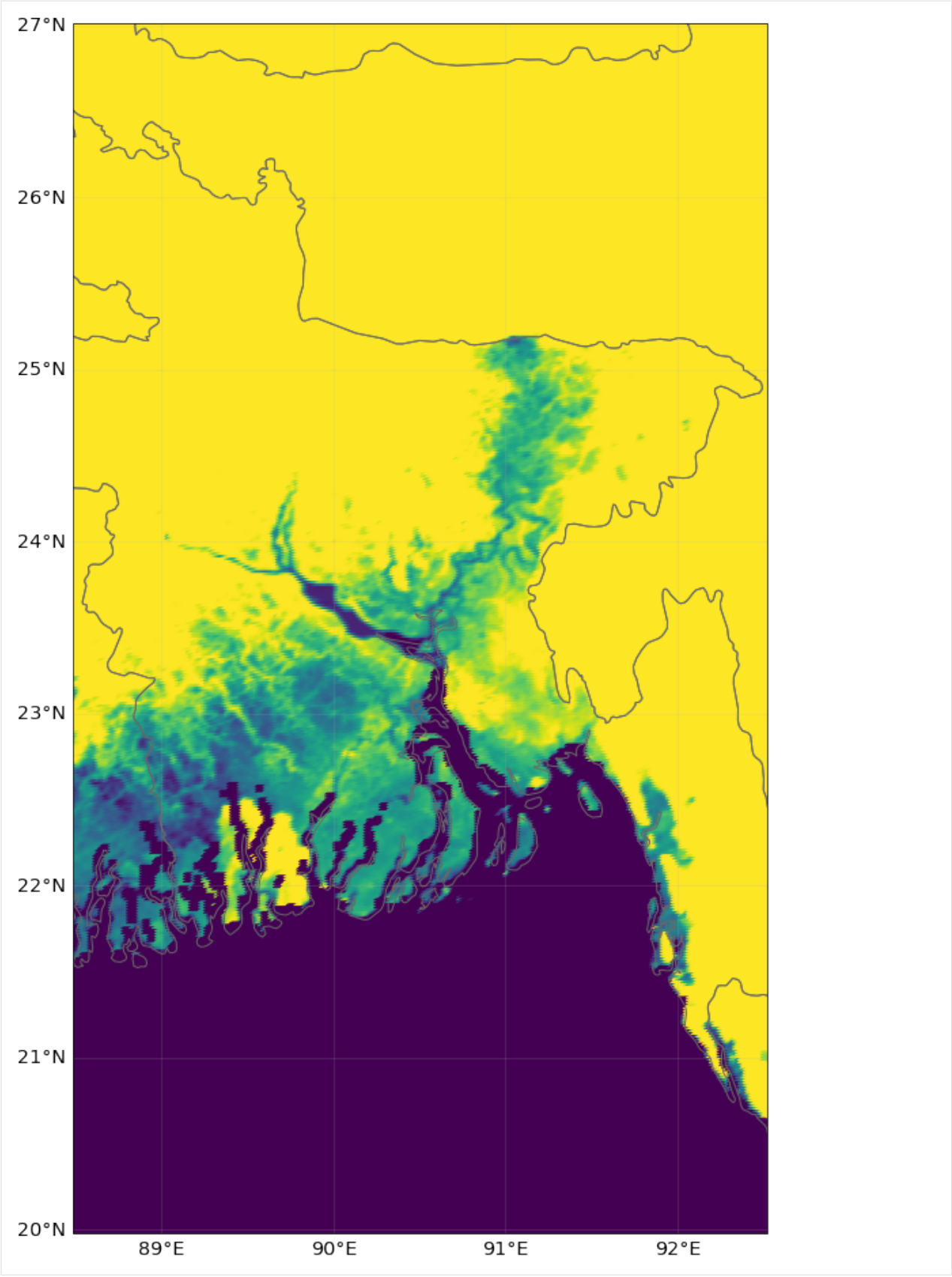
[4]:
```
# 3: surge computation
from climada_petals.hazard import TCSurgeBathtub
from climada.util.constants import DEMO_DIR

# If you have the global SRTM15+V2.0 elevation data set, you can replace the following
# sample DEM data set by your SRTM15+V2.0.tiff:
topo_path = DEMO_DIR.joinpath('SRTM15+V2.0_sample.tiff')
ts_bang = TCSurgeBathtub.from_tc_winds(tc_bang, topo_path)
```

```
2021-07-09 15:24:41,466 - climada.util.coordinates - INFO - Sampling from /home/
↪tovogt/.climada/demo/data/SRTM15+V2.0_sample.tiff
```

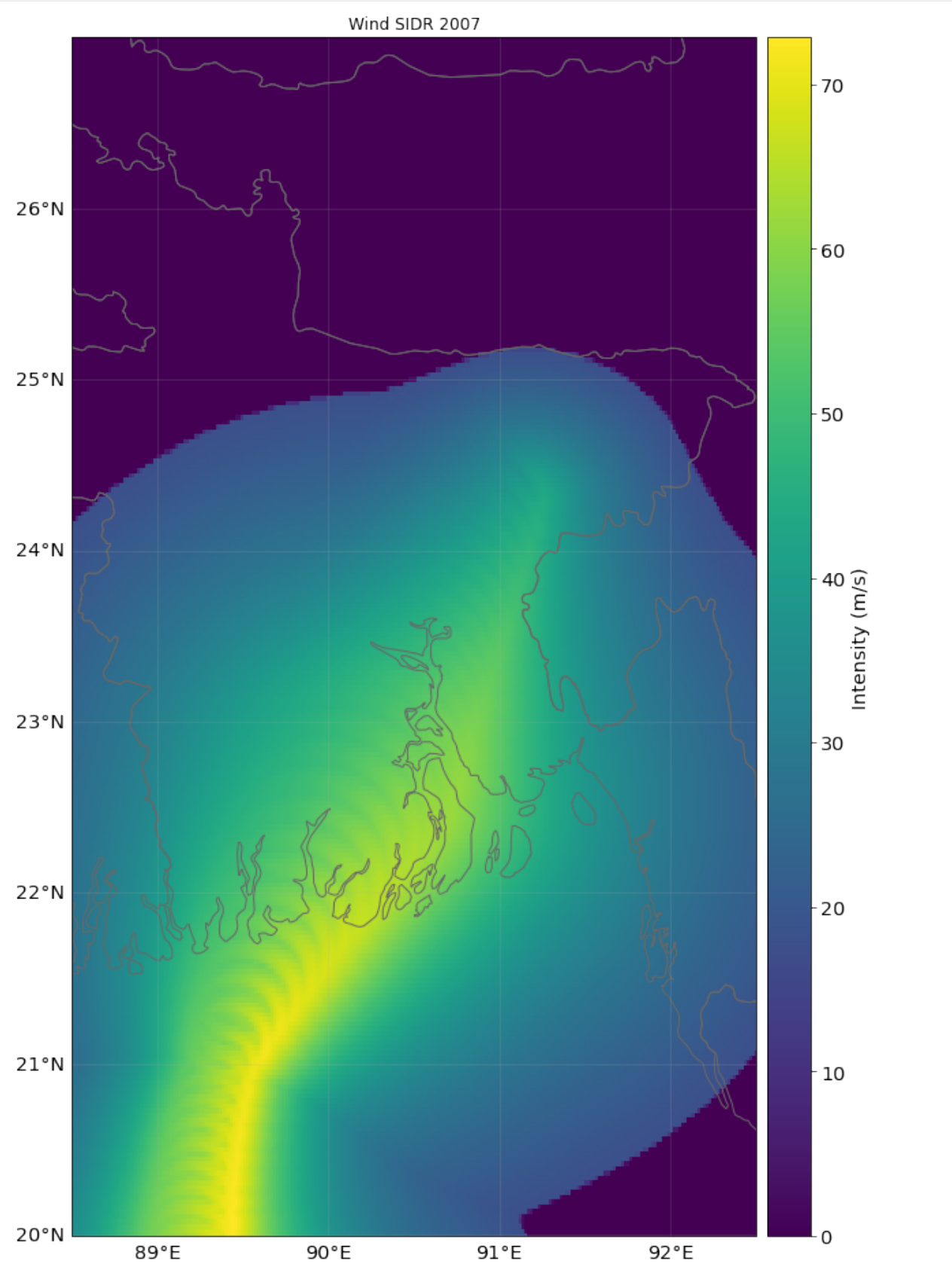
[5]:
```
# plot elevation of the raster
ts_bang.centroids.set_elevation(topo_path)
ts_bang.centroids.plot(c=ts_bang.centroids.elevation, vmin=0, vmax=10)
```

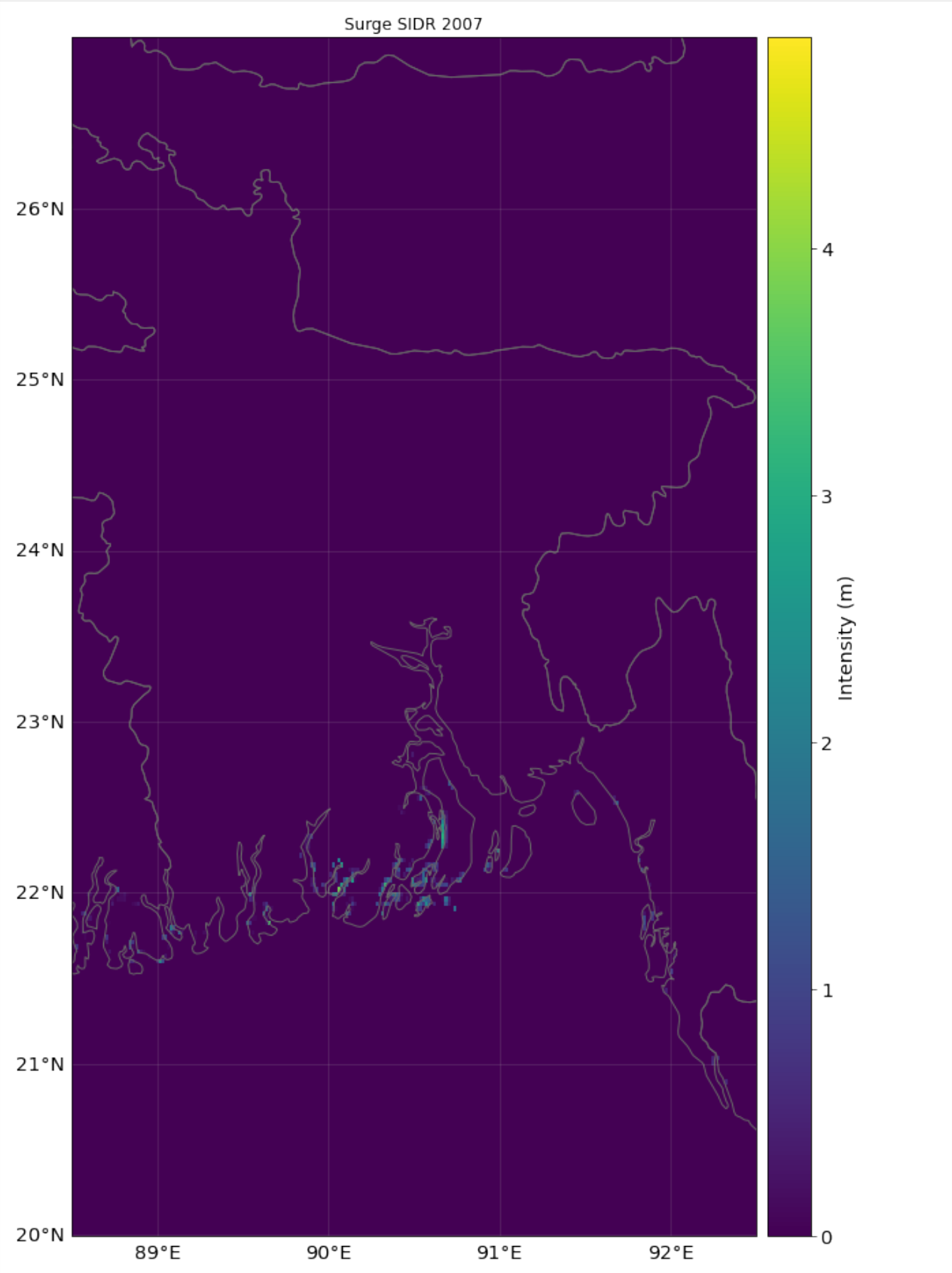
```
2021-07-09 15:24:41,544 - climada.util.coordinates - INFO - Sampling from /home/
↪tovogt/.climada/demo/data/SRTM15+V2.0_sample.tiff
```

[5]:
```
<GeoAxesSubplot:>
```

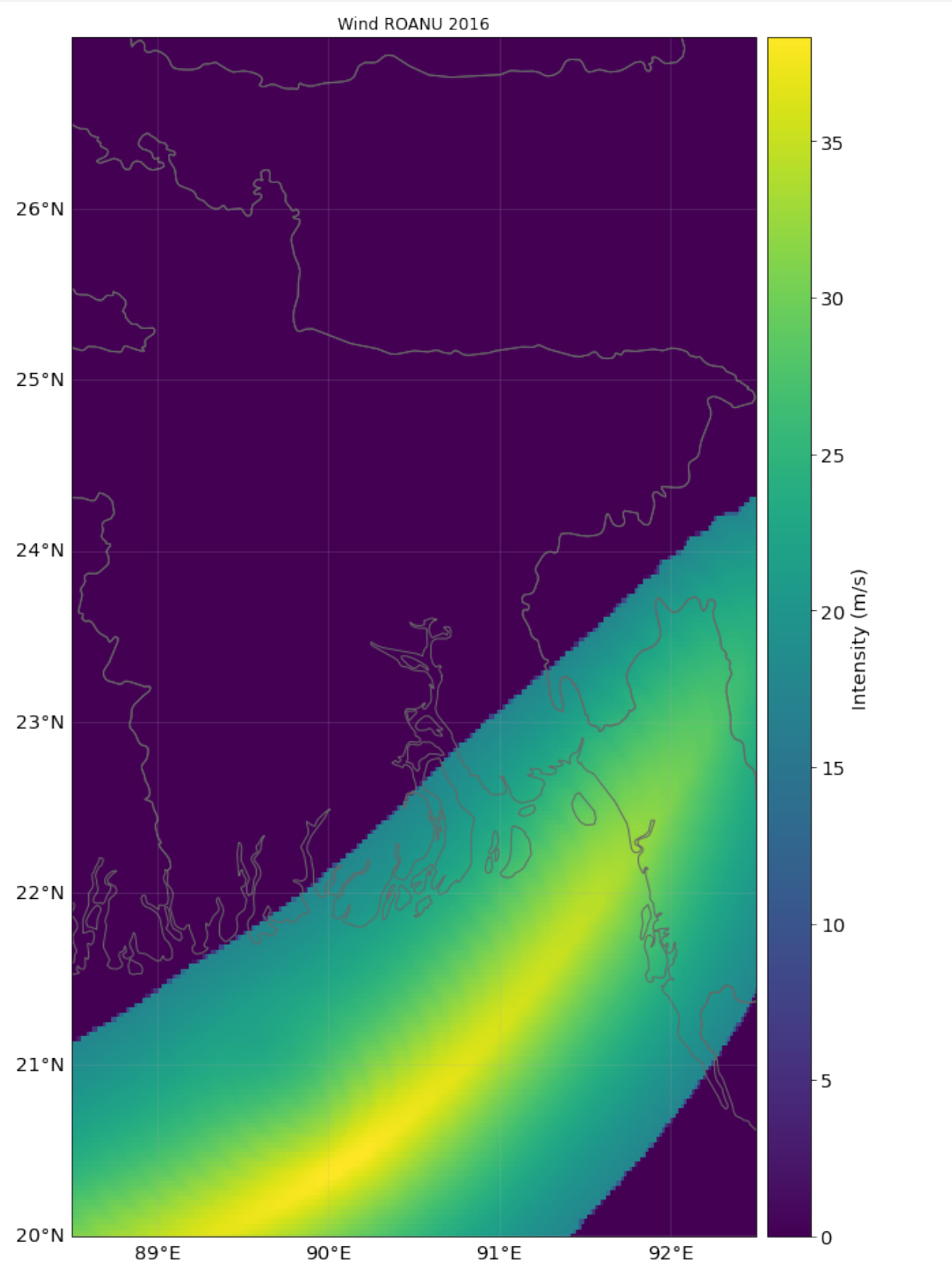

27°N
26°N
25°N
24°N
23°N
22°N
21°N
20°N
89°E
90°E
91°E
92°E

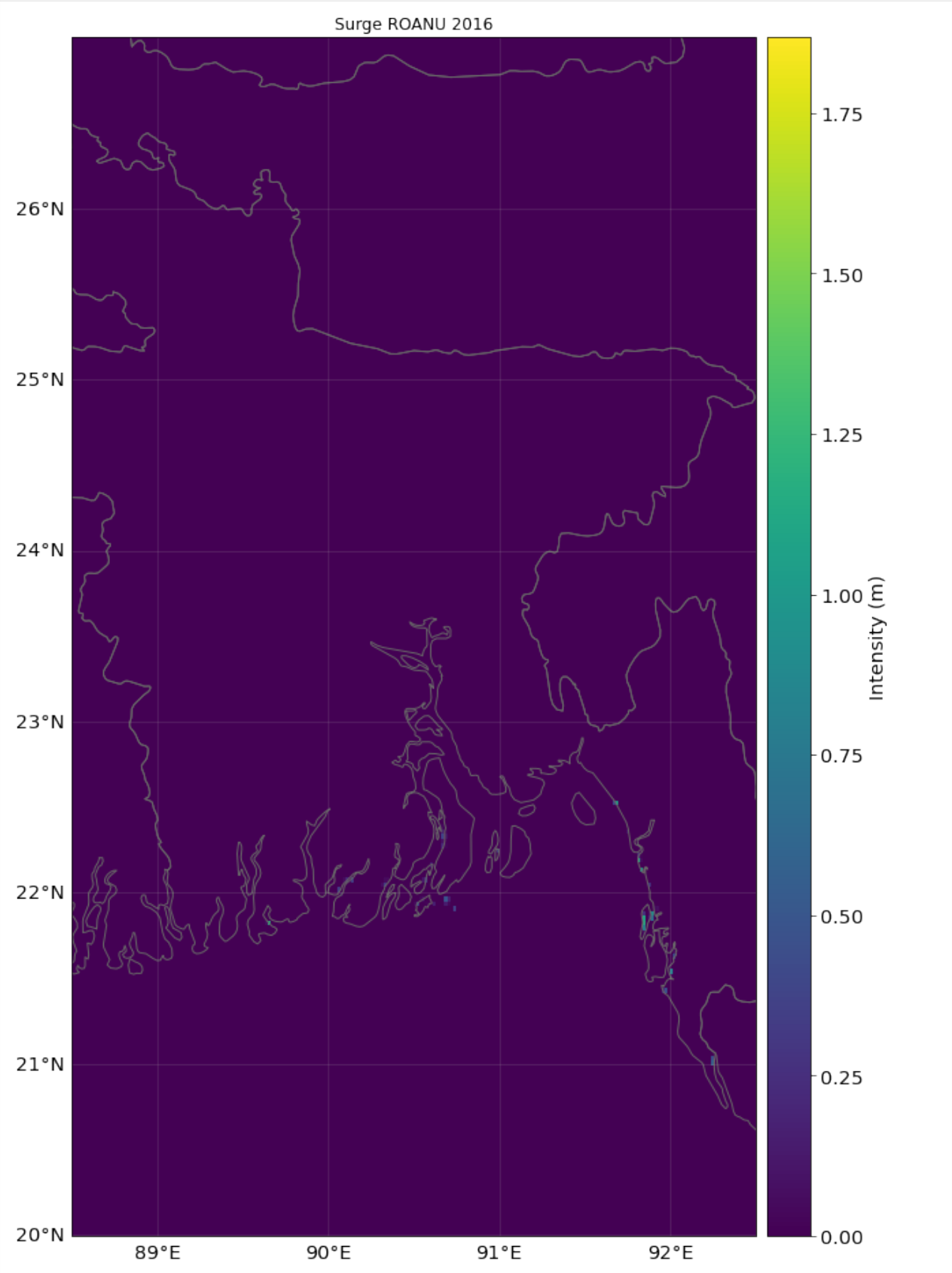
[6]:
```
# plot wind and surge SIDR
ax = tc_bang.plot_intensity(1)
ax.set_title('Wind SIDR 2007')
ax = ts_bang.plot_intensity(1)
ax.set_title('Surge SIDR 2007');
```


Wind SIDR 2007
26°N
25°N
24°N
23°N
22°N
21°N
20°N
89°E
90°E
91°E
92°E
70
60
50
40
30
20
10
0
Intensity (m/s)


Surge SIDR 2007
26°N
25°N
24°N
23°N
22°N
21°N
20°N
89°E
90°E
91°E
92°E
Intensity (m)
0
1
2
3
4

[7]:
```
# plot wind and surge ROANU
ax = tc_bang.plot_intensity(2)
ax.set_title('Wind ROANU 2016')
ax = ts_bang.plot_intensity(2)
ax.set_title('Surge ROANU 2016');
```


Wind ROANU 2016
26°N
25°N
24°N
23°N
22°N
21°N
20°N
89°E
90°E
91°E
92°E
35
30
25
20
15
10
5
0
Intensity (m/s)


Surge ROANU 2016
26°N
25°N
24°N
23°N
22°N
21°N
20°N
89°E
90°E
91°E
92°E
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Intensity (m)

2.7 Hazard: WildFire

This class is used to model the wildfire hazard using the historical data available and creating synthetic fires which are summarized into event years to establish a comprehensive probabilistic risk assessment.

The historical data used comes from the Fire Information for Resource Management System (FIRMS). They collect temperatures from the following satellite instruments:

```
- Moderate Resolution Imaging Spectroradiometer  (MODIS): Near real time or standard␣
↪quality data with 1 km resolution. Data available from November 2000 to present.

- Visible Infrared Imaging Radiometer Suite (VIIRS): Near real time data with 0.375␣
↪km resolution. Data available from 20 January 2012 to present.
```

The data should be obtained at https://firms.modaps.eosdis.nasa.gov/download/ and saved as .csv file. Approximately 15 min after submitting the request, the data can be downloaded by checking the request status.

The `WildFire` class inherits from the `Hazard` class and has an associated hazard type `WF`. It provides a `set_hist_fire_FIRMS()` method which enables to fill the hazard with historical events, a `set_hist_fire_seasons_FIRMS()` method which aggregates historic events to event years, and a `set_proba_fire_seasons()` method which generates random event years.

Some additional notes: - in contrast to other hazards (i.e. TC) there's no clear definition for "a wildfire event". While some countries (i.e. US, Australia) provide very detailed reports, other's (i.e. Chile, Indonesia) rather provide information on the whole fire season. Thus, we produce probabilistic fire seasons rather than single events to ensure global consistency. However, for case studies or calibration studies, identification of single fire events is still of importance. Thus, we differentiate in the hazard type: 'WFsingle' for individual fires as recognized by the algorithm, 'WFseason' for whole fire seasons.

- Wildfires are hazards with a huge societal component - in many cases humans are not only responsible for starting one, but also for ending it (fire fighting). This means, that worst case scenarios (i.e. a wild fire spreading to down town Los Angeles) are less likely, as compared to TC risk, where it is highly likely that a strong TC strikes Manila and people have no chance to prevent that from happening. Accordingly, the probabilistic wildfires come with an even larger uncertainty than other hazard as social behaviour have a major impact on wildfires.
- In our default version for creating probabilistic fire seasons, we rely fully on past data. Thus, in the default, this module is not suited to conduct studies on climate change risk. However, the generation of synthetic fire seasons can be manipulated.
- We are currently working on improving the generation of probabilistic events.

2.7.1 This tutorial describes

- How past fires are computed
- How past fire seasons are computed
- How probabilistic fire seasons are generated

HISTORICAL FIRES

A fire is defined as an event when the temporal and spatial distance of the burning centroids is close enough. There are two parameters which define the "closeness":

```
- days_thres (int): temporal distance in days. Default: 2.
- clus_thres (int): factor to multiply to the centroids resolution. Used to determine␣
↪the cluster maximum distance between two centroids. Default: 15.
```

These parameters are stored in the DataClass `WF.FirmsParams`

In June 2017, Portugal was hit by a series of deadly wild fires which led to 66 fatalities, more than 200 mn. USD damages and 45'000 ha burned area. With `set_hist_fire_FIRMS()` we can model past fire such as the ones in Portugal in 2017. The method takes FIRMS data as a dataframe (`pd.df`) as input. If no `Centroids` are defined, the method automatically defines centroids in line with the spatial extent of the FIRMS data. Hazard resolution can be modified using parameter `centr_res_factor`.

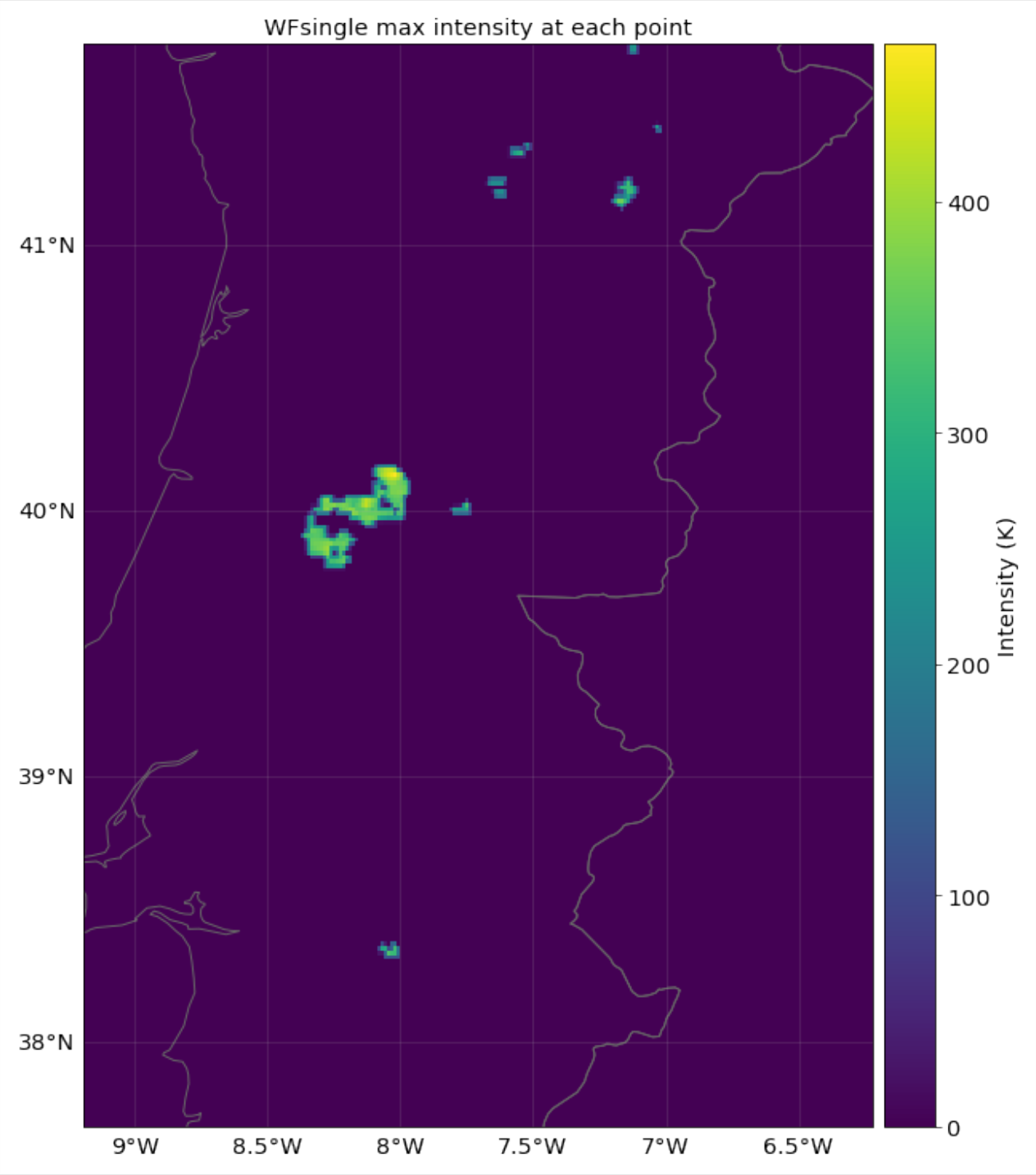
[3]:
```
# Fires in Portugal 2017
import os
import pandas as pd

from climada.util.constants import DEMO_DIR
from climada_petals.hazard import WildFire

# Data downloaded for MODIS
d_path = os.path.join(DEMO_DIR, "Portugal_firms_June_2017.csv")

# read data
firms = pd.read_csv(d_path) # FIRMS data as pandas dataframe

# set up wildfire
wf_pt = WildFire()
wf_pt.set_hist_fire_FIRMS(firms, centr_res_factor=1./2.5) # we decrease the hazard␣
↪resolution to 2.5 km

# plot the maximum intensity of all fires at each centroid
wf_pt.plot_intensity(event=0);
```

```
2022-05-03 22:46:05,849 - climada_petals.hazard.wildfire - WARNING - The use of␣
↪WildFire.set_hist_fire_FIRMS is deprecated.Use WildFire.from_hist_fire_FIRMS .
2022-05-03 22:46:06,120 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2022-05-03 22:46:08,902 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 0.
2022-05-03 22:46:08,946 - climada_petals.hazard.wildfire - INFO - Computing intensity␣
↪of 7 fires.
2022-05-03 22:46:10,539 - climada_petals.hazard.wildfire - INFO - Returning 7 fires␣
↪that impacted the defined centroids.
```

```
/home/yuyue/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↪1282: UserWarning: You will likely lose important projection information when␣
↪converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  proj = self._crs.to_proj4(version=version)
```

[3]:
```
<GeoAxesSubplot:title={'center':'WFsingle max intensity at each point'}>
```

```
<Figure size 432x288 with 0 Axes>
```



We can focus on the region around Coimbra where the most deadly fire occurred.

A given set of points or grid can be provided using the `Centroids` class. The events will be interpolated to the provided coordinates.

[4]:
```
from climada.hazard import Centroids
from climada.util.constants import ONE_LAT_KM
```

(continues on next page)

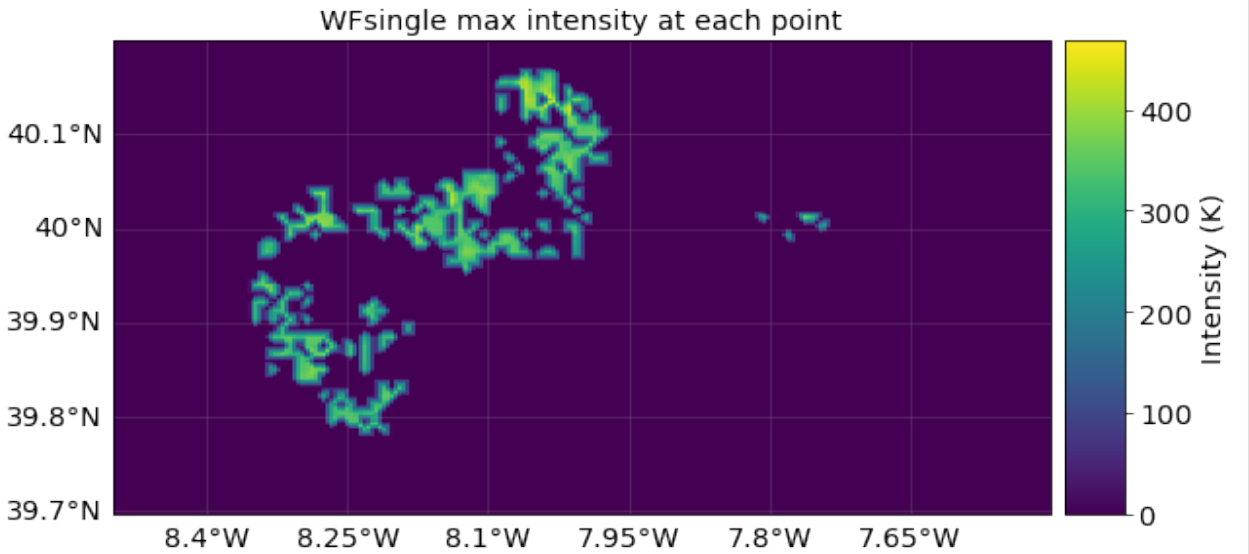
(continued from previous page)

```
# focus on Coimbra region with 1.0 km resolution (MODIS resolution)
res = 1.0/ONE_LAT_KM
centr_zoom = Centroids.from_pnt_bounds((-8.5, 39.7, -7.5, 40.2), res)

wf_zoom = WildFire()
wf_zoom.set_hist_fire_FIRMS(firms, centroids=centr_zoom)
wf_zoom.plot_intensity(0);
```

```
2022-05-03 22:48:33,103 - climada_petals.hazard.wildfire - WARNING - The use of␣
↪WildFire.set_hist_fire_FIRMS is deprecated.Use WildFire.from_hist_fire_FIRMS .
2022-05-03 22:48:33,284 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 0.
2022-05-03 22:48:33,320 - climada_petals.hazard.wildfire - INFO - Computing intensity␣
↪of 7 fires.
2022-05-03 22:48:33,369 - climada_petals.hazard.wildfire - INFO - Returning 2 fires␣
↪that impacted the defined centroids.
```

```
/home/yuyue/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↪1282: UserWarning: You will likely lose important projection information when␣
↪converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  proj = self._crs.to_proj4(version=version)
```

[4]:
```
<GeoAxesSubplot:title={'center':'WFsingle max intensity at each point'}>
```



See more infos on Wikipedia around the fire.

HISTORICAL FIRE SEASONS

`set_hist_fire_seasons_FIRMS()` generates historic fire seasons.

These fire seasons can later be compared to the probabilistic event years for a seamless risk analysis. Note: - all the underlying events of a year are merged into one single event which is saved. - The original events, that result from the clustering are not kept in order to save space. However, this can be done setting the parameter `keep_all_events` to true. - If single fires are already calculated and assigned to centroids, they can be aggregated to seasons using `summarize_fires_to_seasons()`

[5]:
```
# load FIRMS data for 2016-2018 an select main land Portugal only
firms_seasons = pd.read_csv(os.path.join(DEMO_DIR, "Portugal_firms_2016_17_18_MODIS.
↪csv"))
firms_seasons = firms_seasons[firms_seasons['latitude']>35.]
firms_seasons = firms_seasons[firms_seasons['longitude']>-12.]


wf_years = WildFire()
wf_years.set_hist_fire_seasons_FIRMS(firms_seasons, centr_res_factor=1/1.) # we use␣
↪MODIS data resolution (1 km)

print('Events are now named according to their event year:', wf_years.event_name)
print('The number of underyling events is saved as well:', wf_years.n_fires)

# plot the three fire seasons (2016, 2017, 2018)
wf_years.plot_intensity(1)
wf_years.plot_intensity(2)
wf_years.plot_intensity(3);
```

```
2022-05-03 22:52:26,223 - climada_petals.hazard.wildfire - WARNING - The use of␣
↪WildFire.set_hist_fire_seasons_FIRMS is deprecated.Use WildFire.from_hist_fire_
↪seasons_FIRMS .
2022-05-03 22:52:26,226 - climada_petals.hazard.wildfire - INFO - Setting up␣
↪historical fires for year set.
2022-05-03 22:52:27,216 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2022-05-03 22:53:01,125 - climada_petals.hazard.wildfire - INFO - Setting up␣
↪historical fire seasons 2016.
2022-05-03 22:53:04,088 - climada_petals.hazard.wildfire - WARNING - The use of␣
↪WildFire.set_hist_fire_FIRMS is deprecated.Use WildFire.from_hist_fire_FIRMS .
2022-05-03 22:53:05,080 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 3388.
2022-05-03 22:53:06,153 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 11.
2022-05-03 22:53:06,334 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 0.
2022-05-03 22:53:07,808 - climada_petals.hazard.wildfire - INFO - Computing intensity␣
↪of 113 fires.
2022-05-03 22:53:09,394 - climada_petals.hazard.wildfire - INFO - Returning 113 fires␣
↪that impacted the defined centroids.
2022-05-03 22:53:11,622 - climada_petals.hazard.wildfire - INFO - Setting up␣
↪historical fire seasons 2017.
2022-05-03 22:53:14,398 - climada_petals.hazard.wildfire - WARNING - The use of␣
↪WildFire.set_hist_fire_FIRMS is deprecated.Use WildFire.from_hist_fire_FIRMS .
2022-05-03 22:53:15,494 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 2392.
2022-05-03 22:53:17,040 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 210.
```

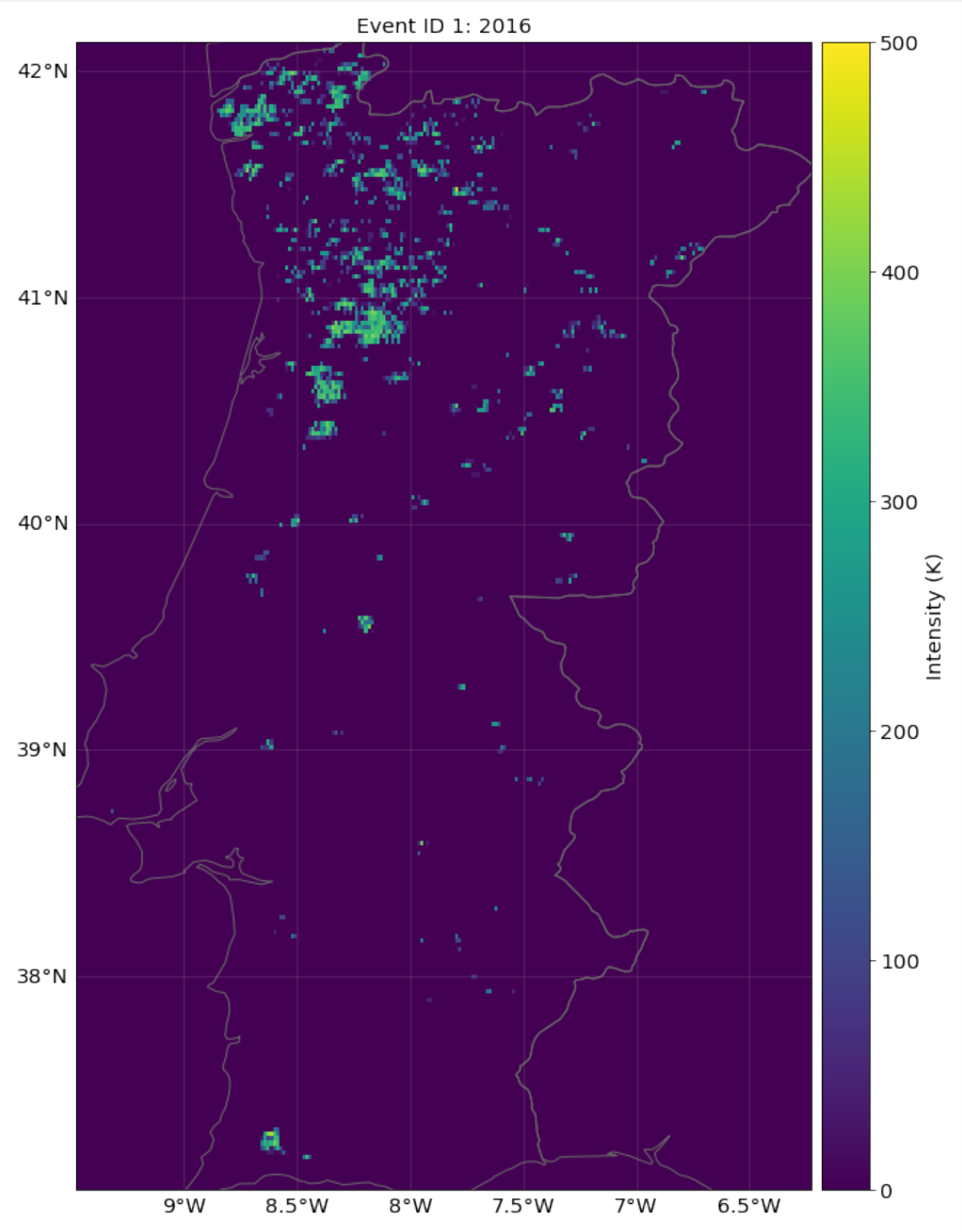
(continues on next page)

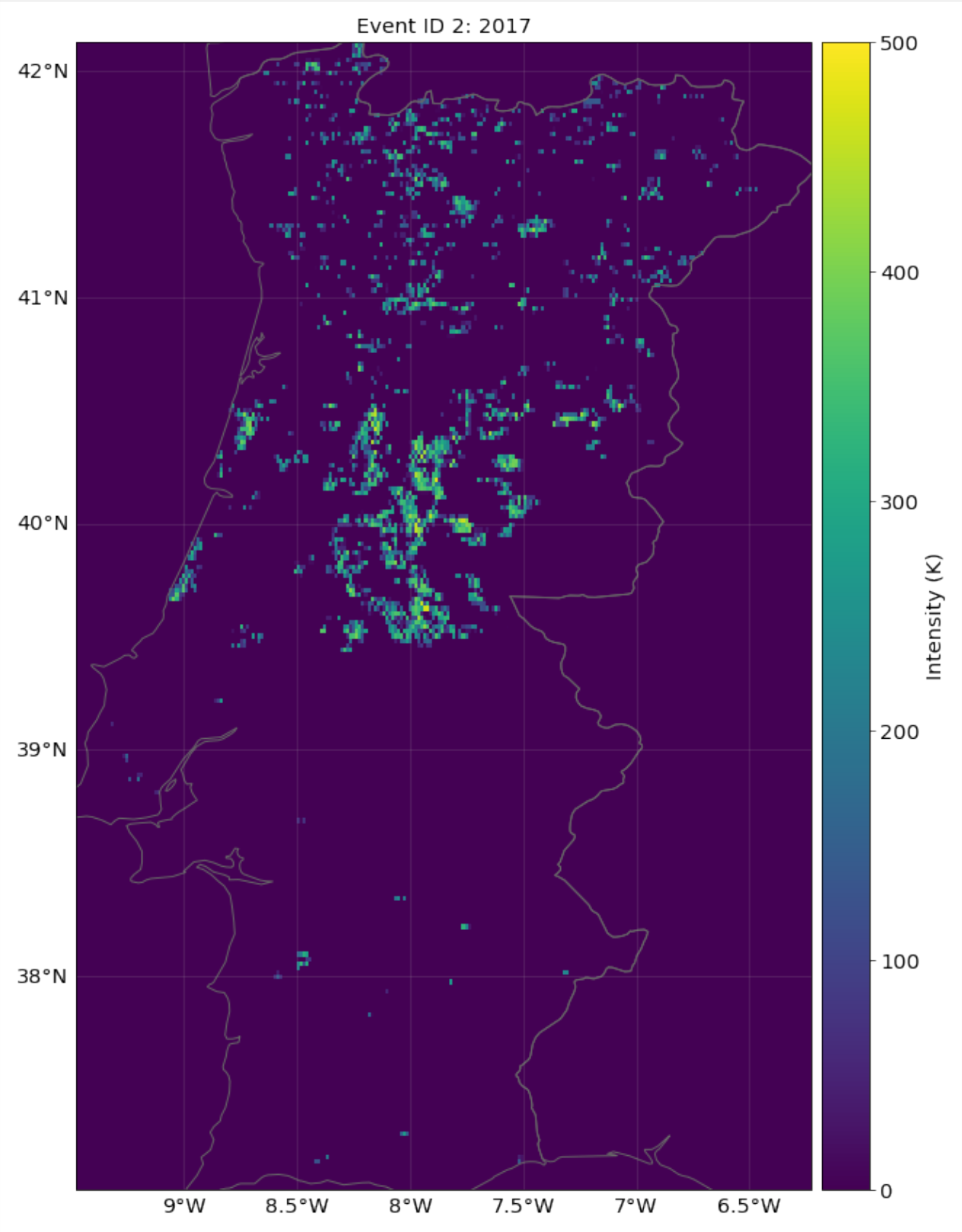
(continued from previous page)

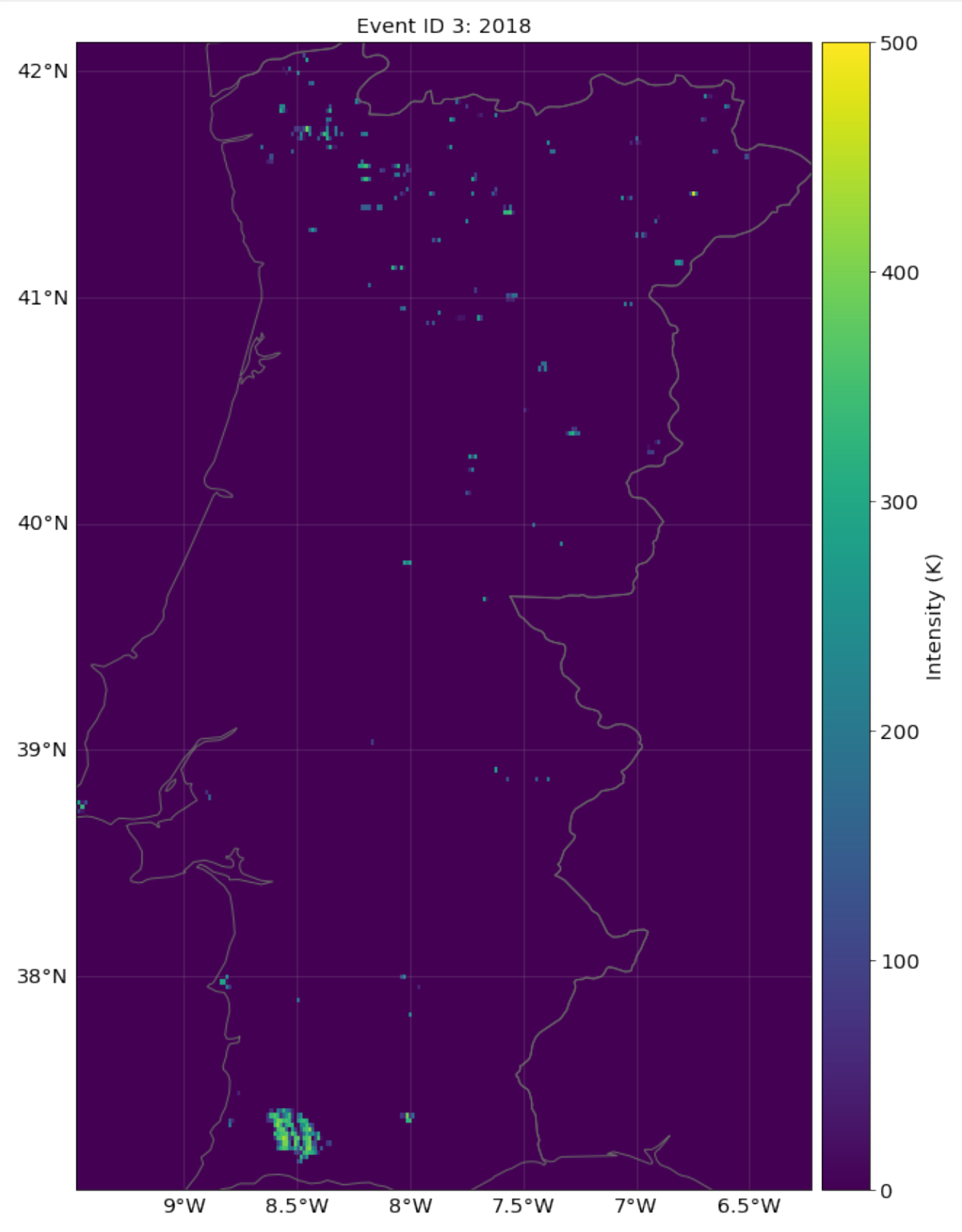
```
2022-05-03 22:53:17,502 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 0.
2022-05-03 22:53:20,061 - climada_petals.hazard.wildfire - INFO - Computing intensity␣
↪of 201 fires.
2022-05-03 22:53:22,711 - climada_petals.hazard.wildfire - INFO - Returning 201 fires␣
↪that impacted the defined centroids.
2022-05-03 22:53:27,146 - climada_petals.hazard.wildfire - INFO - Setting up␣
↪historical fire seasons 2018.
2022-05-03 22:53:30,230 - climada_petals.hazard.wildfire - WARNING - The use of␣
↪WildFire.set_hist_fire_FIRMS is deprecated.Use WildFire.from_hist_fire_FIRMS .
2022-05-03 22:53:31,113 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 169.
2022-05-03 22:53:31,666 - climada_petals.hazard.wildfire - INFO - Remaining fires to␣
↪identify: 0.
2022-05-03 22:53:32,795 - climada_petals.hazard.wildfire - INFO - Computing intensity␣
↪of 72 fires.
2022-05-03 22:53:34,014 - climada_petals.hazard.wildfire - INFO - Returning 72 fires␣
↪that impacted the defined centroids.
Events are now named according to their event year: ['2016', '2017', '2018']
The number of underyling events is saved as well: [113. 201.  72.]
```

```
/home/yuyue/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↪1282: UserWarning: You will likely lose important projection information when␣
↪converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  proj = self._crs.to_proj4(version=version)
/home/yuyue/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↪1282: UserWarning: You will likely lose important projection information when␣
↪converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  proj = self._crs.to_proj4(version=version)
/home/yuyue/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↪1282: UserWarning: You will likely lose important projection information when␣
↪converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  proj = self._crs.to_proj4(version=version)
```

```
[5]: <GeoAxesSubplot:title={'center':'Event ID 3: 2018'}>
```


Event ID 1: 2016
42°N
41°N
40°N
39°N
38°N
9°W
8.5°W
8°W
7.5°W
7°W
6.5°W
500
400
300
200
100
0
Intensity (K)


Event ID 2: 2017
42°N
41°N
40°N
39°N
38°N
9°W
8.5°W
8°W
7.5°W
7°W
6.5°W
500
400
300
200
100
0
Intensity (K)


Event ID 3: 2018
42°N
41°N
40°N
39°N
38°N
9°W
8.5°W
8°W
7.5°W
7°W
6.5°W
500
400
300
200
100
0
Intensity (K)

PROBABILISTIC FIRE SEASONS

`set_proba_fire_seasons()` generates synthetic fire seasons.

A probabilistic fire season is generate the following way:

1. A fire propagation probability matrix is defined - this propagation matrix defines the probability for fire propagation on each centroid - if not further specified, the observation of historic data is used -> the fire can only propagate on centroids that burned in the past, including a blurr around these centroids - this matrix can be used to include additional information (i.e. on vegetation/land-use/elevation/population density) - the fire propagation probability matrix is stored in `wf.centroids.fire_propa_matrix` as an np.array and thus can be adjusted easily
2. A random number n of fires that occurred in the probabilistic year is chosen. - If nothing else is specified, a random number within the range of n_fires (the number of historic events is drawn) - The range can also be specified in the input via the parameter `n_ignition`
3. n synthetic fires are generated following this logic: - A random fire ignition point is selected (the point needs to be burnable on the fire propagation matrix) - The fire is then propagated from the ignition point as a cellular automat following these rules: 1. a neighbouring centroid to the burning centroid becomes a burning centroid with a probability p. This probability is calculated as the product of a overall fire propagation probability (default=0.21) and the centroid specific factor (which is defined on the fire propagation probability matrix) 2. an already burning centroid becomes an ember centroid (does not propagate fire anymore but is labeled as burned) 3. an ember centroid stays an ember centroid 4. The propagation stops when there is no burning centroid left. 5. The temperature provided to each centroid is randomly chosen from the corresponding historical event.
4. These n synthetic events are summarized into a probabilistic fire season which is then appended as a new event to the historic fire seasons

This process can be computationally expensive. Thus working on clusters might be advantageous.

Note: In the default no additional external data is included. The probabilistic event thus solely relies on past experience which likely leads to an underestimation of the current and future fire risk. The frame work allows to include climate change effects by: - defining the fire propagation probability matrix (using more data as described above) - varying the overall fire propagation probability (by i.e. linking it to a fire weather index) - varying the number of events per year (by i.e. linking it to a fire weather index)

[6]:
```
# generate 1 probabilistic fire season for Portugal
wf_years.set_proba_fire_seasons(n_fire_seasons=1)
print('The probabilistic event year is appended to the historic:', wf_years.event_
↪name)

# Plot the synthetic event year
wf_years.plot_intensity(event=4);
```

```
2022-05-03 23:01:35,887 - climada_petals.hazard.wildfire - INFO - Setting up␣
↪probabilistic fire season with 149 fires.
2022-05-03 23:01:36,787 - climada_petals.hazard.wildfire - INFO - Created 0 fires
2022-05-03 23:01:41,790 - climada_petals.hazard.wildfire - INFO - Created 10 fires
2022-05-03 23:01:42,272 - climada_petals.hazard.wildfire - INFO - Created 20 fires
2022-05-03 23:01:42,745 - climada_petals.hazard.wildfire - INFO - Created 30 fires
2022-05-03 23:01:43,662 - climada_petals.hazard.wildfire - INFO - Created 40 fires
2022-05-03 23:01:44,495 - climada_petals.hazard.wildfire - INFO - Created 50 fires
2022-05-03 23:01:45,568 - climada_petals.hazard.wildfire - INFO - Created 60 fires
2022-05-03 23:01:46,360 - climada_petals.hazard.wildfire - INFO - Created 70 fires
2022-05-03 23:01:47,526 - climada_petals.hazard.wildfire - INFO - Created 80 fires
2022-05-03 23:01:48,494 - climada_petals.hazard.wildfire - INFO - Created 90 fires
2022-05-03 23:01:49,354 - climada_petals.hazard.wildfire - INFO - Created 100 fires
2022-05-03 23:01:50,100 - climada_petals.hazard.wildfire - INFO - Created 110 fires
2022-05-03 23:01:51,017 - climada_petals.hazard.wildfire - INFO - Created 120 fires
```

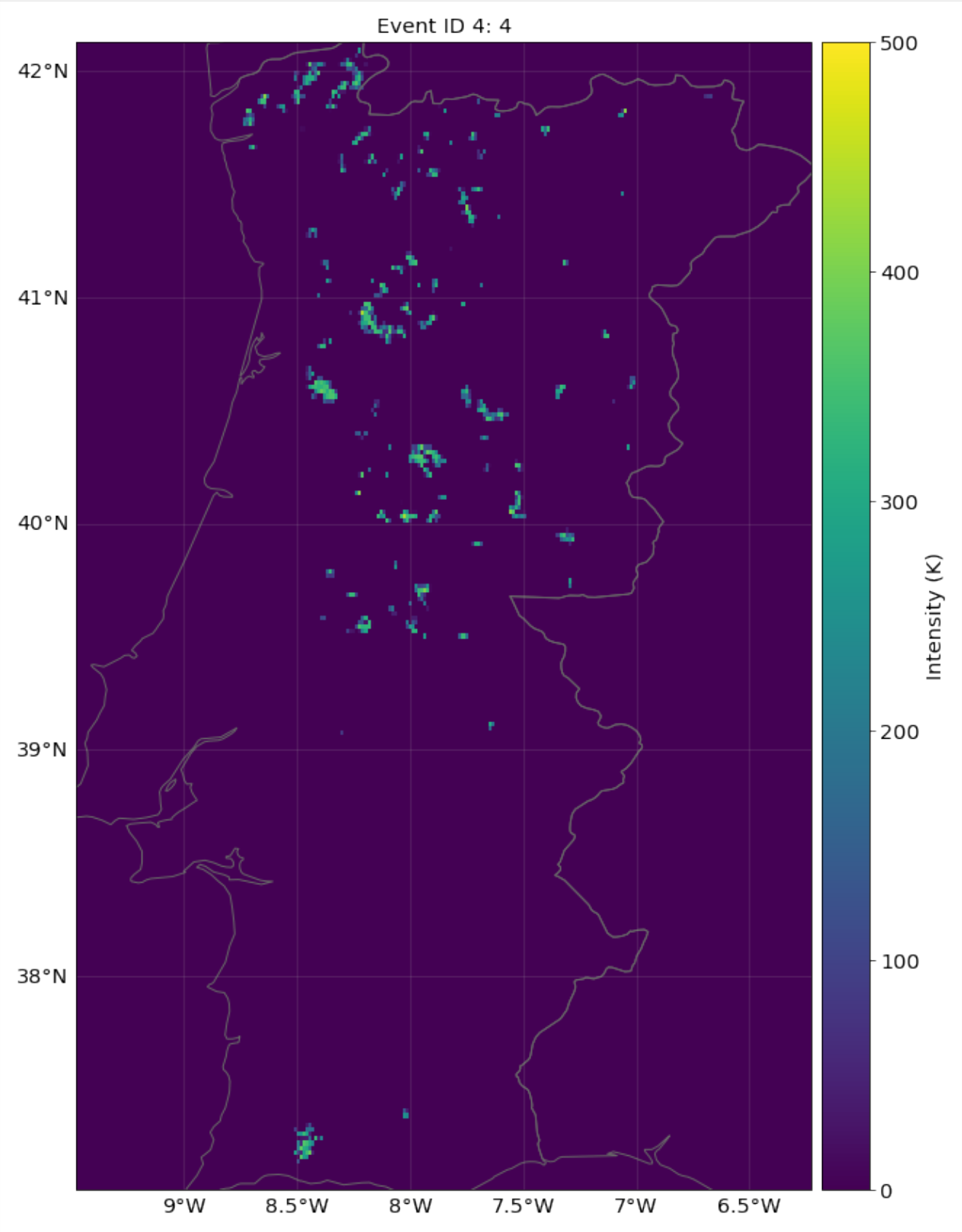
(continues on next page)

(continued from previous page)

```
2022-05-03 23:01:51,618 - climada_petals.hazard.wildfire - INFO - Created 130 fires
2022-05-03 23:01:52,650 - climada_petals.hazard.wildfire - INFO - Created 140 fires
The probabilistic event year is appended to the historic: ['2016' '2017' '2018' '4']
```

```
/home/yuyue/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↪1282: UserWarning: You will likely lose important projection information when␣
↪converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  proj = self._crs.to_proj4(version=version)
```

[6]:
```
<GeoAxesSubplot:title={'center':'Event ID 4: 4'}>
```



[7]:
```
# The overall fire propagation probability is stored as a parameter and can be␣
```

(continues on next page)

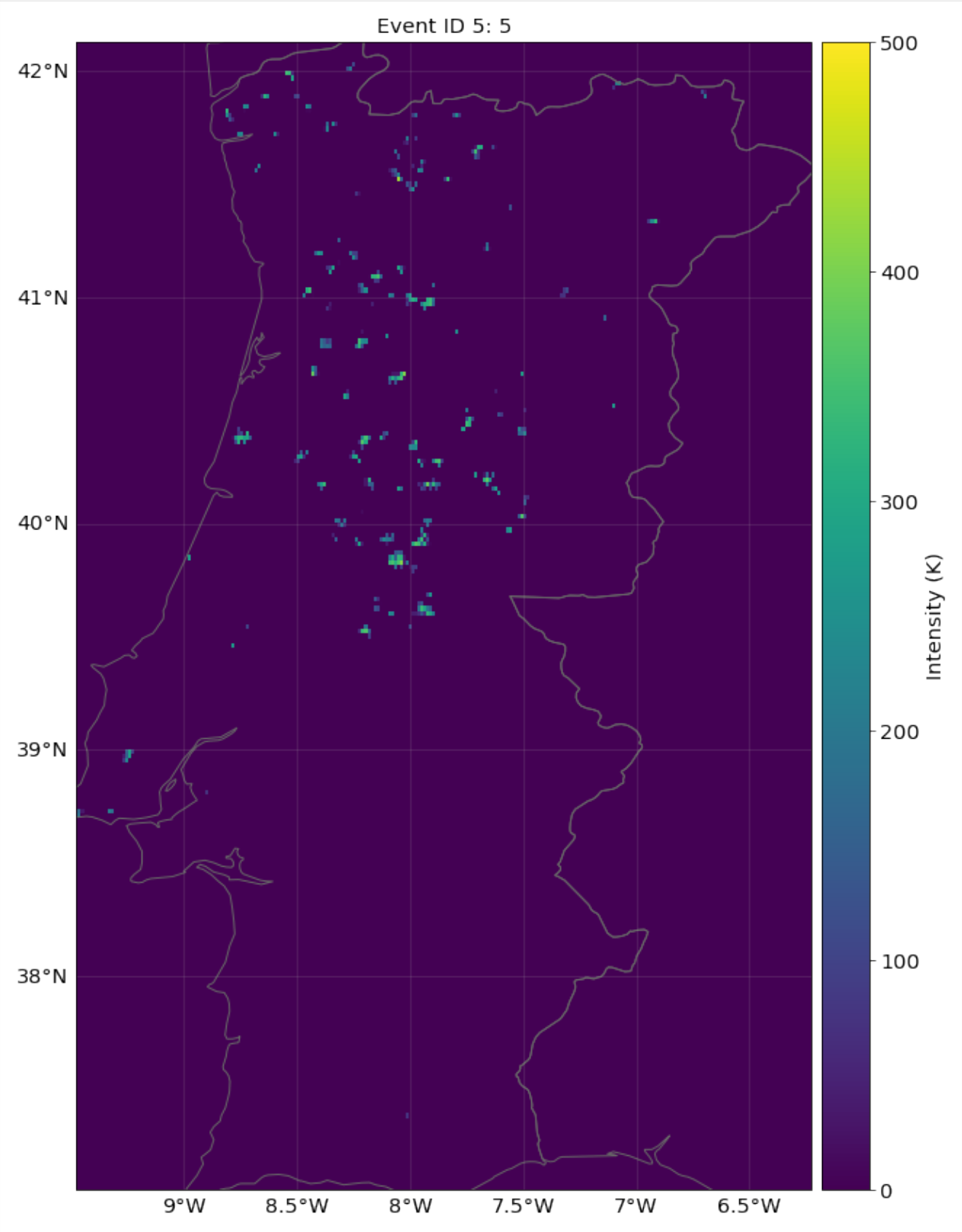
(continued from previous page)

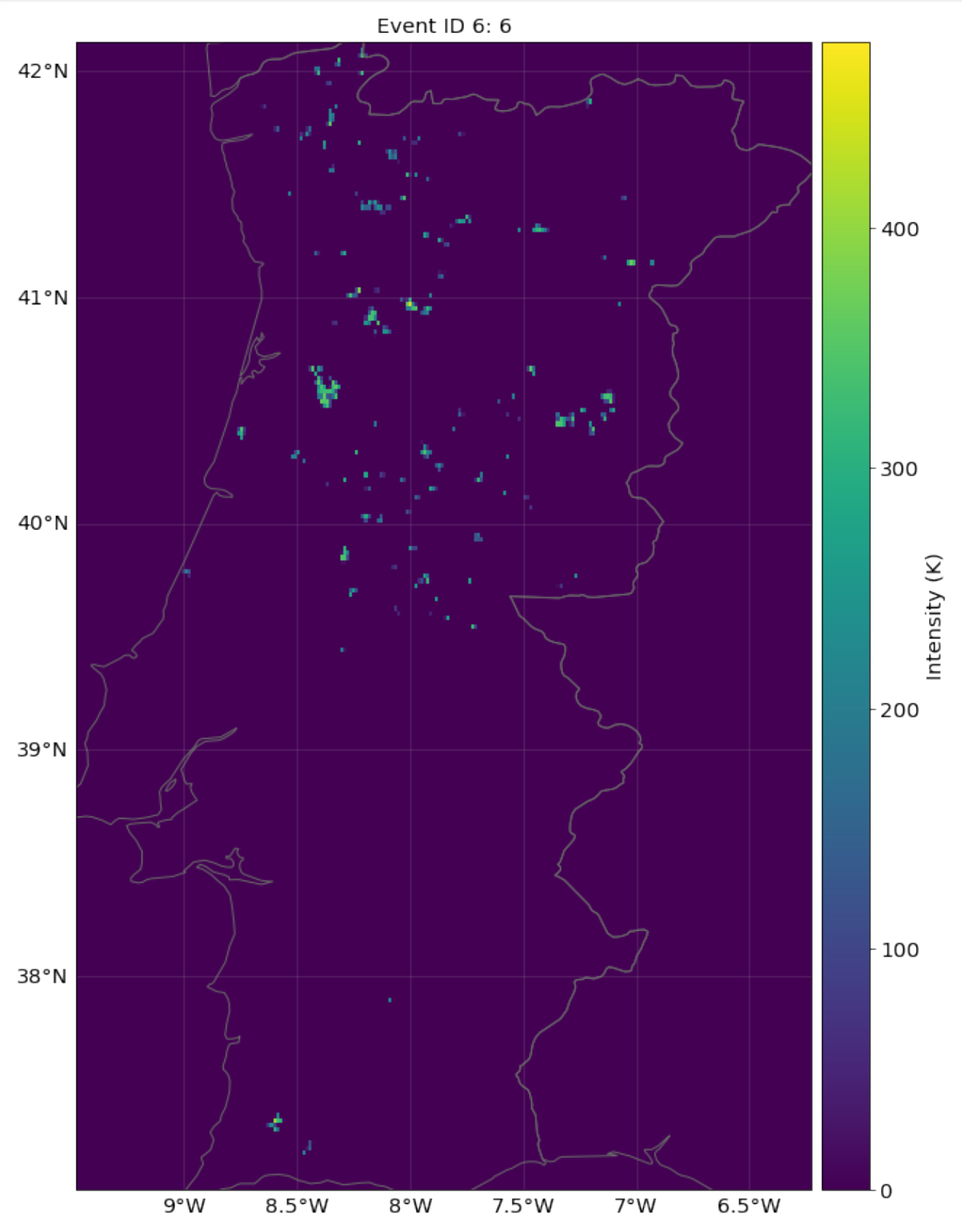
```
↪modified easily (default is 0.21)
wf_years.ProbaParams.prop_proba = 0.18
# The range of the number of events for a synthetic year can be modified when␣
↪creating the event years
ign_range = [125,150]

wf_years.set_proba_fire_seasons(n_fire_seasons=2, n_ignitions=ign_range)
wf_years.plot_intensity(event=5)
wf_years.plot_intensity(event=6);
```

```
2022-05-03 23:04:39,810 - climada_petals.hazard.wildfire - INFO - Setting up␣
↪probabilistic fire season with 128 fires.
2022-05-03 23:04:40,489 - climada_petals.hazard.wildfire - INFO - Created 0 fires
2022-05-03 23:04:40,953 - climada_petals.hazard.wildfire - INFO - Created 10 fires
2022-05-03 23:04:41,592 - climada_petals.hazard.wildfire - INFO - Created 20 fires
2022-05-03 23:04:41,910 - climada_petals.hazard.wildfire - INFO - Created 30 fires
2022-05-03 23:04:42,304 - climada_petals.hazard.wildfire - INFO - Created 40 fires
2022-05-03 23:04:42,719 - climada_petals.hazard.wildfire - INFO - Created 50 fires
2022-05-03 23:04:43,030 - climada_petals.hazard.wildfire - INFO - Created 60 fires
2022-05-03 23:04:43,327 - climada_petals.hazard.wildfire - INFO - Created 70 fires
2022-05-03 23:04:43,733 - climada_petals.hazard.wildfire - INFO - Created 80 fires
2022-05-03 23:04:44,139 - climada_petals.hazard.wildfire - INFO - Created 90 fires
2022-05-03 23:04:44,674 - climada_petals.hazard.wildfire - INFO - Created 100 fires
2022-05-03 23:04:45,295 - climada_petals.hazard.wildfire - INFO - Created 110 fires
2022-05-03 23:04:45,969 - climada_petals.hazard.wildfire - INFO - Created 120 fires
2022-05-03 23:04:46,421 - climada_petals.hazard.wildfire - INFO - Setting up␣
↪probabilistic fire season with 142 fires.
2022-05-03 23:04:47,210 - climada_petals.hazard.wildfire - INFO - Created 0 fires
2022-05-03 23:04:47,643 - climada_petals.hazard.wildfire - INFO - Created 10 fires
2022-05-03 23:04:48,204 - climada_petals.hazard.wildfire - INFO - Created 20 fires
2022-05-03 23:04:48,910 - climada_petals.hazard.wildfire - INFO - Created 30 fires
2022-05-03 23:04:49,586 - climada_petals.hazard.wildfire - INFO - Created 40 fires
2022-05-03 23:04:50,473 - climada_petals.hazard.wildfire - INFO - Created 50 fires
2022-05-03 23:04:51,024 - climada_petals.hazard.wildfire - INFO - Created 60 fires
2022-05-03 23:04:51,372 - climada_petals.hazard.wildfire - INFO - Created 70 fires
2022-05-03 23:04:51,700 - climada_petals.hazard.wildfire - INFO - Created 80 fires
2022-05-03 23:04:51,941 - climada_petals.hazard.wildfire - INFO - Created 90 fires
2022-05-03 23:04:52,261 - climada_petals.hazard.wildfire - INFO - Created 100 fires
2022-05-03 23:04:52,594 - climada_petals.hazard.wildfire - INFO - Created 110 fires
2022-05-03 23:04:52,993 - climada_petals.hazard.wildfire - INFO - Created 120 fires
2022-05-03 23:04:53,510 - climada_petals.hazard.wildfire - INFO - Created 130 fires
2022-05-03 23:04:53,943 - climada_petals.hazard.wildfire - INFO - Created 140 fires
```

```
/home/yuyue/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↪1282: UserWarning: You will likely lose important projection information when␣
↪converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  proj = self._crs.to_proj4(version=version)
/home/yuyue/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↪1282: UserWarning: You will likely lose important projection information when␣
↪converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  proj = self._crs.to_proj4(version=version)
```

[7]:
```
<GeoAxesSubplot:title={'center':'Event ID 6: 6'}>
```


Event ID 5: 5
42°N
41°N
40°N
39°N
38°N
9°W
8.5°W
8°W
7.5°W
7°W
6.5°W
500
400
300
200
100
0
Intensity (K)


Event ID 6: 6
42°N
41°N
40°N
39°N
38°N
9°W
8.5°W
8°W
7.5°W
7°W
6.5°W
400
300
200
100
0
Intensity (K)

Impact calculation

Impact calibration follows the usual CLIMADA logic. Calibrated impact functions are available within `climada.entity.impact_funcs.wildfire`. The calibration was performed on 1, 4 and 10 km resolution (please refer to the work by Lüthi et al. (in prep.).

Note: - as there is the ambiguity between single fires and fire seasons, the `hazard.haz_type` must be set accordingly in the impact function - working on higher resolution often requires better resolved exposure data, especially on rural sites, where the LitPop approach is limited (i.e. an expensive wineyard does not glow in the night)

[ ]:

CHAPTER

THREE

SUPPLYCHAIN CLASS

[1]:
```
import numpy as np
import pandas as pd

from climada.util.api_client import Client
from climada_petals.engine import SupplyChain
from climada.entity import ImpfTropCyclone, ImpactFuncSet
from climada.engine.impact_calc import ImpactCalc

client = Client()
```

This tutorial shows how to use the `SupplyChain` class of CLIMADA. This class allows assessing indirect impacts via Input-Output (IO) modeling. Before diving into this class, it is highly recommended that the user first understands how direct impact is calculated with CLIMADA. This requires getting familiar with the `Exposures`, `Hazard` and `Impact` classes.

This tutorial shows how to set up a global supply chain risk analysis due to tropical cyclones hitting the United States.

CHAPTER

FOUR

1. CALCULATE DIRECT ECONOMIC IMPACTS

The first step is to conduct a direct impact analysis. To do so, we need to define an exposure, an hazard and a vulnerability. In this tutorial we will load the LitPop exposure for the USA from CLIMADA Data API.

[2]:
```
exp_usa = client.get_litpop('USA')
```

```
2023-09-27 15:06:38,917 - climada.entity.exposures.base - INFO - Reading /Users/
↪aciullo/climada/data/exposures/litpop/LitPop_150arcsec_USA/v2/LitPop_150arcsec_USA.
↪hdf5
```

```
/Users/aciullo/opt/anaconda3/envs/climada_env/lib/python3.9/pickle.py:1717:␣
↪UserWarning: Unpickling a shapely <2.0 geometry object. Please save the pickle␣
↪again; shapely 2.1 will not have this compatibility.
  setstate(state)
```

Then, we load a probabilistic hazard set of tropical cyclones for the USA from the CLIMADA Data API.

[3]:
```
tc_usa = client.get_hazard('tropical_cyclone', properties={'country_iso3alpha':'USA',
↪'climate_scenario':'historical'})
```

```
2023-09-27 15:06:49,402 - climada.hazard.base - INFO - Reading /Users/aciullo/climada/
↪data/hazard/tropical_cyclone/tropical_cyclone_10synth_tracks_150arcsec_USA_1980_
↪2020/v2/tropical_cyclone_10synth_tracks_150arcsec_USA_1980_2020.hdf5
```

Then we define vulnerability by loading impact functions for tropical cyclone in the USA:

[4]:
```
# Define impact function
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_set = ImpactFuncSet()
impf_set.append(impf_tc)
impf_set.check()
```

And we finally calculate impacts.

[5]:
```
# Calculate direct impacts to the USA due to TC
imp_calc = ImpactCalc(exp_usa, impf_set, tc_usa)
direct_impact_usa = imp_calc.impact()
```

```
2023-09-27 15:06:49,555 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-09-27 15:06:49,555 - climada.entity.exposures.base - INFO - Matching 643099␣
↪exposures with 503715 centroids.
2023-09-27 15:06:49,562 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2023-09-27 15:06:50,515 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 129140 coordinates.
```

(continues on next page)

(continued from previous page)

```
2023-09-27 15:06:50,542 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪1327884 assets (>0) and 43560 events.
```

CHAPTER

FIVE

2. CALCULATE INDIRECT ECONOMIC IMPACTS

5.1 2.1 Instantiate a `SupplyChain` object by loading the Multi-Regional Input-Output Table of interest.

`SupplyChain` computes indirect economic impacts via Input-Output (IO) modeling. At the core of IO modeling lies an Input-Output Table. `SupplyChain` uses the pymrio python package to download, parse and save Multi-Regional Input Output Tables (MRIOTs). In principle, any IO table can be loaded and used, as long as the structure is consistent with those internally supported by `SupplyChain`, which are: - EXIOBASE3 (1995-2011; 44 countries; 163 industries) - WIOD16 (2000-2014; 43 countries; 56 industries) - OECD21 (1995-2018; 66 countries; 45 industries)

These MRIOTs can be downloaded, parsed and saved automatically.

The first step is to instantiate a `SupplyChain` class. This can be done by passing a customized MRIOT or by calling the `from_mriot` class method to use one of the supported MRIOTs.

[6]:
```
supchain = SupplyChain.from_mriot(mriot_type='WIOD16', mriot_year=2011)
```

```
2023-09-27 15:06:51,329 - climada.util.files_handler - INFO - Downloading https://
↪dataverse.nl/api/access/datafile/199104 to file /Users/aciullo/climada/data/MRIOT/
↪WIOD16/downloads/199104
```

```
898kKB [00:47, 19.0kKB/s]
```

The instatiated class now has an `mriot` attribute, which is a pymrio `IOSystem` object. As such, one can access several info of the MRIOT incl. regions, sectors, total production, transaction matrix and final demand. Please see the pymrio project on how to make best use of all the provided functions.

For example, one can access regions, sectors and IOT data:

[7]:
```
# regions
supchain.mriot.get_regions()
```

[7]:
```
Index(['AUS', 'AUT', 'BEL', 'BGR', 'BRA', 'CAN', 'CHE', 'CHN', 'CYP', 'CZE',
       'DEU', 'DNK', 'ESP', 'EST', 'FIN', 'FRA', 'GBR', 'GRC', 'HRV', 'HUN',
       'IDN', 'IND', 'IRL', 'ITA', 'JPN', 'KOR', 'LTU', 'LUX', 'LVA', 'MEX',
       'MLT', 'NLD', 'NOR', 'POL', 'PRT', 'ROU', 'RUS', 'SVK', 'SVN', 'SWE',
       'TUR', 'TWN', 'USA', 'ROW'],
      dtype='object', name='region')
```

[8]:
```
# sectors
supchain.mriot.get_sectors()
```

[8]:
```
Index(['Crop and animal production, hunting and related service activities',
       'Forestry and logging', 'Fishing and aquaculture',
       'Mining and quarrying',
```

(continues on next page)

(continued from previous page)

```
       'Manufacture of food products, beverages and tobacco products',
       'Manufacture of textiles, wearing apparel and leather products',
       'Manufacture of wood and of products of wood and cork, except furniture;␣
↪manufacture of articles of straw and plaiting materials',
       'Manufacture of paper and paper products',
       'Printing and reproduction of recorded media',
       'Manufacture of coke and refined petroleum products ',
       'Manufacture of chemicals and chemical products ',
       'Manufacture of basic pharmaceutical products and pharmaceutical preparations',
       'Manufacture of rubber and plastic products',
       'Manufacture of other non-metallic mineral products',
       'Manufacture of basic metals',
       'Manufacture of fabricated metal products, except machinery and equipment',
       'Manufacture of computer, electronic and optical products',
       'Manufacture of electrical equipment',
       'Manufacture of machinery and equipment n.e.c.',
       'Manufacture of motor vehicles, trailers and semi-trailers',
       'Manufacture of other transport equipment',
       'Manufacture of furniture; other manufacturing',
       'Repair and installation of machinery and equipment',
       'Electricity, gas, steam and air conditioning supply',
       'Water collection, treatment and supply',
       'Sewerage; waste collection, treatment and disposal activities; materials␣
↪recovery; remediation activities and other waste management services ',
       'Construction',
       'Wholesale and retail trade and repair of motor vehicles and motorcycles',
       'Wholesale trade, except of motor vehicles and motorcycles',
       'Retail trade, except of motor vehicles and motorcycles',
       'Land transport and transport via pipelines', 'Water transport',
       'Air transport',
       'Warehousing and support activities for transportation',
       'Postal and courier activities',
       'Accommodation and food service activities', 'Publishing activities',
       'Motion picture, video and television programme production, sound recording␣
↪and music publishing activities; programming and broadcasting activities',
       'Telecommunications',
       'Computer programming, consultancy and related activities; information service␣
↪activities',
       'Financial service activities, except insurance and pension funding',
       'Insurance, reinsurance and pension funding, except compulsory social security
↪',
       'Activities auxiliary to financial services and insurance activities',
       'Real estate activities',
       'Legal and accounting activities; activities of head offices; management␣
↪consultancy activities',
       'Architectural and engineering activities; technical testing and analysis',
       'Scientific research and development',
       'Advertising and market research',
       'Other professional, scientific and technical activities; veterinary activities
↪',
       'Administrative and support service activities',
       'Public administration and defence; compulsory social security',
       'Education', 'Human health and social work activities',
       'Other service activities',
       'Activities of households as employers; undifferentiated goods- and services-
↪producing activities of households for own use',
       'Activities of extraterritorial organizations and bodies'],
```

(continues on next page)

(continued from previous page)

```
      dtype='object', name='sector')
```

[9]:
```
# transaction matrix
supchain.mriot.Z
```

[9]:
```
region                                                                  ␣
↪                                    AUS  \
sector                                                 Crop and animal production,␣
↪hunting and related service activities
region sector
AUS    Crop and animal production, hunting and related…                 ␣
↪        10954.209508
       Forestry and logging                                             ␣
↪             113.905445
       Fishing and aquaculture                                          ␣
↪              22.267974
       Mining and quarrying                                             ␣
↪             232.437685
       Manufacture of food products, beverages and tob…                 ␣
↪          1662.225516
…                                                                       ␣
↪                  …
ROW    Education                                                        ␣
↪               1.305193
       Human health and social work activities                          ␣
↪               0.535337
       Other service activities                                         ␣
↪               1.856064
       Activities of households as employers; undiffer…                 ␣
↪             0.027816
       Activities of extraterritorial organizations an…                 ␣
↪             0.000000

region                                                                        \
sector                                                 Forestry and logging
region sector
AUS    Crop and animal production, hunting and related…          337.318807
       Forestry and logging                                      127.867760
       Fishing and aquaculture                                     0.025312
       Mining and quarrying                                        2.323883
       Manufacture of food products, beverages and tob…            4.046841
…                                                                        …
ROW    Education                                                   0.022124
       Human health and social work activities                     0.038688
       Other service activities                                    0.058495
       Activities of households as employers; undiffer…            0.000064
       Activities of extraterritorial organizations an…            0.000000

region                                                                              \
sector                                                 Fishing and aquaculture
region sector
AUS    Crop and animal production, hunting and related…             166.404066
       Forestry and logging                                           0.588653
       Fishing and aquaculture                                       21.871529
       Mining and quarrying                                           8.353050
       Manufacture of food products, beverages and tob…              78.987732
…                                                                          …
```

(continues on next page)

(continued from previous page)

```
ROW    Education                                                    0.093113
       Human health and social work activities                      0.141195
       Other service activities                                     0.149588
       Activities of households as employers; undiffer…             0.000191
       Activities of extraterritorial organizations an…             0.000000

region                                                                     \
sector                                                  Mining and quarrying
region sector
AUS    Crop and animal production, hunting and related…          410.533435
       Forestry and logging                                       20.834686
       Fishing and aquaculture                                     7.780646
       Mining and quarrying                                    11305.879190
       Manufacture of food products, beverages and tob…          324.106374
…                                                                       …
ROW    Education                                                  19.406213
       Human health and social work activities                     3.483453
       Other service activities                                   16.013345
       Activities of households as employers; undiffer…           2.084145
       Activities of extraterritorial organizations an…           0.000000

region                                                                                 ␣
↪                                    \
sector                                                  Manufacture of food␣
↪products, beverages and tobacco products
region sector
AUS    Crop and animal production, hunting and related…                                ␣
↪        23137.486536
       Forestry and logging                                                            ␣
↪              2.496693
       Fishing and aquaculture                                                         ␣
↪            388.947468
       Mining and quarrying                                                            ␣
↪            645.526758
       Manufacture of food products, beverages and tob…                                ␣
↪        12908.033161
…                                                                                      ␣
↪                  …
ROW    Education                                                                       ␣
↪              6.600711
       Human health and social work activities                                         ␣
↪              3.362211
       Other service activities                                                        ␣
↪              5.855415
       Activities of households as employers; undiffer…                                ␣
↪            0.355832
       Activities of extraterritorial organizations an…                                ␣
↪            0.000000

region                                                                                 ␣
↪                                    \
sector                                                  Manufacture of textiles,␣
↪wearing apparel and leather products
region sector
AUS    Crop and animal production, hunting and related…                                ␣
↪         498.613588
       Forestry and logging                                                            ␣
```

(continues on next page)

(continued from previous page)

```
↪              0.193944
       Fishing and aquaculture                                                 ␣
↪             32.476194
       Mining and quarrying                                                    ␣
↪             19.184141
       Manufacture of food products, beverages and tob…                        ␣
↪          305.968696
…                                                                              ␣
↪                  …
ROW    Education                                                               ␣
↪              0.683306
       Human health and social work activities                                 ␣
↪              0.861746
       Other service activities                                                ␣
↪              0.346651
       Activities of households as employers; undiffer…                        ␣
↪             0.034880
       Activities of extraterritorial organizations an…                        ␣
↪             0.000000

region                                                                         ␣
↪                                                                              ␣
↪                 \
sector                                                  Manufacture of wood and of␣
↪products of wood and cork, except furniture; manufacture of articles of straw and␣
↪plaiting materials
region sector
AUS    Crop and animal production, hunting and related…                        ␣
↪             4.624214
       Forestry and logging                                                    ␣
↪            1458.441418
       Fishing and aquaculture                                                 ␣
↪              0.302724
       Mining and quarrying                                                    ␣
↪             64.479442
       Manufacture of food products, beverages and tob…                        ␣
↪            11.279065
…                                                                              ␣
↪                  …
ROW    Education                                                               ␣
↪              0.391899
       Human health and social work activities                                 ␣
↪              1.215619
       Other service activities                                                ␣
↪              0.257664
       Activities of households as employers; undiffer…                        ␣
↪             0.035656
       Activities of extraterritorial organizations an…                        ␣
↪             0.000000

region                                                                         ␣
↪              \
sector                                                  Manufacture of paper and␣
↪paper products
region sector
AUS    Crop and animal production, hunting and related…                        ␣
↪ 2.635747
```

(continues on next page)

(continued from previous page)

```
       Forestry and logging                                                    ␣
↪  57.134150
       Fishing and aquaculture                                                 ␣
↪   0.290594
       Mining and quarrying                                                    ␣
↪  99.190650
       Manufacture of food products, beverages and tob…                        ␣
↪18.659012
…                                                                              ␣
↪      …
ROW    Education                                                               ␣
↪   0.911419
       Human health and social work activities                                 ␣
↪   1.110087
       Other service activities                                                ␣
↪   0.644754
       Activities of households as employers; undiffer…                        ␣
↪ 0.163347
       Activities of extraterritorial organizations an…                        ␣
↪ 0.000000

region                                                                         ␣
↪                \
sector                                                 Printing and reproduction␣
↪of recorded media
region sector
AUS    Crop and animal production, hunting and related…                        ␣
↪     1.592439
       Forestry and logging                                                    ␣
↪       3.066086
       Fishing and aquaculture                                                 ␣
↪       0.264986
       Mining and quarrying                                                    ␣
↪      40.675856
       Manufacture of food products, beverages and tob…                        ␣
↪     9.042004
…                                                                              ␣
↪          …
ROW    Education                                                               ␣
↪       0.776923
       Human health and social work activities                                 ␣
↪       0.995861
       Other service activities                                                ␣
↪       0.360074
       Activities of households as employers; undiffer…                        ␣
↪     0.058402
       Activities of extraterritorial organizations an…                        ␣
↪     0.000000

region                                                                         ␣
↪                          \
sector                                                 Manufacture of coke and␣
↪refined petroleum products
region sector
AUS    Crop and animal production, hunting and related…                        ␣
↪            0.850784
       Forestry and logging                                                    ␣
```

(continues on next page)

(continued from previous page)

```
↪              1.055167
       Fishing and aquaculture                                                 ␣
↪              0.214344
       Mining and quarrying                                                    ␣
↪           12437.829210
       Manufacture of food products, beverages and tob…                        ␣
↪              5.859550
…                                                                              ␣
↪                  …
ROW    Education                                                               ␣
↪              0.909038
       Human health and social work activities                                 ␣
↪              0.183099
       Other service activities                                                ␣
↪              0.418569
       Activities of households as employers; undiffer…                        ␣
↪              0.023192
       Activities of extraterritorial organizations an…                        ␣
↪              0.000000

region                                                          …  \
sector                                                          …
region sector                                                   …
AUS    Crop and animal production, hunting and related…         …
       Forestry and logging                                     …
       Fishing and aquaculture                                  …
       Mining and quarrying                                     …
       Manufacture of food products, beverages and tob…         …
…                                                               …
ROW    Education                                                …
       Human health and social work activities                  …
       Other service activities                                 …
       Activities of households as employers; undiffer…         …
       Activities of extraterritorial organizations an…         …

region                                                                         ␣
↪    ROW  \
sector                                                 Scientific research and␣
↪development
region sector
AUS    Crop and animal production, hunting and related…                    5.
↪702333
       Forestry and logging                                                 ␣
↪0.006981
       Fishing and aquaculture                                              ␣
↪0.132873
       Mining and quarrying                                                 ␣
↪3.125064
       Manufacture of food products, beverages and tob…                    2.
↪109958
…                                                                           ␣
↪  …
ROW    Education                                                           ␣
↪184.998529
       Human health and social work activities                              ␣
↪11.443401
       Other service activities                                            ␣
```

(continues on next page)

(continued from previous page)

```
↪662.923912
       Activities of households as employers; undiffer…                          0.
↪000000
       Activities of extraterritorial organizations an…                          0.
↪000000

region                                                                            ␣
↪      \
sector                                                 Advertising and market␣
↪research
region sector
AUS    Crop and animal production, hunting and related…                           0.
↪0
       Forestry and logging                                                       ␣
↪0.0
       Fishing and aquaculture                                                    ␣
↪0.0
       Mining and quarrying                                                       ␣
↪0.0
       Manufacture of food products, beverages and tob…                           0.
↪0
…                                                                                 …
ROW    Education                                                                  ␣
↪0.0
       Human health and social work activities                                    ␣
↪0.0
       Other service activities                                                   ␣
↪0.0
       Activities of households as employers; undiffer…                          0.
↪0
       Activities of extraterritorial organizations an…                          0.
↪0

region                                                                            ␣
↪                                                    \
sector                                                     Other professional,␣
↪scientific and technical activities; veterinary activities
region sector
AUS    Crop and animal production, hunting and related…                           ␣
↪            0.574232
       Forestry and logging                                                       ␣
↪              0.000691
       Fishing and aquaculture                                                    ␣
↪              0.000007
       Mining and quarrying                                                       ␣
↪              7.220349
       Manufacture of food products, beverages and tob…                           ␣
↪            1.408069
…                                                                                 ␣
↪                 …
ROW    Education                                                                  ␣
↪            111.704454
       Human health and social work activities                                    ␣
↪             17.244464
       Other service activities                                                   ␣
↪           1583.310837
       Activities of households as employers; undiffer…                          ␣
```

(continues on next page)

(continued from previous page)

```
↪            0.000000
       Activities of extraterritorial organizations an…                    ␣
↪            0.000000

region                                                                      ␣
↪                   \
sector                                                Administrative and support␣
↪service activities
region sector
AUS    Crop and animal production, hunting and related…                     ␣
↪      22.914001
       Forestry and logging                                                 ␣
↪          0.017126
       Fishing and aquaculture                                              ␣
↪          0.000029
       Mining and quarrying                                                 ␣
↪          3.832179
       Manufacture of food products, beverages and tob…                     ␣
↪        0.971246
…                                                                           ␣
↪             …
ROW    Education                                                            ␣
↪        275.103411
       Human health and social work activities                              ␣
↪         13.649976
       Other service activities                                             ␣
↪       1823.500084
       Activities of households as employers; undiffer…                     ␣
↪        0.000000
       Activities of extraterritorial organizations an…                    ␣
↪        0.000000

region                                                                      ␣
↪                                    \
sector                                                Public administration and␣
↪defence; compulsory social security
region sector
AUS    Crop and animal production, hunting and related…                     ␣
↪             4.833452
       Forestry and logging                                                 ␣
↪               0.019646
       Fishing and aquaculture                                              ␣
↪               0.002032
       Mining and quarrying                                                 ␣
↪              21.010887
       Manufacture of food products, beverages and tob…                     ␣
↪            15.274674
…                                                                           ␣
↪                  …
ROW    Education                                                            ␣
↪            2928.841882
       Human health and social work activities                              ␣
↪             780.123840
       Other service activities                                             ␣
↪           11328.214047
       Activities of households as employers; undiffer…                     ␣
↪             0.000000
```

(continues on next page)

(continued from previous page)

```
       Activities of extraterritorial organizations an…                                ␣
↪            0.000000

region                                                                    \
sector                                                       Education
region sector
AUS    Crop and animal production, hunting and related…    17.818997
       Forestry and logging                                  0.015224
       Fishing and aquaculture                               0.086917
       Mining and quarrying                                 23.183020
       Manufacture of food products, beverages and tob…    23.276147
…                                                                 …
ROW    Education                                          5458.479001
       Human health and social work activities             268.995847
       Other service activities                           6798.108893
       Activities of households as employers; undiffer…     0.000000
       Activities of extraterritorial organizations an…     0.000000

region                                                                                  ␣
↪             \
sector                                                  Human health and social␣
↪work activities
region sector
AUS    Crop and animal production, hunting and related…                                 ␣
↪21.617131
       Forestry and logging                                                             ␣
↪    0.012279
       Fishing and aquaculture                                                          ␣
↪    0.024285
       Mining and quarrying                                                             ␣
↪   42.104660
       Manufacture of food products, beverages and tob…                                 ␣
↪21.356355
…                                                                                       ␣
↪       …
ROW    Education                                                                        ␣
↪ 901.834914
       Human health and social work activities                                          ␣
↪4363.233769
       Other service activities                                                         ␣
↪4401.062672
       Activities of households as employers; undiffer…                                ␣
↪ 0.000000
       Activities of extraterritorial organizations an…                                ␣
↪ 0.000000

region                                                                          \
sector                                                  Other service activities
region sector
AUS    Crop and animal production, hunting and related…                9.703475
       Forestry and logging                                            0.024694
       Fishing and aquaculture                                         0.036828
       Mining and quarrying                                           15.832950
       Manufacture of food products, beverages and tob…               26.949427
…                                                                           …
ROW    Education                                                     283.139286
       Human health and social work activities                        85.398007
```

(continues on next page)

(continued from previous page)

```
       Other service activities                                    15253.490477
       Activities of households as employers; undiffer…              0.000000
       Activities of extraterritorial organizations an…              0.000000

region                                                                        ␣
↪                                                                             ␣
↪            \
sector                                                 Activities of households as␣
↪employers; undifferentiated goods- and services-producing activities of households␣
↪for own use
region sector
AUS    Crop and animal production, hunting and related…                       ␣
↪                0.0
       Forestry and logging                                                   ␣
↪                 0.0
       Fishing and aquaculture                                                ␣
↪                 0.0
       Mining and quarrying                                                   ␣
↪                 0.0
       Manufacture of food products, beverages and tob…                       ␣
↪                0.0
…                                                                             ␣
↪                 …
ROW    Education                                                              ␣
↪                 0.0
       Human health and social work activities                                ␣
↪                 0.0
       Other service activities                                               ␣
↪                 0.0
       Activities of households as employers; undiffer…                       ␣
↪                0.0
       Activities of extraterritorial organizations an…                       ␣
↪                0.0

region
sector                                                 Activities of␣
↪extraterritorial organizations and bodies
region sector
AUS    Crop and animal production, hunting and related…                       ␣
↪        1.159407e-06
       Forestry and logging                                                   ␣
↪          5.846157e-08
       Fishing and aquaculture                                                ␣
↪          4.824417e-09
       Mining and quarrying                                                   ␣
↪          5.314913e-04
       Manufacture of food products, beverages and tob…                       ␣
↪        1.579590e-05
…                                                                             ␣
↪                 …
ROW    Education                                                              ␣
↪          5.338385e-03
       Human health and social work activities                                ␣
↪          3.905298e-04
       Other service activities                                               ␣
↪          7.981605e-01
       Activities of households as employers; undiffer…                       ␣
```

(continues on next page)

(continued from previous page)

```
↪        0.000000e+00
      Activities of extraterritorial organizations an…                    ␣
↪        0.000000e+00

[2464 rows x 2464 columns]
```

[10]:
```
# final demand
supchain.mriot.Y
```

[10]:
```
                                                        final demand
region sector
AUS    Crop and animal production, hunting and related…   14467.050267
       Forestry and logging                                 342.352502
       Fishing and aquaculture                             1787.071582
       Mining and quarrying                               19445.035584
       Manufacture of food products, beverages and tob…   51480.063474
…                                                                  …
ROW    Education                                          483744.763934
       Human health and social work activities            382325.468496
       Other service activities                           234071.579585
       Activities of households as employers; undiffer…   14600.614007
       Activities of extraterritorial organizations an…      33.142482

[2464 rows x 1 columns]
```

[11]:
```
# total production
supchain.mriot.x
```

[11]:
```
                                                       total production
region sector
AUS    Crop and animal production, hunting and related…      69566.630223
       Forestry and logging                                   2719.293635
       Fishing and aquaculture                                3113.703184
       Mining and quarrying                                 250207.439237
       Manufacture of food products, beverages and tob…      92033.592425
…                                                                       …
ROW    Education                                            508147.713351
       Human health and social work activities              399362.733830
       Other service activities                             367396.468789
       Activities of households as employers; undiffer…      22003.013824
       Activities of extraterritorial organizations an…         33.142482

[2464 rows x 1 columns]
```

5.2 2.2 Assign stock exposure and impact to MRIOT countries-sectors

After loading the MRIOT, one needs to translate the direct impacts previously calculated - which are defined at an arbitrary spatial resolution - into impacts to sectors and countries defined by the MRIOT. To do this one needs to know what countries and sectors the used exposure represents.

The first is straightforward, as exposure contains latitude and longitude information, and even a regional id that often defines the country of interest.

The latter needs input from the user, who needs to know/assess what sectors in the MRIOT the used exposure represents. For example, assuming the `LitPop` exposure is representative of the service sector, and assuming that sub-sectors at positions 26 to 56 in `WIOD16` do represent this sector, then one can translate spatially disaggregated impacts into country/sector impacts as follows:

[12]:
```
impacted_secs = supchain.mriot.get_sectors()[range(26,56)].tolist()
supchain.calc_shock_to_sectors(exp_usa, direct_impact_usa, impacted_secs)
```

Which creates the attributes `self.secs_exp`, `self.secs_imp`, and `self.secs_shock`. The first two show `Exposure` and `Impact` values at the country-sector level. This translation is accomplished assuming that exposure/impact of an affected sector is proportional to this sector's contribution to the overall production of all affected sectors. For example, if the total (spatially distributed) exposed value is 100, and there are two affected sectors, A (whose production is 2) and B (whose production is 8), then sector A has an exposure of 20 and sector B has an exposure of 80. The same reasoning is applied to the distributions of direct impacts.

One can easily check that `self.secs_exp`, `self.secs_imp` have the same total values of `Exposure` and `Impact` and that this only involves the directly hit countries-sectors :

[13]:
```
# exposure
print(
      exp_usa.gdf.value.sum(),
      supchain.secs_exp.sum().sum(),
      supchain.secs_exp.loc[:, ('USA', impacted_secs)].sum().sum(),
      )

# impact
print(supchain.secs_imp.sum().sum(),
      supchain.secs_imp.loc[:, ('USA', impacted_secs)].sum().sum(),
      direct_impact_usa.imp_mat.sum().sum())
```

```
65384554304412.63 65384554304412.64 65384554304412.64
16543899066892.074 16543899066892.07 16543899066892.068
```

The attribute `self.secs_shock` is proportional to the ratio between `self.secs_imp` and `self.secs_exp`, so `self.secs_shock` is a number between 0 and 1. `self.secs_shock` will be used in the indirect impact calculation to assses how much production loss is experienced by each sector.

In terms of structure, it is a dataframe with columns the MRIOT's columns, and with index the event-ids of the hazard events that have *non-zero impacts*:

[14]:
```
supchain.secs_shock.loc[:, ('USA', impacted_secs)].head()
```

[14]:
```
region            USA  \
sector   Construction
event_id
10594        0.000352
10597        0.000634
10600        0.000694
10603        0.000032
10604        0.000071

region                                                                  \
sector   Wholesale and retail trade and repair of motor vehicles and motorcycles
event_id
10594                                                 0.000352
10597                                                 0.000634
10600                                                 0.000694
10603                                                 0.000032
```

(continues on next page)

(continued from previous page)

```
10604                                                 0.000071

region                                                                  \
sector   Wholesale trade, except of motor vehicles and motorcycles
event_id
10594                                                 0.000352
10597                                                 0.000634
10600                                                 0.000694
10603                                                 0.000032
10604                                                 0.000071

region                                                              \
sector   Retail trade, except of motor vehicles and motorcycles
event_id
10594                                                 0.000352
10597                                                 0.000634
10600                                                 0.000694
10603                                                 0.000032
10604                                                 0.000071

region                                                                   \
sector   Land transport and transport via pipelines Water transport
event_id
10594                                    0.000352        0.000352
10597                                    0.000634        0.000634
10600                                    0.000694        0.000694
10603                                    0.000032        0.000032
10604                                    0.000071        0.000071

region                                                                                \
sector   Air transport Warehousing and support activities for transportation
event_id
10594         0.000352                                                 0.000352
10597         0.000634                                                 0.000634
10600         0.000694                                                 0.000694
10603         0.000032                                                 0.000032
10604         0.000071                                                 0.000071

region                                    \
sector   Postal and courier activities
event_id
10594                          0.000352
10597                          0.000634
10600                          0.000694
10603                          0.000032
10604                          0.000071

region                                                 …  \
sector   Accommodation and food service activities      …
event_id                                               …
10594                                       0.000352   …
10597                                       0.000634   …
10600                                       0.000694   …
10603                                       0.000032   …
10604                                       0.000071   …

region                                                                              \
```

(continues on next page)

(continued from previous page)

```
sector   Scientific research and development Advertising and market research
event_id
10594                                0.000352                       0.000352
10597                                0.000634                       0.000634
10600                                0.000694                       0.000694
10603                                0.000032                       0.000032
10604                                0.000071                       0.000071

region                                                                      ␣
↪    \
sector   Other professional, scientific and technical activities; veterinary␣
↪activities
event_id
10594                                                 0.000352
10597                                                 0.000634
10600                                                 0.000694
10603                                                 0.000032
10604                                                 0.000071

region                                                    \
sector   Administrative and support service activities
event_id
10594                                          0.000352
10597                                          0.000634
10600                                          0.000694
10603                                          0.000032
10604                                          0.000071

region                                                                \
sector   Public administration and defence; compulsory social security
event_id
10594                                                 0.000352
10597                                                 0.000634
10600                                                 0.000694
10603                                                 0.000032
10604                                                 0.000071

region                                                          \
sector   Education Human health and social work activities
event_id
10594     0.000352                                    0.000352
10597     0.000634                                    0.000634
10600     0.000694                                    0.000694
10603     0.000032                                    0.000032
10604     0.000071                                    0.000071

region                              \
sector   Other service activities
event_id
10594                    0.000352
10597                    0.000634
10600                    0.000694
10603                    0.000032
10604                    0.000071

region                                                                      ␣
↪                                             \
```

(continues on next page)

(continued from previous page)

```
sector   Activities of households as employers; undifferentiated goods- and services-
↪producing activities of households for own use
event_id
10594                                                 0.000352
10597                                                 0.000634
10600                                                 0.000694
10603                                                 0.000032
10604                                                 0.000071

region
sector   Activities of extraterritorial organizations and bodies
event_id
10594                                                      0.0
10597                                                      0.0
10600                                                      0.0
10603                                                      0.0
10604                                                      0.0

[5 rows x 30 columns]
```

By default, `self.secs_shock` is exactly the ratio between `Impact` and `Exposure`, which results in the same shock across sectors for a given event:

[15]:
```
# let's try the first three events
for event_id in supchain.secs_shock.index[:3]:
    imp_event = direct_impact_usa.at_event[direct_impact_usa.event_id == event_id][0]
    print(imp_event / exp_usa.gdf.value.sum(), supchain.secs_shock.loc[event_id, ('USA
↪', impacted_secs)].values[0])
```

```
0.0003524059371639781 0.0003524059371639781
0.0006340759642834555 0.0006340759642834556
0.0006936414424383894 0.0006936414424383894
```

This practically means that the fraction of *production* losses is assumed to be equal to the fractions of *stock* losses, since `Impact` and `Exposure` typically refer to stocks in CLIMADA. However, since depending on the sector one can reasonably expect production losses to be proportionally higher or lower than stock losses, a `shock_factor` argument can also be passed to define - for each sector - how much should production shocks be higher/lower than stocks shocks (i.e., the mere `Impact` / `Exposure` ratio):

[16]:
```
shock_factor = pd.DataFrame(np.repeat(1, supchain.mriot.x.shape[0]),
                            index=supchain.mriot.x.index,
                            columns=['shock'])

# randomly generated for this tutorial
shock_facs_service_USA = np.array([
                                   0.38324804, 1.15930626, 0.73846477, 0.5430206 , 0.
↪54147014,
                                   0.28362671, 0.53829353, 1.95367016, 1.33675622, 0.
↪42285787,
                                   0.86974667, 1.4685637 , 1.24804793, 0.56915521, 0.
↪43723048,
                                   0.23372398, 0.69268485, 0.74130451, 0.74739106, 1.
↪18719852,
                                   1.02203697, 1.0412411 , 0.09315484, 1.23612412, 0.
↪55947349,
                                   0.8608431, 0.58983156, 1.13137055, 0.93014364, 0.
↪39092134
```

(continues on next page)

(continued from previous page)

```
                                    ])

shock_factor.loc[('USA', impacted_secs), :] = shock_facs_service_USA

# supply shock factors when calculating sectorial shocks
supchain.calc_shock_to_sectors(exp_usa, direct_impact_usa, impacted_secs, shock_
↪factor.values.flatten())
```

[17]:
```
supchain.secs_shock.loc[:, ('USA', impacted_secs)].head()
```

[17]:
```
region            USA  \
sector   Construction
event_id
10594        0.000135
10597        0.000243
10600        0.000266
10603        0.000012
10604        0.000027

region                                                                    \
sector   Wholesale and retail trade and repair of motor vehicles and motorcycles
event_id
10594                                                0.000409
10597                                                0.000735
10600                                                0.000804
10603                                                0.000037
10604                                                0.000082

region                                                          \
sector   Wholesale trade, except of motor vehicles and motorcycles
event_id
10594                                                0.000260
10597                                                0.000468
10600                                                0.000512
10603                                                0.000024
10604                                                0.000052

region                                                              \
sector   Retail trade, except of motor vehicles and motorcycles
event_id
10594                                                0.000191
10597                                                0.000344
10600                                                0.000377
10603                                                0.000017
10604                                                0.000039

region                                                                  \
sector   Land transport and transport via pipelines Water transport
event_id
10594                                     0.000191        0.000100
10597                                     0.000343        0.000180
10600                                     0.000376        0.000197
10603                                     0.000017        0.000009
10604                                     0.000038        0.000020

region                                                                       \
sector   Air transport Warehousing and support activities for transportation
```

(continues on next page)

(continued from previous page)

```
event_id
10594         0.000190                                                0.000688
10597         0.000341                                                0.001239
10600         0.000373                                                0.001355
10603         0.000017                                                0.000062
10604         0.000038                                                0.000139

region                                    \
sector   Postal and courier activities
event_id
10594                          0.000471
10597                          0.000848
10600                          0.000927
10603                          0.000043
10604                          0.000095

region                                                 …  \
sector   Accommodation and food service activities     …
event_id                                               …
10594                                       0.000149   …
10597                                       0.000268   …
10600                                       0.000293   …
10603                                       0.000013   …
10604                                       0.000030   …

region                                                                    \
sector   Scientific research and development Advertising and market research
event_id
10594                                 0.000360                       0.000367
10597                                 0.000648                       0.000660
10600                                 0.000709                       0.000722
10603                                 0.000033                       0.000033
10604                                 0.000072                       0.000074

region                                                                    ␣
↪    \
sector   Other professional, scientific and technical activities; veterinary␣
↪activities
event_id
10594                                                 0.000033
10597                                                 0.000059
10600                                                 0.000065
10603                                                 0.000003
10604                                                 0.000007

region                                                    \
sector   Administrative and support service activities
event_id
10594                                        0.000436
10597                                        0.000784
10600                                        0.000857
10603                                        0.000039
10604                                        0.000088

region                                                                \
sector   Public administration and defence; compulsory social security
event_id
```

(continues on next page)

(continued from previous page)

```
10594                                                    0.000197
10597                                                    0.000355
10600                                                    0.000388
10603                                                    0.000018
10604                                                    0.000040

region                                                          \
sector   Education Human health and social work activities
event_id
10594     0.000303                                    0.000208
10597     0.000546                                    0.000374
10600     0.000597                                    0.000409
10603     0.000027                                    0.000019
10604     0.000061                                    0.000042

region                                \
sector   Other service activities
event_id
10594                    0.000399
10597                    0.000717
10600                    0.000785
10603                    0.000036
10604                    0.000080

region                                                                          ␣
↪                                                    \
sector   Activities of households as employers; undifferentiated goods- and services-
↪producing activities of households for own use
event_id
10594                                                    0.000328
10597                                                    0.000590
10600                                                    0.000645
10603                                                    0.000030
10604                                                    0.000066

region
sector   Activities of extraterritorial organizations and bodies
event_id
10594                                                       0.0
10597                                                       0.0
10600                                                       0.0
10603                                                       0.0
10604                                                       0.0

[5 rows x 30 columns]
```

Even though the default values (all one) for the shock factors are correct only in the (uncommon) case in which CLIMADA's direct impacts already express production losses, a proper assignment of the shock factors requires extensive expert knowledge on how a sector's production responds to the sector's stock losses. Therefore, it is recommended to change these values only when detailed knowledge about the relationship between stock and production losses is available.

5.3 2.3 Calculate the propagation of production losses

After sectorial production shocks are defined, one can calculate the extent of the shocks and how these propagate through the supply chain. This can be done with the via the `self.calc_impacts` method by specifying the `ghosh` or `leontief` approach. Both approached can be run on the same instantiation of `SupplyChain` and results are stored in a dictionary with keys equal to the method's name. In addition, `self.calc_impacts` calls `self.calc_shock_to_sectors` if `self.secs_exp`, `self.secs_imp` and `self.secs_shock` are not defined, in which case the respective arguments need to be passed to `self.calc_impacts`.

[18]:
```
supchain.calc_impacts(io_approach='ghosh')
supchain.calc_impacts(io_approach='leontief')
```

This creates the `supchain_imp` dictionary:

[19]:
```
supchain.supchain_imp.keys()
```

[19]:
```
dict_keys(['ghosh', 'leontief'])
```

[20]:
```
supchain.supchain_imp['ghosh'].head()
```

[20]:
```
region                                                                  AUS  \
sector   Crop and animal production, hunting and related service activities
event_id
10594                                                          0.108493
10597                                                          0.195209
10600                                                          0.213548
10603                                                          0.009819
10604                                                          0.021834

region                                                                        \
sector   Forestry and logging Fishing and aquaculture Mining and quarrying
event_id
10594                0.003844                0.005600             0.375871
10597                0.006916                0.010075             0.676297
10600                0.007566                0.011022             0.739829
10603                0.000348                0.000507             0.034016
10604                0.000774                0.001127             0.075643

region                                                                 \
sector   Manufacture of food products, beverages and tobacco products
event_id
10594                                                          0.163791
10597                                                          0.294706
10600                                                          0.322391
10603                                                          0.014823
10604                                                          0.032963

region                                                                  \
sector   Manufacture of textiles, wearing apparel and leather products
event_id
10594                                                          0.016370
10597                                                          0.029454
10600                                                          0.032220
10603                                                          0.001481
10604                                                          0.003294

region
```

(continues on next page)

(continued from previous page)

```
↪                                                    \
sector   Manufacture of wood and of products of wood and cork, except furniture;␣
↪manufacture of articles of straw and plaiting materials
event_id
10594                                                0.017644
10597                                                0.031746
10600                                                0.034728
10603                                                0.001597
10604                                                0.003551

region                                              \
sector   Manufacture of paper and paper products
event_id
10594                                    0.023483
10597                                    0.042252
10600                                    0.046221
10603                                    0.002125
10604                                    0.004726

region                                                   \
sector   Printing and reproduction of recorded media
event_id
10594                                         0.019950
10597                                         0.035895
10600                                         0.039267
10603                                         0.001805
10604                                         0.004015

region                                                          …  \
sector   Manufacture of coke and refined petroleum products      …
event_id                                                         …
10594                                                0.045949    …
10597                                                0.082674    …
10600                                                0.090441    …
10603                                                0.004158    …
10604                                                0.009247    …

region                                       ROW                                    \
sector   Scientific research and development Advertising and market research
event_id
10594                                  1.831726                                0.0
10597                                  3.295782                                0.0
10600                                  3.605390                                0.0
10603                                  0.165769                                0.0
10604                                  0.368630                                0.0

region                                                                            ␣
↪    \
sector   Other professional, scientific and technical activities; veterinary␣
↪activities
event_id
10594                                                0.698952
10597                                                1.257608
10600                                                1.375748
10603                                                0.063254
10604                                                0.140662
```

(continues on next page)

(continued from previous page)

```
region                                                      \
sector   Administrative and support service activities
event_id
10594                                          1.212268
10597                                          2.181207
10600                                          2.386111
10603                                          0.109709
10604                                          0.243966

region                                                                  \
sector   Public administration and defence; compulsory social security
event_id
10594                                                 3.918551
10597                                                 7.050559
10600                                                 7.712893
10603                                                 0.354624
10604                                                 0.788597

region                                                            \
sector   Education Human health and social work activities
event_id
10594     4.494818                                 2.555403
10597     8.087423                                 4.597877
10600     8.847160                                 5.029805
10603     0.406776                                 0.231261
10604     0.904569                                 0.514267

region                              \
sector   Other service activities
event_id
10594                   2.139706
10597                   3.849924
10600                   4.211588
10603                   0.193641
10604                   0.430610

region                                                                  ␣
↪                                                  \
sector   Activities of households as employers; undifferentiated goods- and services-
↪producing activities of households for own use
event_id
10594                                                    0.0
10597                                                    0.0
10600                                                    0.0
10603                                                    0.0
10604                                                    0.0

region
sector   Activities of extraterritorial organizations and bodies
event_id
10594                                             0.000542
10597                                             0.000976
10600                                             0.001068
10603                                             0.000049
10604                                             0.000109

[5 rows x 2464 columns]
```

[21]:
```
supchain.supchain_imp['leontief'].head()
```

[21]:
```
region                                                                 AUS  \
sector   Crop and animal production, hunting and related service activities
event_id
10594                                                          0.046333
10597                                                          0.083367
10600                                                          0.091198
10603                                                          0.004193
10604                                                          0.009324

region                                                                       \
sector   Forestry and logging Fishing and aquaculture Mining and quarrying
event_id
10594                0.004260                0.001743             0.921647
10597                0.007665                0.003136             1.658299
10600                0.008385                0.003431             1.814080
10603                0.000386                0.000158             0.083408
10604                0.000857                0.000351             0.185479

region                                                                 \
sector   Manufacture of food products, beverages and tobacco products
event_id
10594                                                          0.053181
10597                                                          0.095688
10600                                                          0.104677
10603                                                          0.004813
10604                                                          0.010703

region                                                                  \
sector   Manufacture of textiles, wearing apparel and leather products
event_id
10594                                                          0.014059
10597                                                          0.025295
10600                                                          0.027671
10603                                                          0.001272
10604                                                          0.002829

region                                                                  ␣
↪                                                      \
sector   Manufacture of wood and of products of wood and cork, except furniture;␣
↪manufacture of articles of straw and plaiting materials
event_id
10594                                                          0.007054
10597                                                          0.012692
10600                                                          0.013884
10603                                                          0.000638
10604                                                          0.001420

region                                                 \
sector   Manufacture of paper and paper products
event_id
10594                                   0.031176
10597                                   0.056094
10600                                   0.061364
10603                                   0.002821
10604                                   0.006274
```

(continues on next page)

(continued from previous page)

```
region                                                  \
sector   Printing and reproduction of recorded media
event_id
10594                                        0.010033
10597                                        0.018053
10600                                        0.019749
10603                                        0.000908
10604                                        0.002019

region                                                        …  \
sector   Manufacture of coke and refined petroleum products    …
event_id                                                      …
10594                                               0.045333  …
10597                                               0.081567  …
10600                                               0.089230  …
10603                                               0.004103  …
10604                                               0.009123  …

region                                       ROW                                    \
sector   Scientific research and development Advertising and market research
event_id
10594                                  0.455992                        0.450742
10597                                  0.820456                        0.811010
10600                                  0.897531                        0.887197
10603                                  0.041267                        0.040792
10604                                  0.091767                        0.090711

region                                                                             ␣
↪    \
sector   Other professional, scientific and technical activities; veterinary␣
↪activities
event_id
10594                                               0.459801
10597                                               0.827310
10600                                               0.905028
10603                                               0.041612
10604                                               0.092534

region                                                    \
sector   Administrative and support service activities
event_id
10594                                       10.275136
10597                                       18.487818
10600                                       20.224575
10603                                        0.929888
10604                                        2.067842

region                                                                \
sector   Public administration and defence; compulsory social security
event_id
10594                                               0.308279
10597                                               0.554679
10600                                               0.606786
10603                                               0.027899
10604                                               0.062040

region                                                        \
```

(continues on next page)

(continued from previous page)

```
sector   Education Human health and social work activities
event_id
10594     0.250009                                  0.502068
10597     0.449835                                  0.903360
10600     0.492093                                  0.988222
10603     0.022626                                  0.045437
10604     0.050314                                  0.101040

region                                \
sector   Other service activities
event_id
10594                    0.801998
10597                    1.443016
10600                    1.578574
10603                    0.072580
10604                    0.161400

region                                                                          ␣
↪                                                    \
sector   Activities of households as employers; undifferentiated goods- and services-
↪producing activities of households for own use
event_id
10594                                                 0.020219
10597                                                 0.036379
10600                                                 0.039796
10603                                                 0.001830
10604                                                 0.004069

region
sector   Activities of extraterritorial organizations and bodies
event_id
10594                                                       0.0
10597                                                       0.0
10600                                                       0.0
10603                                                       0.0
10604                                                       0.0

[5 rows x 2464 columns]
```

values of the dictionary are `pandas.DataFrame` with indexes equal to the hazard events id *leading to non-zero impact* and columns representing the countries-sectors in the MRIOT. Numbers are expressed **in the same unit** as the used MRIOT and they represent global production losses due to tropical cyclones in the USA.

The unit can be checked by doing:

[22]:
```
supchain.mriot.unit
```

[22]:
```
                                                      unit
region sector
AUS    Crop and animal production, hunting and related…  M.EUR
       Forestry and logging                               M.EUR
       Fishing and aquaculture                            M.EUR
       Mining and quarrying                               M.EUR
       Manufacture of food products, beverages and tob…  M.EUR
…                                                         …
ROW    Education                                          M.EUR
       Human health and social work activities            M.EUR
```

(continues on next page)

(continued from previous page)

```
       Other service activities                                M.EUR
       Activities of households as employers; undiffer…  M.EUR
       Activities of extraterritorial organizations an…  M.EUR

[2464 rows x 1 columns]
```

As an example, the 10 largest impacted Swiss sectors from such events according to Ghosh are:

[23]:
```
supchain.supchain_imp['ghosh'].loc[:,('CHE', slice(None))].max(0).sort_
↪values(ascending=False)[:10]
```

[23]:
```
region  sector
CHE     Manufacture of basic pharmaceutical products and pharmaceutical preparations␣
↪              43.403808
        Wholesale trade, except of motor vehicles and motorcycles                 ␣
↪              19.367726
        Computer programming, consultancy and related activities; information service␣
↪activities    10.181569
        Financial service activities, except insurance and pension funding        ␣
↪               8.498905
        Land transport and transport via pipelines                                ␣
↪               8.182419
        Manufacture of computer, electronic and optical products                  ␣
↪               7.778191
        Insurance, reinsurance and pension funding, except compulsory social security␣
↪               6.988161
        Construction                                                              ␣
↪               6.603282
        Electricity, gas, steam and air conditioning supply                       ␣
↪               6.543271
        Human health and social work activities                                   ␣
↪               5.958961
dtype: float64
```

While for Leontief are:

[24]:
```
supchain.supchain_imp['leontief'].loc[:,('CHE', slice(None))].max(0).sort_
↪values(ascending=False)[:10]
```

[24]:
```
region  sector
CHE     Insurance, reinsurance and pension funding, except compulsory social security␣
↪                   75.087467
        Manufacture of computer, electronic and optical products                  ␣
↪                   21.647353
        Wholesale trade, except of motor vehicles and motorcycles                 ␣
↪                   13.481617
        Manufacture of basic pharmaceutical products and pharmaceutical preparations␣
↪                   12.788959
        Manufacture of chemicals and chemical products                            ␣
↪                   10.844197
        Legal and accounting activities; activities of head offices; management␣
↪consultancy activities    10.724539
        Financial service activities, except insurance and pension funding        ␣
↪                    9.149769
        Electricity, gas, steam and air conditioning supply                       ␣
↪                    7.972747
        Manufacture of furniture; other manufacturing                             ␣
↪                    5.372191
```

(continues on next page)

(continued from previous page)

```
        Manufacture of fabricated metal products, except machinery and equipment      ␣
↪                    5.251315
dtype: float64
```

CHAPTER

SIX

BLACKMARBLE CLASS

The `BlackMarble` class inherits the `Exposures <https://climada-python.readthedocs.io/en/stable/tutorial/climada_entity_Exposures.html>`__ class and it is used to approximate economic exposure. This class models exposures of countries and provinces by interpolating GDP and income group values through the nightlight intensities of a specific year.

6.1 Input data:

The used nightlight images are the following: - from 2012 to 2016: https://earthobservatory.nasa.gov/Features/NightLights (15 arcsec resolution (~500m)) - from 1992 to 2013: https://ngdc.noaa.gov/eog/dmsp/downloadV4composites.html (30 arcsec resolution (~1km), stable lights)

By default, for years higher than 2013 the NASA images are used, whilst for 2013 and earlier years the NOAA ones are considered. However, there is a flag which allows to choose the closest NASA or NOAA images. The default resolution is that of the image, but any resolution can be set and will be computed by interpolation.

Regarding GDP (nominal GDP at current USD) and income group values, they are obtained from the World Bank using the pandas-datareader API. If a value is missing, the value of the closest year is considered. When no values are provided from the World Bank, we use the Natural Earth repository values.

6.2 Import BlackMarble()

[15]:
```
%matplotlib inline
from climada_petals.entity.exposures import BlackMarble
```

6.3 Possible settings

The class provides a `set_countries()` method which enables to model a country using different settings. The first time a nightlight image is used, it is downloaded and stored locally. This might take some time.

Let's look into `set_countries()`:

[16]:
```
bm = BlackMarble()
bm.set_countries?
```

```
Signature:
bm.set_countries(
    countries,
    ref_year=2016,
    res_km=None,
    from_hr=None,
    admin_file='admin_0_countries',
    **kwargs,
)
Docstring:
Model countries using values at reference year. If GDP or income
group not available for that year, consider the value of the closest
available year.

Parameters:
    countries (list or dict): list of country names (admin0 or subunits)
        or dict with key = admin0 name and value = [admin1 names]
    ref_year (int, optional): reference year. Default: 2016
    res_km (float, optional): approx resolution in km. Default:
        nightlights resolution.
    from_hr (bool, optional): force to use higher resolution image,
        independently of its year of acquisition.
    admin_file (str): file name, admin_0_countries or admin_0_map_subunits
    kwargs (optional): 'gdp' and 'inc_grp' dictionaries with keys the
        country ISO_alpha3 code. 'poly_val' list of polynomial coefficients
        [1,x,x^2,…] to apply to nightlight (DEF_POLY_VAL used if not
        provided). If provided, these are used.
File:      c:\users\aleciu\documents\github\climada_python\climada\entity\exposures\
↪black_marble.py
Type:      method
```

The only required parameter is: - countries: countries to model

Optional parameters are: - ref_year: year for which exposure is modeled - res_km: resolution of the exposure layer - from_hr: whether to use higher resolution (i.e. NASA data) regardless the specified year - admin_file: whether to model sovereign states or also subunits as described in https://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-admin-0-countries/ - kwargs: it allows the user to specify GDP and income group data to a given country (group of countries). These should be input as dictionaries of the form gdp = {country1_ISO3_code: gdp1, .., countryn_ISO3_code: gdpn} and inc_grp = {country1_ISO3_code: inc_grp1, .., countryn_ISO3_code: inc_grpn}

6.4 Modeling exposure of Sovereign States

As an example, let's model Switzerland in the year 2010. We need to pass the full country name. When a country name is misspelled, an error is raised with the list of possible names.

[17]:
```
# Note: execution of this cell will fail
ch_2010 = BlackMarble()
ch_2010.set_countries(['Swizerland'], ref_year=2010)
```

```
2021-02-15 10:17:19,208 - climada.entity.exposures.black_marble - ERROR - Country␣
↪Swizerland not found. Possible options: ['Indonesia', 'Malaysia', 'Chile', 'Bolivia
↪', 'Peru', 'Argentina', 'Dhekelia Sovereign Base Area', 'Cyprus', 'India', 'China',
↪'Israel', 'Palestine', 'Lebanon', 'Ethiopia', 'South Sudan', 'Somalia', 'Kenya',
↪'Pakistan', 'Malawi', 'United Republic Of Tanzania', 'Syria', 'Somaliland', 'France
```

(continues on next page)

(continued from previous page)

```
↪', 'Suriname', 'Guyana', 'South Korea', 'North Korea', 'Morocco', 'Western Sahara',
↪'Costa Rica', 'Nicaragua', 'Republic Of The Congo', 'Democratic Republic Of The␣
↪Congo', 'Bhutan', 'Ukraine', 'Belarus', 'Namibia', 'South Africa', 'Saint Martin',
↪'Sint Maarten', 'Oman', 'Uzbekistan', 'Kazakhstan', 'Tajikistan', 'Lithuania',
↪'Brazil', 'Uruguay', 'Mongolia', 'Russia', 'Czechia', 'Germany', 'Estonia', 'Latvia
↪', 'Norway', 'Sweden', 'Finland', 'Vietnam', 'Cambodia', 'Luxembourg', 'United Arab␣
↪Emirates', 'Belgium', 'Georgia', 'Macedonia', 'Albania', 'Azerbaijan', 'Kosovo',
↪'Turkey', 'Spain', 'Laos', 'Kyrgyzstan', 'Armenia', 'Denmark', 'Libya', 'Tunisia',
↪'Romania', 'Hungary', 'Slovakia', 'Poland', 'Ireland', 'United Kingdom', 'Greece',
↪'Zambia', 'Sierra Leone', 'Guinea', 'Liberia', 'Central African Republic', 'Sudan',
↪'Djibouti', 'Eritrea', 'Austria', 'Iraq', 'Italy', 'Switzerland', 'Iran',
↪'Netherlands', 'Liechtenstein', 'Ivory Coast', 'Republic Of Serbia', 'Mali',
↪'Senegal', 'Nigeria', 'Benin', 'Angola', 'Croatia', 'Slovenia', 'Qatar', 'Saudi␣
↪Arabia', 'Botswana', 'Zimbabwe', 'Bulgaria', 'Thailand', 'San Marino', 'Haiti',
↪'Dominican Republic', 'Chad', 'Kuwait', 'El Salvador', 'Guatemala', 'East Timor',
↪'Brunei', 'Monaco', 'Algeria', 'Mozambique', 'Eswatini', 'Burundi', 'Rwanda',
↪'Myanmar', 'Bangladesh', 'Andorra', 'Afghanistan', 'Montenegro', 'Bosnia And␣
↪Herzegovina', 'Uganda', 'Us Naval Base Guantanamo Bay', 'Cuba', 'Honduras', 'Ecuador
↪', 'Colombia', 'Paraguay', 'Portugal', 'Moldova', 'Turkmenistan', 'Jordan', 'Nepal',
↪ 'Lesotho', 'Cameroon', 'Gabon', 'Niger', 'Burkina Faso', 'Togo', 'Ghana', 'Guinea-
↪Bissau', 'Gibraltar', 'United States Of America', 'Canada', 'Mexico', 'Belize',
↪'Panama', 'Venezuela', 'Papua New Guinea', 'Egypt', 'Yemen', 'Mauritania',
↪'Equatorial Guinea', 'Gambia', 'Hong Kong S.A.R.', 'Vatican', 'Northern Cyprus',
↪'Cyprus No Mans Area', 'Siachen Glacier', 'Baykonur Cosmodrome', 'Akrotiri␣
↪Sovereign Base Area', 'Antarctica', 'Australia', 'Greenland', 'Fiji', 'New Zealand',
↪ 'New Caledonia', 'Madagascar', 'Philippines', 'Sri Lanka', 'Curaã§Ao', 'Aruba',
↪'The Bahamas', 'Turks And Caicos Islands', 'Taiwan', 'Japan', 'Saint Pierre And␣
↪Miquelon', 'Iceland', 'Pitcairn Islands', 'French Polynesia', 'French Southern And␣
↪Antarctic Lands', 'Seychelles', 'Kiribati', 'Marshall Islands', 'Trinidad And Tobago
↪', 'Grenada', 'Saint Vincent And The Grenadines', 'Barbados', 'Saint Lucia',
↪'Dominica', 'United States Minor Outlying Islands', 'Montserrat', 'Antigua And␣
↪Barbuda', 'Saint Kitts And Nevis', 'United States Virgin Islands', 'Saint Barthelemy
↪', 'Puerto Rico', 'Anguilla', 'British Virgin Islands', 'Jamaica', 'Cayman Islands',
↪ 'Bermuda', 'Heard Island And Mcdonald Islands', 'Saint Helena', 'Mauritius',
↪'Comoros', 'Sã£O Tomã© And Principe', 'Cabo Verde', 'Malta', 'Jersey', 'Guernsey',
↪'Isle Of Man', 'Aland', 'Faroe Islands', 'Indian Ocean Territories', 'British␣
↪Indian Ocean Territory', 'Singapore', 'Norfolk Island', 'Cook Islands', 'Tonga',
↪'Wallis And Futuna', 'Samoa', 'Solomon Islands', 'Tuvalu', 'Maldives', 'Nauru',
↪'Federated States Of Micronesia', 'South Georgia And The Islands', 'Falkland Islands
↪', 'Vanuatu', 'Niue', 'American Samoa', 'Palau', 'Guam', 'Northern Mariana Islands',
↪ 'Bahrain', 'Coral Sea Islands', 'Spratly Islands', 'Clipperton Island', 'Macao S.A.
↪R', 'Ashmore And Cartier Islands', 'Bajo Nuevo Bank (Petrel Is.)', 'Serranilla Bank
↪', 'Scarborough Reef']
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-983cb625fccf> in <module>
      1 ch_2010 = BlackMarble()
----> 2 ch_2010.set_countries(['Swizerland'], ref_year=2010)

~\Documents\GitHub\climada_python\climada\entity\exposures\black_marble.py in set_
↪countries(self, countries, ref_year, res_km, from_hr, admin_file, **kwargs)
     95
---> 97                                                  admin_key_dict[admin_
↪file])
     96         cntry_info, cntry_admin1 = country_iso_geom(countries, shp_file,
     98         fill_econ_indicators(ref_year, cntry_info, shp_file, **kwargs)
```

(continues on next page)

(continued from previous page)

```
     99

~\Documents\GitHub\climada_python\climada\entity\exposures\black_marble.py in country_
↪iso_geom(countries, shp_file, admin_key)
    221             LOGGER.error('Country %s not found. Possible options: %s',
    222                          country_name, options)
--> 223             raise ValueError
    224         iso3 = list_records[country_idx].attributes[admin_key[1]]
    225         try:

ValueError:
```

[18]:
```
ch_2010 = BlackMarble()
ch_2010.set_countries(['Switzerland'], ref_year=2010)
```

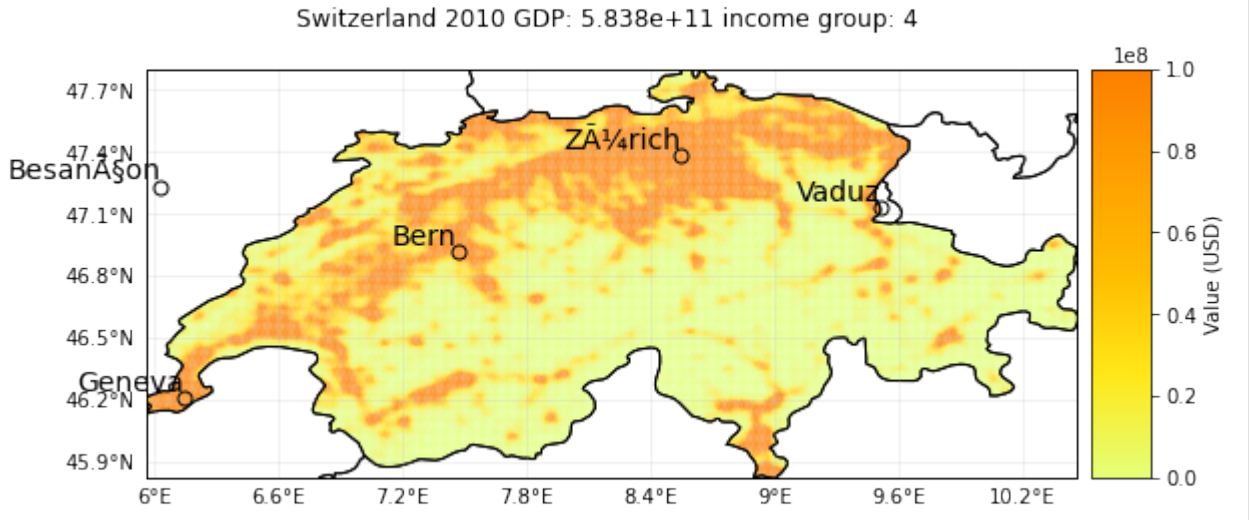
```
2021-02-15 10:17:53,670 - climada.util.finance - INFO - GDP CHE 2010: 5.838e+11.
2021-02-15 10:17:53,765 - climada.util.finance - INFO - Income group CHE 2010: 4.
2021-02-15 10:17:53,765 - climada.entity.exposures.black_marble - INFO - Nightlights␣
↪from NOAA's earth observation group for year 2010.
2021-02-15 10:17:54,436 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Switzerland.
2021-02-15 10:17:54,600 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 1.0 km.
```

[19]:
```
ch_2010.plot_hexbin(vmax=1e8);
```

```
C:\Users\aleciu\Documents\GitHub\climada_python\climada\util\plot.py:314: UserWarning:
↪ Tight layout not applied. The left and right margins cannot be made large enough␣
↪to accommodate all axes decorations.
  fig.tight_layout()
```

[19]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x21408998408>
```

```
C:\Users\aleciu\Anaconda3\envs\climada_env\lib\site-packages\cartopy\mpl\feature_
↪artist.py:225: MatplotlibDeprecationWarning: Using a string of single character␣
↪colors as a color sequence is deprecated. Use an explicit list instead.
  **dict(style))
```



From the log we can see that the resolution of the created exposure is 1 km. We can change that:

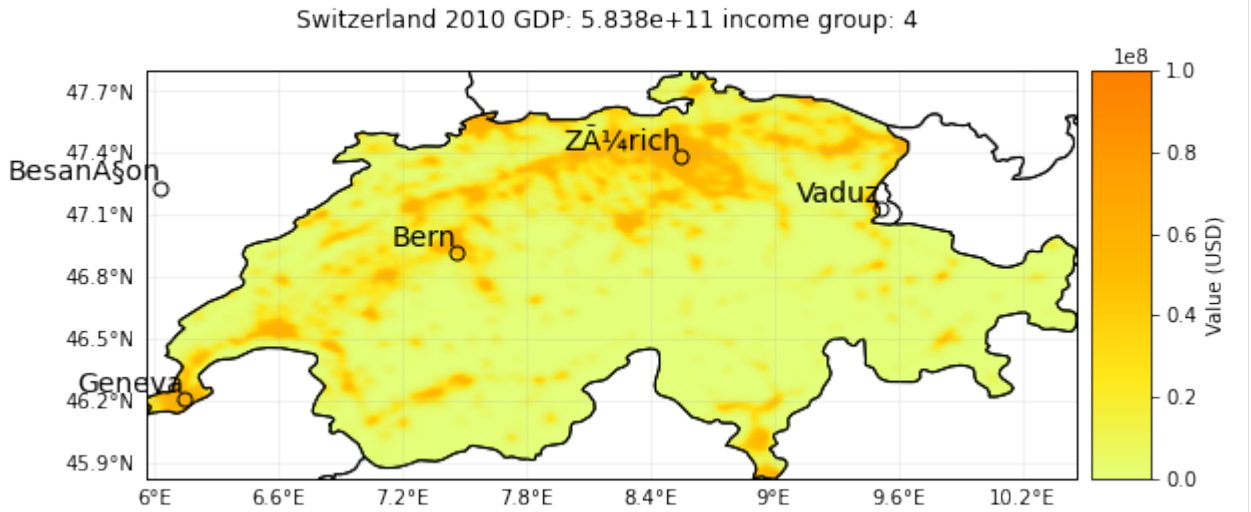
[20]:
```
ch_2010_05 = BlackMarble()
ch_2010_05.set_countries(['Switzerland'], ref_year=2010, res_km=0.5)
ch_2010_05.plot_hexbin(vmax=1e8);
```

```
2021-02-15 10:18:08,821 - climada.util.finance - INFO - GDP CHE 2010: 5.838e+11.
2021-02-15 10:18:08,932 - climada.util.finance - INFO - Income group CHE 2010: 4.
2021-02-15 10:18:08,934 - climada.entity.exposures.black_marble - INFO - Nightlights␣
↪from NOAA's earth observation group for year 2010.
2021-02-15 10:18:09,216 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Switzerland.
2021-02-15 10:18:09,346 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 0.5 km.
```

```
C:\Users\aleciu\Documents\GitHub\climada_python\climada\util\plot.py:314: UserWarning:
↪ Tight layout not applied. The left and right margins cannot be made large enough␣
↪to accommodate all axes decorations.
  fig.tight_layout()
```

[20]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x2144f7bcfc8>
```

```
C:\Users\aleciu\Anaconda3\envs\climada_env\lib\site-packages\cartopy\mpl\feature_
↪artist.py:225: MatplotlibDeprecationWarning: Using a string of single character␣
↪colors as a color sequence is deprecated. Use an explicit list instead.
  **dict(style))
```



Let's now see what happens when the flag *'from_hr'* is set to True:

[21]:
```
ch_2010_hr = BlackMarble()
ch_2010_hr.set_countries(['Switzerland'], ref_year=2010, from_hr=True)
ch_2010_hr.plot_hexbin(vmax=1e8);
```

```
2021-02-15 10:18:29,714 - climada.util.finance - INFO - GDP CHE 2010: 5.838e+11.
2021-02-15 10:18:29,854 - climada.util.finance - INFO - Income group CHE 2010: 4.
2021-02-15 10:18:29,856 - climada.entity.exposures.black_marble - INFO - Nightlights␣
↪from NASA's earth observatory for year 2012.
2021-02-15 10:18:45,264 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Switzerland.
2021-02-15 10:18:45,930 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 0.5 km.
```

```
C:\Users\aleciu\Documents\GitHub\climada_python\climada\util\plot.py:314: UserWarning:
↪ Tight layout not applied. The left and right margins cannot be made large enough␣
```

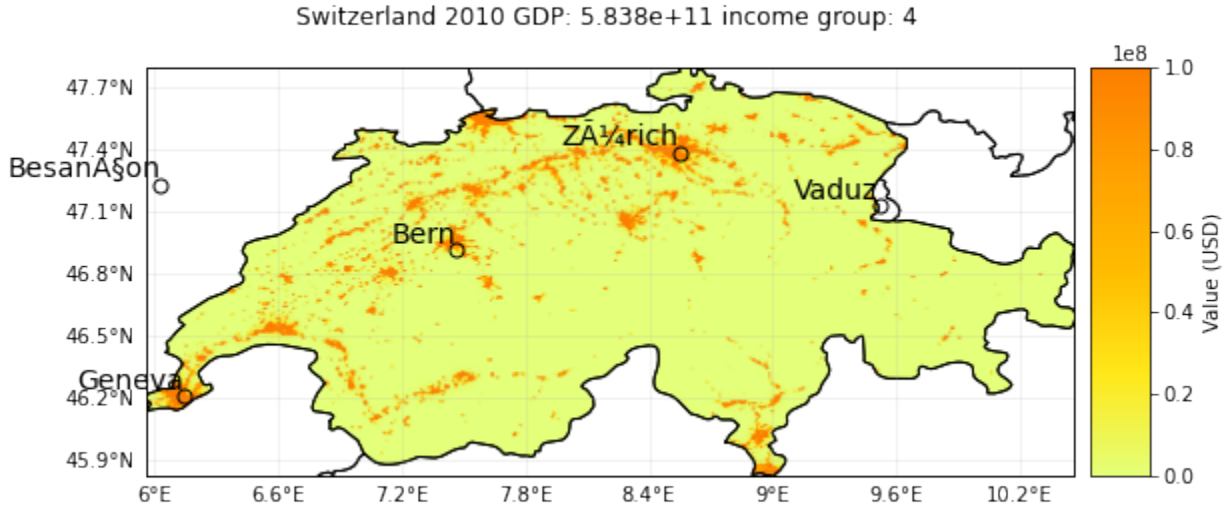
(continues on next page)

(continued from previous page)

```
↪to accommodate all axes decorations.
  fig.tight_layout()
```

[21]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x2140c719348>
```

```
C:\Users\aleciu\Anaconda3\envs\climada_env\lib\site-packages\cartopy\mpl\feature_
↪artist.py:225: MatplotlibDeprecationWarning: Using a string of single character␣
↪colors as a color sequence is deprecated. Use an explicit list instead.
  **dict(style))
```



From the log of the last two generated exposures, one reads that both are generated at a resolution of 0.5 km and that GDP for both refers to the year 2010 (as required by the relative keyword argument). However, it is also clear that for the second exposure (i.e. with the *'from_hr'* flag set to true) a NASA image for the year 2012 is used, which is the high resolution image for the year closest to the reference year (i.e. 2010).

To model more than one country, simply pass more countries' names:

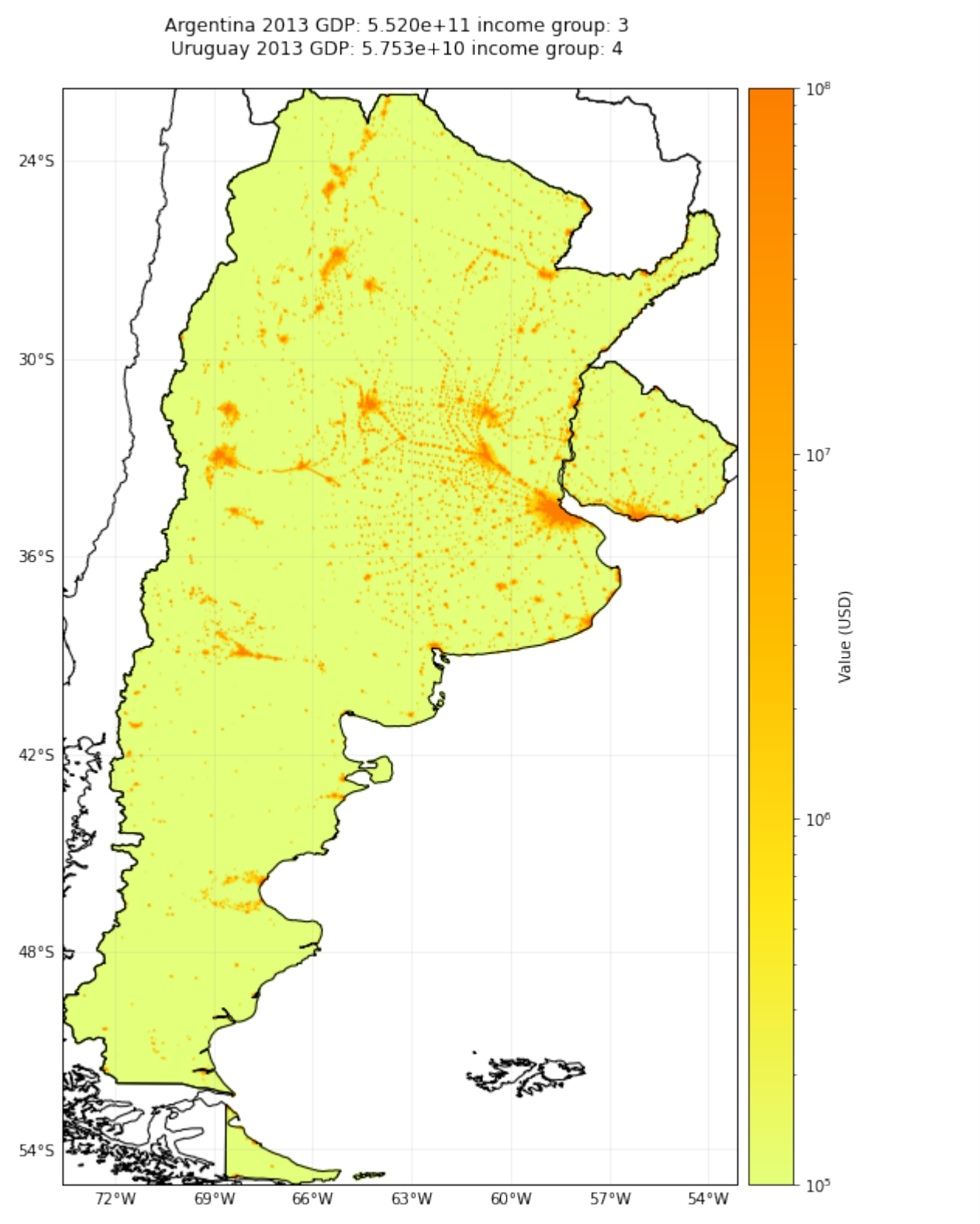
[26]:
```
from matplotlib import colors

au = BlackMarble()
au.set_countries(['Argentina', 'Uruguay'], 2013, res_km=1.0)
norm=colors.LogNorm(vmin=1.0e5, vmax=1.0e8)
au.plot_hexbin(pop_name=False, norm=norm);
```

```
2021-02-15 10:36:27,233 - climada.util.finance - INFO - GDP ARG 2013: 5.520e+11.
2021-02-15 10:36:27,336 - climada.util.finance - INFO - Income group ARG 2013: 3.
2021-02-15 10:36:27,797 - climada.util.finance - INFO - GDP URY 2013: 5.753e+10.
2021-02-15 10:36:27,897 - climada.util.finance - INFO - Income group URY 2013: 4.
2021-02-15 10:36:27,899 - climada.entity.exposures.black_marble - INFO - Nightlights␣
↪from NOAA's earth observation group for year 2013.
2021-02-15 10:36:28,149 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Argentina.
2021-02-15 10:36:36,706 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 1.0 km.
2021-02-15 10:36:37,557 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Uruguay.
2021-02-15 10:36:37,895 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 1.0 km.
```

```
C:\Users\aleciu\Documents\GitHub\climada_python\climada\util\plot.py:314: UserWarning:
↪ Tight layout not applied. The left and right margins cannot be made large enough␣
↪to accommodate all axes decorations.
  fig.tight_layout()
```

[26]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x21408a42148>
```

```
C:\Users\aleciu\Anaconda3\envs\climada_env\lib\site-packages\cartopy\mpl\feature_
↪artist.py:225: MatplotlibDeprecationWarning: Using a string of single character␣
↪colors as a color sequence is deprecated. Use an explicit list instead.
  **dict(style))
```

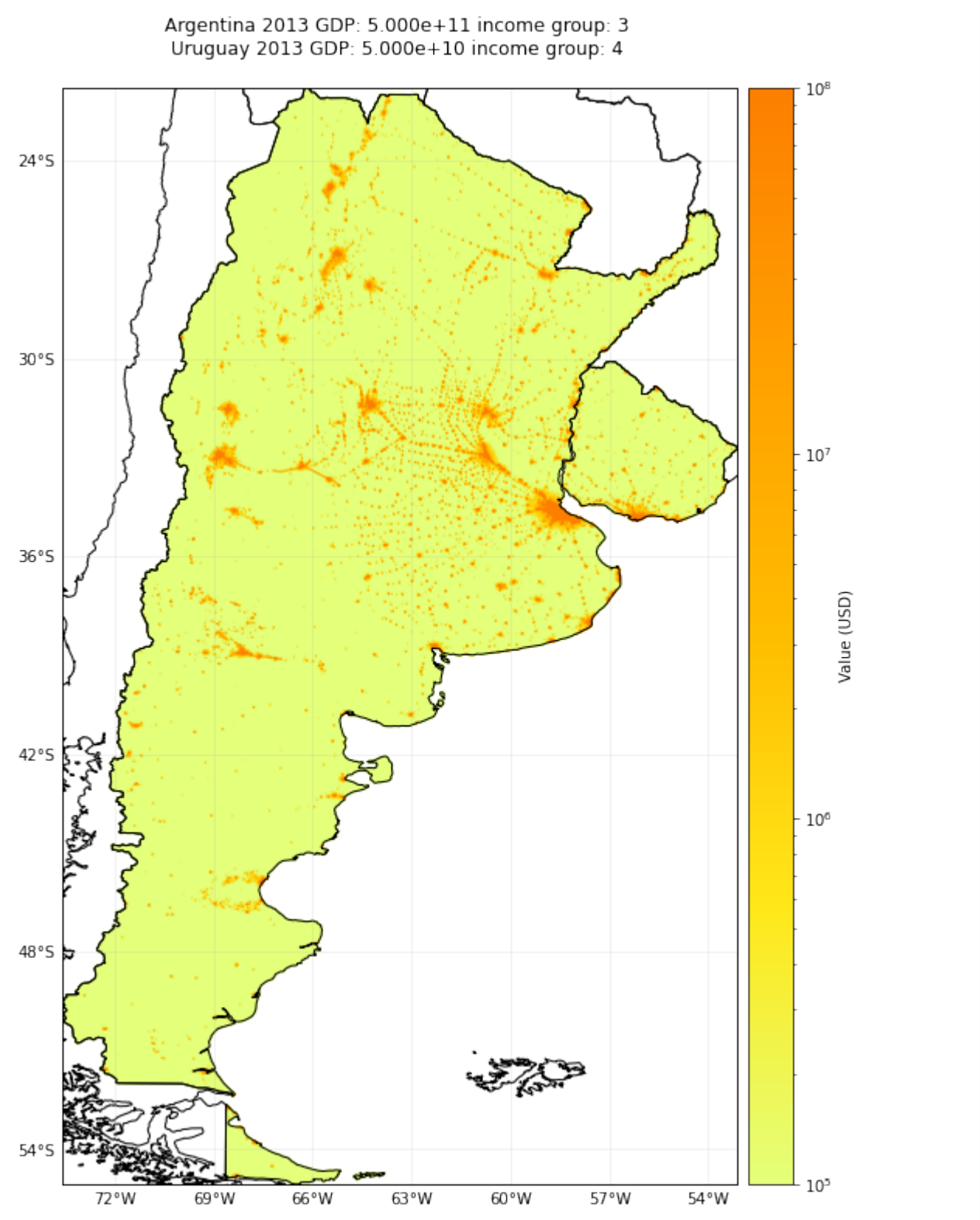


The user can apply her own gdp and income group values by passing a dictionary called *gdp*, with keys the countries iso-codes and values gdp values, and one called *inc_grp*, with keys the countries iso-codes and values income groups (from 1 to 4).

[27]:
```
au = BlackMarble()
au.set_countries(['Argentina', 'Uruguay'], 2013, res_km=1.0, gdp={'ARG': 5e11, 'URY':␣
↪5e10}, inc_grp={'ARG': 3, 'URY': 4})
norm=colors.LogNorm(vmin=1.0e5, vmax=1.0e8)
au.plot_hexbin(pop_name=False, norm=norm);
```

```
2021-02-15 10:38:04,507 - climada.entity.exposures.black_marble - INFO - Nightlights␣
↪from NOAA's earth observation group for year 2013.
2021-02-15 10:38:04,750 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Argentina.
2021-02-15 10:38:13,399 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 1.0 km.
2021-02-15 10:38:14,302 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Uruguay.
2021-02-15 10:38:14,649 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 1.0 km.
```

[27]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x21450396f48>
```


Argentina 2013 GDP: 5.000e+11 income group: 3
Uruguay 2013 GDP: 5.000e+10 income group: 4
24°S
30°S
36°S
42°S
48°S
54°S
72°W
69°W
66°W
63°W
60°W
57°W
54°W
10^8
10^7
10^6
10^5
Value (USD)

6.5 Model specific territories of Sovereign States

Regions, provinces, autonomous territories can also be modelled. For example, in order to model first-order administrative boundaries of Germany and Czech Republic, one can do:

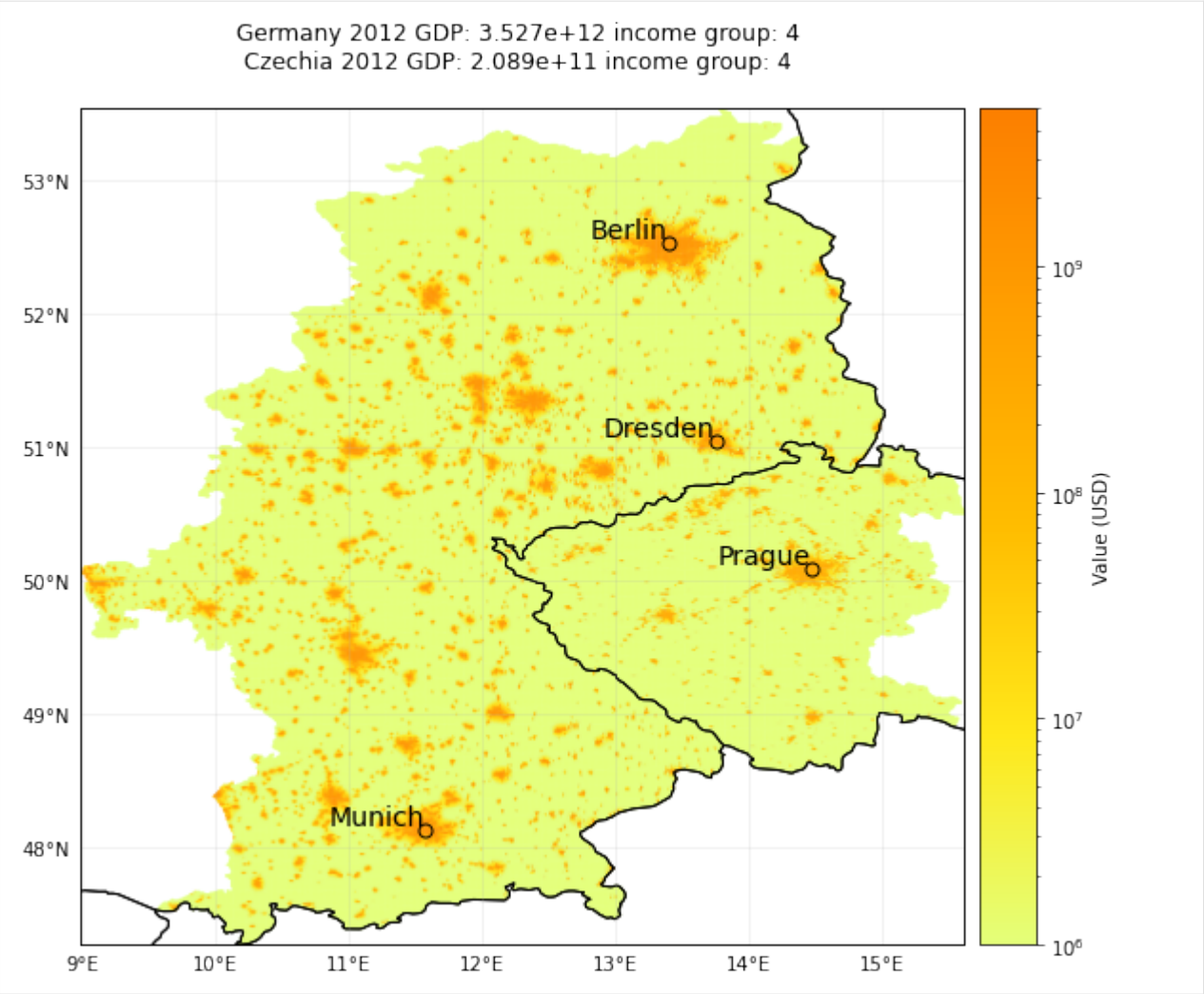
[28]:
```
country_name = {'Germany': ['Berlin', 'Brandenburg', 'Bayern', 'Sachsen', 'Thüringen',
↪ 'Sachsen-Anhalt'],
                'Czechia': ['Prague', 'Karlovarský', 'Ústecký', 'Liberecký',
↪'Středočeský', 'Plzeňský', 'Jihomoravský']}

ent = BlackMarble()
ent.set_countries(country_name, 2012, res_km=1.0, from_hr=True)

norm=colors.LogNorm(vmin=1.0e6, vmax=5.0e9)
ent.plot_hexbin(norm=norm);
```

```
2021-02-15 10:39:42,488 - climada.util.finance - INFO - GDP DEU 2012: 3.527e+12.
2021-02-15 10:39:42,580 - climada.util.finance - INFO - Income group DEU 2012: 4.
2021-02-15 10:39:43,080 - climada.util.finance - INFO - GDP CZE 2012: 2.089e+11.
2021-02-15 10:39:43,174 - climada.util.finance - INFO - Income group CZE 2012: 4.
2021-02-15 10:39:43,174 - climada.entity.exposures.black_marble - INFO - Nightlights␣
↪from NASA's earth observatory for year 2012.
2021-02-15 10:40:00,515 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Germany.
2021-02-15 10:40:04,810 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 1.0 km.
2021-02-15 10:40:05,645 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Czechia.
2021-02-15 10:40:06,613 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 1.0 km.
```

```
C:\Users\aleciu\Documents\GitHub\climada_python\climada\util\plot.py:314: UserWarning:
↪ Tight layout not applied. The left and right margins cannot be made large enough␣
↪to accommodate all axes decorations.
  fig.tight_layout()
```

[28]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x2140f317188>
```

```
C:\Users\aleciu\Anaconda3\envs\climada_env\lib\site-packages\cartopy\mpl\feature_
↪artist.py:225: MatplotlibDeprecationWarning: Using a string of single character␣
↪colors as a color sequence is deprecated. Use an explicit list instead.
  **dict(style))
```



As above, if the region's name is misspelled, a list with all possible regions' names for that country is provided:

[29]:
```
# Note: execution of this cell will fail
ent = BlackMarble()
ent.set_countries({'Germany': ['']}, 2012, res_km=1.0, from_hr=True)
```

```
2021-02-15 10:40:31,404 - climada.entity.exposures.black_marble - ERROR -  not found.␣
↪Possible provinces of DEU are: ['Sachsen', 'Bayern', 'Rheinland-Pfalz', 'Saarland',
↪'Schleswig-Holstein', 'Niedersachsen', 'Nordrhein-Westfalen', 'Baden-WÃ¼rttemberg',
↪'Brandenburg', 'Mecklenburg-Vorpommern', 'Bremen', 'Hamburg', 'Hessen', 'ThÃ¼ringen
↪', 'Sachsen-Anhalt', 'Berlin']
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-29-9e7340f1afa7> in <module>
      1 ent = BlackMarble()
----> 2 ent.set_countries({'Germany': ['']}, 2012, res_km=1.0, from_hr=True)

~\Documents\GitHub\climada_python\climada\entity\exposures\black_marble.py in set_
↪countries(self, countries, ref_year, res_km, from_hr, admin_file, **kwargs)
     95
---> 97                                                admin_key_dict[admin_
```

(continues on next page)

(continued from previous page)

```
↪file])
     96         cntry_info, cntry_admin1 = country_iso_geom(countries, shp_file,
     98         fill_econ_indicators(ref_year, cntry_info, shp_file, **kwargs)
     99

~\Documents\GitHub\climada_python\climada\entity\exposures\black_marble.py in country_
↪iso_geom(countries, shp_file, admin_key)
    229         cntry_info[iso3] = [cntry_id, country_name.title(),
    230                             list_records[country_idx].geometry]
--> 231         cntry_admin1[iso3] = _fill_admin1_geom(iso3, admin1_rec, prov_list)
    232
    233     return cntry_info, cntry_admin1

~\Documents\GitHub\climada_python\climada\entity\exposures\black_marble.py in _fill_
↪admin1_geom(iso3, admin1_rec, prov_list)
    343             LOGGER.error('%s not found. Possible provinces of %s are: %s',
    344                          prov, iso3, options)
--> 345             raise ValueError
    346
    347     return prov_geom

ValueError:
```

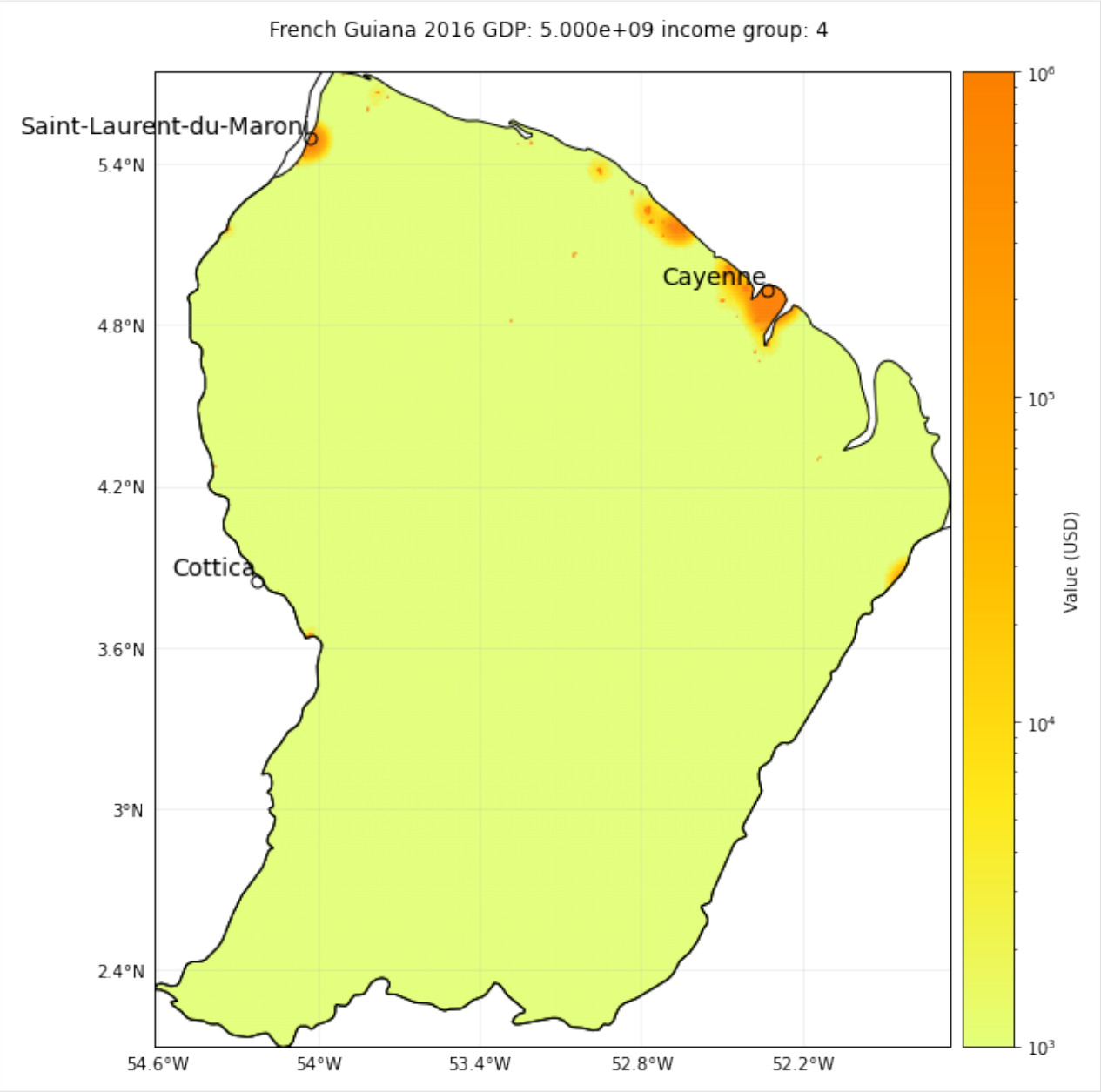
One can also model countries' subunits. This requires setting the admin_file argument to *'admin_0_map_subunits'*. Note that most likely in these cases both the gdp and inc_grp dicts must be passed.

[51]:
```
fg = BlackMarble()
fg.set_countries(countries=['French Guiana'], gdp={'GUF': 5*1e9}, inc_grp={'GUF': 4},␣
↪admin_file='admin_0_map_subunits')
```

```
2021-02-15 10:58:44,618 - climada.entity.exposures.black_marble - INFO - Nightlights␣
↪from NASA's earth observatory for year 2016.
2021-02-15 10:58:58,119 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country French Guiana.
2021-02-15 10:58:58,698 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 0.5 km.
```

[54]:
```
norm=colors.LogNorm(vmin=1.0e3, vmax=1.0e6)
fg.plot_hexbin(norm=norm);
```

[54]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x21481092488>
```



6.6 Change exponents of the polynomial transformation used for nighlight intensity:

One can also change the exponents of the polynomial transformation used in the nightlight intensity through the `poly_val` parameter. The default transformation is x^2 (i.e. poly_val = [0, 0, 1])

[32]:
```
ch_pol_val = BlackMarble()
ch_pol_val.set_countries(['Switzerland'])
ch_pol_val.plot_hexbin(vmax=1e8)

ch_pol_val1 = BlackMarble()
```

(continues on next page)

(continued from previous page)

```
ch_pol_val1.set_countries(['Switzerland'], poly_val=[0, 0, 0, 0, 1])
ch_pol_val1.plot_hexbin(vmax=1e8);
```

```
2021-02-15 10:46:49,884 - climada.util.finance - INFO - GDP CHE 2016: 6.713e+11.
2021-02-15 10:46:49,986 - climada.util.finance - INFO - Income group CHE 2016: 4.
2021-02-15 10:46:49,986 - climada.entity.exposures.black_marble - INFO - Nightlights␣
↪from NASA's earth observatory for year 2016.
2021-02-15 10:47:06,318 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Switzerland.
2021-02-15 10:47:06,922 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 0.5 km.
```

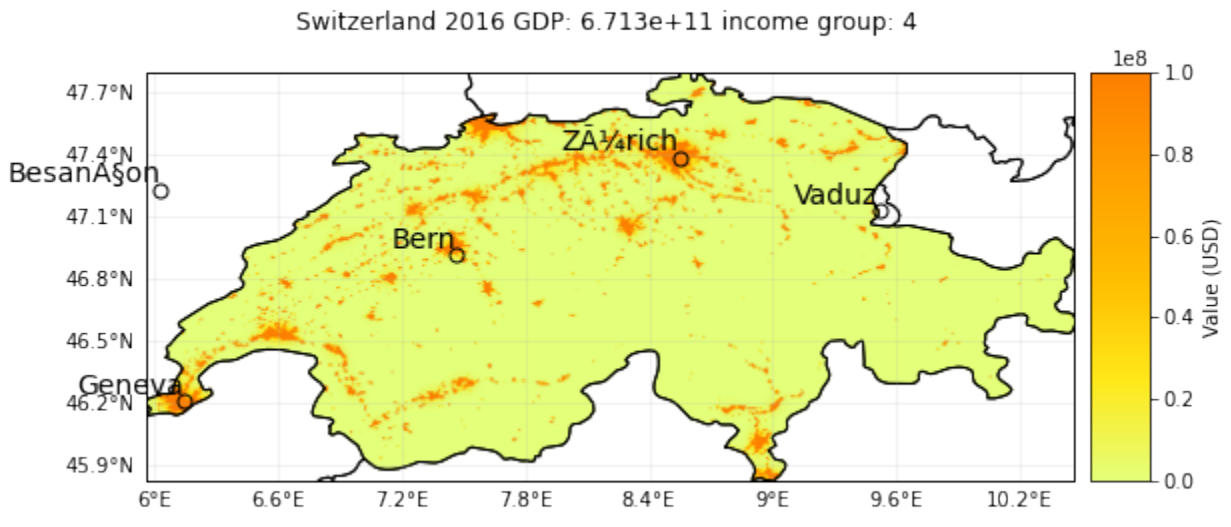
```
C:\Users\aleciu\Documents\GitHub\climada_python\climada\util\plot.py:314: UserWarning:
↪ Tight layout not applied. The left and right margins cannot be made large enough␣
↪to accommodate all axes decorations.
  fig.tight_layout()
```

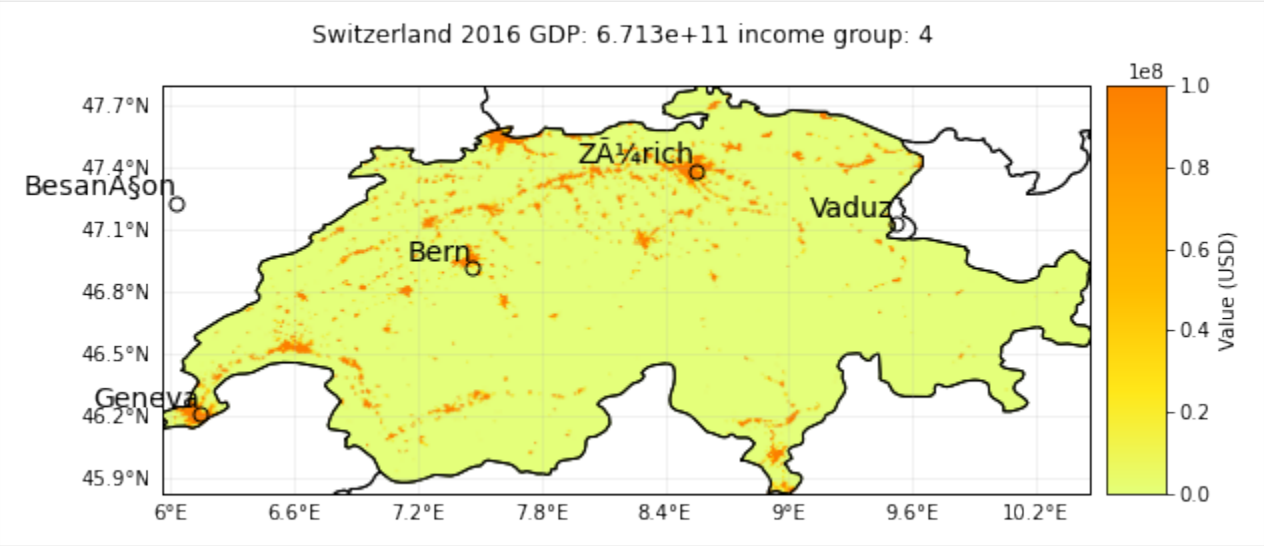
```
2021-02-15 10:47:24,209 - climada.util.finance - INFO - GDP CHE 2016: 6.713e+11.
2021-02-15 10:47:24,375 - climada.util.finance - INFO - Income group CHE 2016: 4.
2021-02-15 10:47:24,375 - climada.entity.exposures.black_marble - INFO - Nightlights␣
↪from NASA's earth observatory for year 2016.
2021-02-15 10:47:43,090 - climada.entity.exposures.black_marble - INFO - Processing␣
↪country Switzerland.
2021-02-15 10:47:43,737 - climada.entity.exposures.black_marble - INFO - Generating␣
↪resolution of approx 0.5 km.
```

```
C:\Users\aleciu\Documents\GitHub\climada_python\climada\util\plot.py:314: UserWarning:
↪ Tight layout not applied. The left and right margins cannot be made large enough␣
↪to accommodate all axes decorations.
  fig.tight_layout()
```

[32]:
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x2148f8f2088>
```

```
C:\Users\aleciu\Anaconda3\envs\climada_env\lib\site-packages\cartopy\mpl\feature_
↪artist.py:225: MatplotlibDeprecationWarning: Using a string of single character␣
↪colors as a color sequence is deprecated. Use an explicit list instead.
  **dict(style))
```




Switzerland 2016 GDP: 6.713e+11 income group: 4
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Value (USD)
47.7°N
47.4°N
47.1°N
46.8°N
46.5°N
46.2°N
45.9°N
6°E
6.6°E
7.2°E
7.8°E
8.4°E
9°E
9.6°E
10.2°E
BesanÃ§on
ZÃ¼rich
Bern
Vaduz
Geneva

CHAPTER

SEVEN

CLIMADA & OPENSTREETMAP

CLIMADA provides some ways to make use of the entire OpenStreetMap data and to use those data within the risk modelling chain of CLIMADA as exposures. This tutorial will walk you through how to get any OSM data into tabular format, and how to perform impact / risk computations on this data.

Recommended reading for OSM and risk assessments: *Mühlhofer, Kropf, Riedel, Bresch and Koks (under rev.): OpenStreetMap for Multi-Faceted Climate Risk Assessments. Environmental Research Communications. doi:* 10.31223/X5SQ2J

7.1 Outline

- Part 0: **Introduction & Overview on OSM data**
- Part 1: **Downloading, clipping and parsing OSM data**
  - 1.1: Downloading, clipping and parsing OSM data from OSM data dumps
  - 1.2 Downloading data directly froom the overpass-api.
- Part 2: **Risk computation with OSM data as CLIMADA Exposures**
  - 2.1 Setting up a high-resolution Exposures instance using OSM data
  - 2.2 Using OSM data as high-resolution stencil for LitPop Exposres - *under development*
  - 2.3 Performing risk computations with high-res Exposures

Part 0: Introduction & Overview on OSM data

Which methods exist for getting data and which one is right for my query?

OSM data can be obtained in geodataframe format along two pathways

Option 1: first download *all* map raw data witin a certain geographic area that is on OSM, in the OSM-specific data format (osm.pbf). Then, an SQL query extracts only the info that is wanted and loads it into memory. One may either downnload country files (from a daily-updated provider, Geofabrik.de), or a larger region or even the entire planet file once, and then clip sub-files with raw data for any desired custom-made region from these larger parent file. The query/extraction process afterwards is the same.

Option 2: Directly and selectively download the desired info from the overpass-turbo API, without downloading the complete raw data dumps beforehand. This is however heavily constrained in terms of download quota and requires stable internet connection for each download process.

The first option of downloading data dumps and then parsing from them is implemented in the external package `osm-flex`, which is installed within the CLIMADA coding environment. The second option is implemented within CLIMADA, as `OSMApiQuery` - methods in the `Exposures.osm_dataloader` module.

A few words on the OSM data world, OSM overpass-turbo, Overpass QL & OSM data structures (nodes, ways, relations)

OSM tags, key-value pairs, and quick overview on "what's out there" Check out https://taginfo.openstreetmap.org/ for finding the key-value pairs ("tags") you're looking for. You will need to know this for specifying your query (at least in the API-pathway). Check out https://overpass-turbo.eu/ for a fast visual overview on results that your query will yield.

The OSM API (overpass): The OSM API has constraints on how much can be downloaded at once (reached quite fast, especially around mid-day / early afternoon..). This is why providers (such as Geofabrik) exist that generate data daily data dumps of all the OSM data that exists. For the API downloading strategy, wait-times are implemented in the querying, but re-consider your strategy if you run into a time-out error. Overpass Query language, which is needed to query directly from the OSM API can be a bit cryptic. The majority of what you will need is transformed in the `OSMApiQuery` class automatically. For more, check out the detailed read-the-docs: https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL

OSM data structures: Elements are the basic components of OpenStreetMap's conceptual data model of the physical world. Elements are of three types:

- nodes (defining points in space),
- ways (defining linear features and area boundaries), and
- relations (which are sometimes used to explain how other elements work together).

Check out https://wiki.openstreetmap.org/wiki/Elements for more info.

Part 1: Downloading, clipping and parsing OSM data

1.1: Getting OSM data from OSM data dumps (*osm.pbf files)

In this example, we download the data dump for a country (Honduras) from Geofabrik. In the second step, specific info can be extracted (parsed) from the raw files into tabular format.

NOTE: This is only a minimal example, as downloading, clipping and parsing is performed with the `osm-flex` package, which has extensive documentation that can be found here.

[17]:
```
import matplotlib.pyplot as plt
import shapely
import contextily as ctx
import os

import osm_flex
import osm_flex.download
import osm_flex.extract
import osm_flex.clip
import osm_flex.simplify

from climada import CONFIG
osm_flex.enable_logs()
```

Download a raw osm.pbf file ("data dump") for Honduras from geofabrik.de

[4]:
```
# (checks if file honduras-latest.osm.pbf already exists)
# file is stored as defined in osm_flex.config.OSM_DATA_DIR unless specified otherwise
iso3 = 'HND'
path_hnd_dump = osm_flex.download.get_country_geofabrik(iso3)
```

```
INFO:osm_flex.download:Skip existing file: /Users/evelynm/osm/osm_bpf/honduras-latest.
↪osm.pbf
```

Extracting pre-written query classes from the data dump

Extracting critical infrastructure with pre-written queries: For critical infrastructure, a set of wrappers exist that parse all data belonging to this sector.

[5]:
```
# check available critical infrastructure types:
osm_flex.config.DICT_CIS_OSM.keys()
```

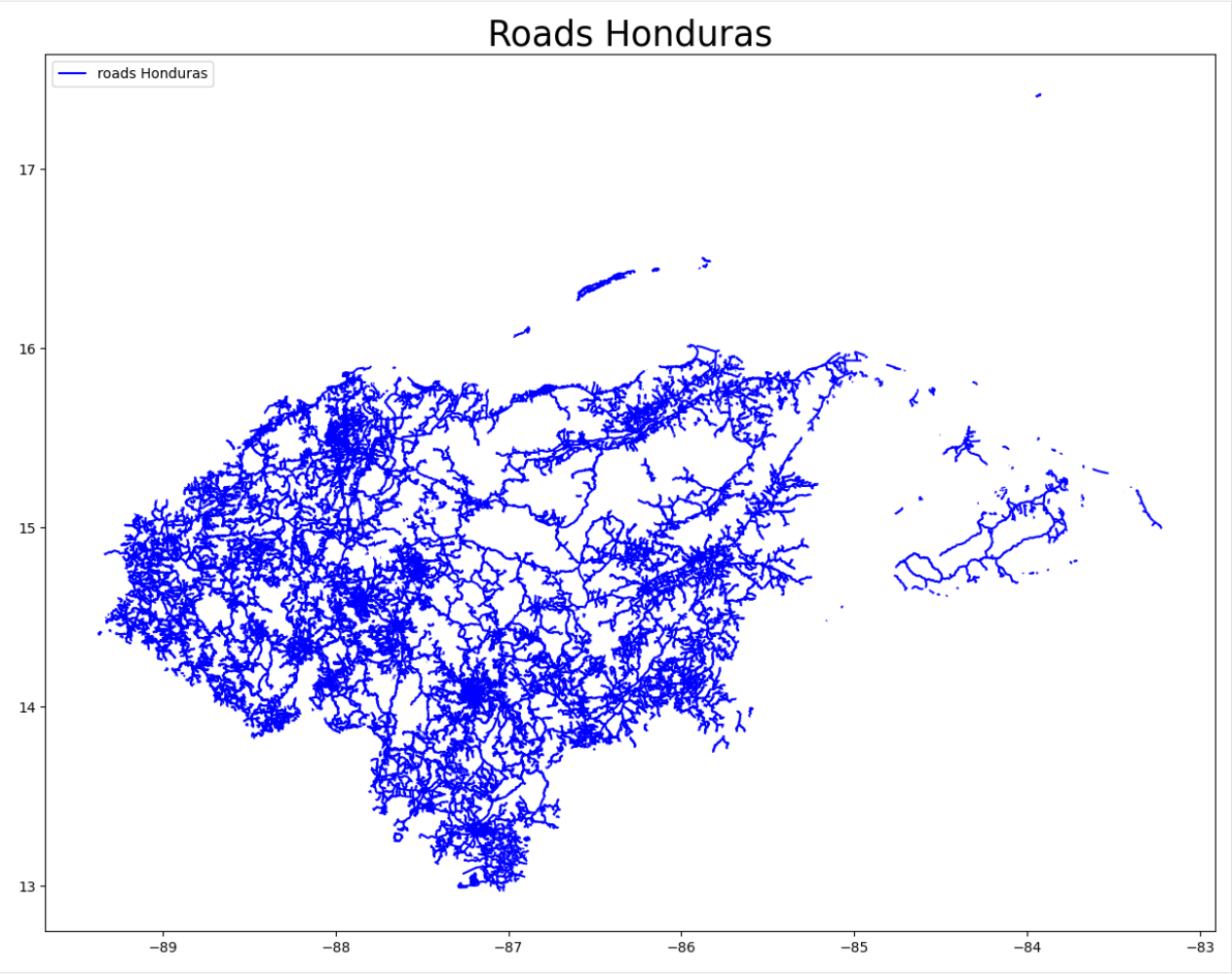
[5]:
```
dict_keys(['education', 'healthcare', 'water', 'telecom', 'road', 'main_road', 'rail',
↪ 'air', 'gas', 'oil', 'power', 'wastewater', 'food', 'buildings'])
```

[6]:
```
# lets extract all roads from the Honduras file, via the wrapper
gdf_roads_hnd = osm_flex.extract.extract_cis(
    path_hnd_dump,
    'road')
```

```
INFO:osm_flex.extract:query is finished, lets start the loop
extract points: 0it [00:00, ?it/s]
INFO:osm_flex.extract:query is finished, lets start the loop
extract multipolygons: 100
↪%|████████████████████████████████████████████████████████████████████████████████
↪2/2 [00:05<00:00,  2.56s/it]
INFO:osm_flex.extract:query is finished, lets start the loop
extract lines: 100
↪%|████████████████████████████████████████████████████████████████████████████████
↪112215/112215 [00:07<00:00, 14463.46it/s]
```

[10]:
```
# plot results
ax = gdf_roads_hnd.plot(figsize=(15, 15), alpha=1, markersize=5, color='blue',
                        edgecolor='blue', label='roads Honduras')
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles=handles, loc='upper left')
ax.set_title('Roads Honduras', fontsize=25)
plt.show()
```

```
/var/folders/jm/4przql117d9gb9g4hb45fd4c0000gp/T/ipykernel_4655/3520574044.py:4:␣
↪UserWarning: Legend does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html
↪#implementing-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
```



Extracting any custom-made query from the data dump

Any other tags that are available on openstreetmap can also be queried with the `extract()` method. Two variables have to be specified: `osm_keys`, a list with all the columns to report in the gdf, and `osm_query`, a string of key-value constraints to apply during the search.

[8]:
```
# Forests (multipolygons) in Honduras
osm_keys = ['landuse']
osm_query = "landuse='forest'"
gdf_forest = osm_flex.extract.extract(path_hnd_dump, 'multipolygons', osm_keys, osm_
↪query)
```

```
INFO:osm_flex.extract:query is finished, lets start the loop
extract multipolygons: 100
↪%|
↪709/709 [00:04<00:00, 144.00it/s]
```

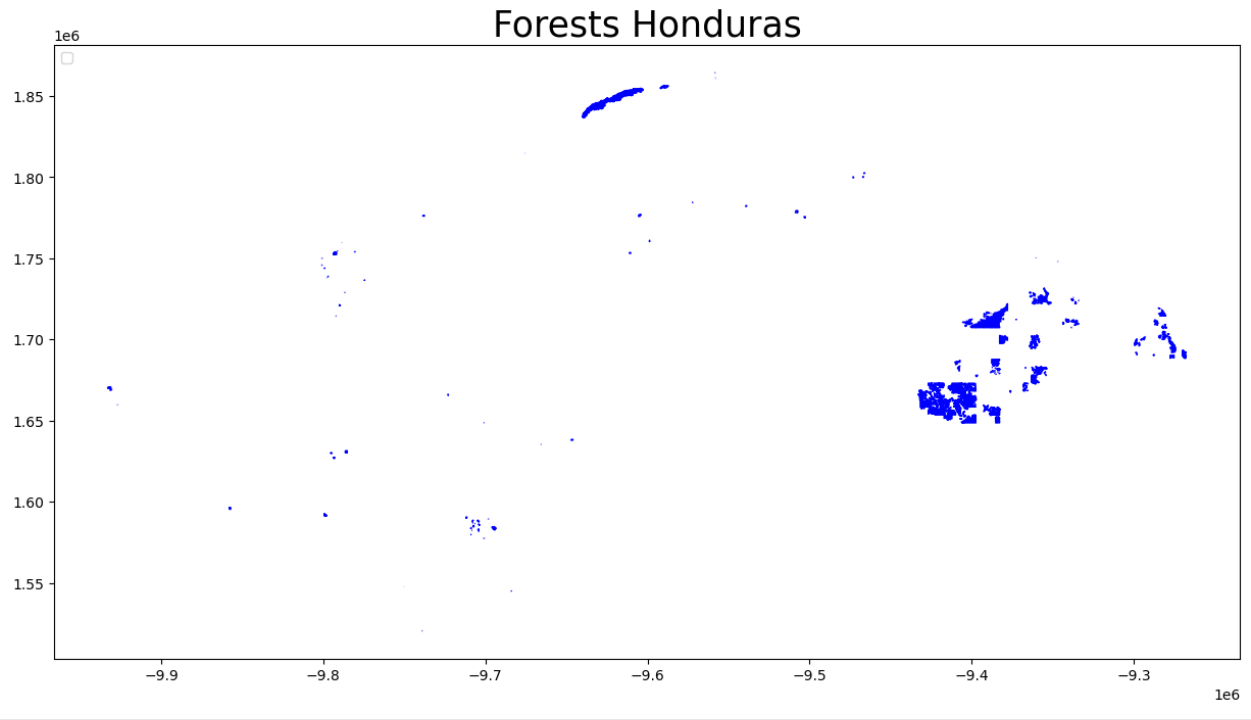
[11]:
```
# Plot results
ax = gdf_forest.to_crs(epsg=3857).plot(figsize=(15, 15), alpha=1, markersize=5, color=
↪'blue',
                       edgecolor='blue', label='forests HND')
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles=handles, loc='upper left')
```

(continues on next page)

(continued from previous page)

```
ax.set_title('Forests Honduras', fontsize=25)
plt.show()
```

```
/var/folders/jm/4przql117d9gb9g4hb45fd4c0000gp/T/ipykernel_4655/3095862923.py:4:␣
↪UserWarning: Legend does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html
↪#implementing-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
```



Extracting data from a custom-clipped data dump

Instead of retrieving all the applicable data within a country raw data file, we can cut out (=clip) any desired shape or bounding box from the entire planet file (has to be downloaded first once, ca. 60 GB) or from a regional file (such as Europe, North America, etc.), and then perform an extraction on this sub-file.

To clip shapes from the planet.osm.pbf file, `osm-flex` uses one of teh command line tools `osmosis` or `osmconvert` under the hood. Both need to be manually installed:

Osmosis (works for Windows, Linux and Mac) needs to be installed as explained here: https://wiki.openstreetmap.org/wiki/Osmosis/Installation. You may also need to install Java.

Osmconvert (works for Windows, Linux and Mac) needs to be installed as explained here: https://wiki.openstreetmap.org/wiki/osmconvert. For Mac, installation instructions may not be satisfactory. Alternatively, it may work running the command line command `brew install osmfilter`, which includes osmconvert.

There are currently three implemented ways to clip desired shapes: * By indicating a bounding box passed as lists ([xmin, ymin, xmax, ymax]) –> `clip_from_bbox()` * By providing a path to a .poly file which contains the outline of the shape to be cut out. –> `clip_from_poly()` * By providing a list of shapes (polygons, multipolygons), which represent the outline of the shape to be cut out. This method also creates and saves a corresponding .poly file in the background, which is then passed to osmosis / osmconvert. –> `clip_from_shapes()`

Note Takes quite a while to cut out certain parts (up to an hour..) The cutting process is sped up if the parent file is smaller (e.g. a regional osm.pbf file instead of the entire planet file).

Note A description of the file format of .poly files, with certain aspects to consider, can be found here: https://wiki.openstreetmap.org/wiki/Osmosis/Polygon_Filter_File_Format . A routine to convert shapes at admin3 to admin1 level for any country in the world into poly files can be found here: https://github.com/ElcoK/osm_clipper.

Note Anti-meridian crossings (along the 180° latitude line) cause issues. Polygons have to be split along this line and passed separately (within the same list, or in the same .poly file is ok). For detecting and splitting those cases, see the following article with linked code on GitHub: https://towardsdatascience.com/around-the-world-in-80-lines-crossing-the-antimeridian-with-python-and-shapely-c87c9b6e1513

Example: Clip data within a bounding box

[21]:
```
# Define a bbox around roughly Honduras, San Salvador & Nicaragua
bbox_customreg = [-90.043806, 10.939216, -83.116254, 15.956278]

# load the central-america parent file from which we want to clip
path_ca_dump = osm_flex.download.get_region_geofabrik('central-america')
# new save path
path_custom_dump = os.path.join(osm_flex.config.OSM_DATA_DIR, 'custom_region.osm.pbf')

# perform clipping with command line tool osmosis (other option: osmconvert)
osm_flex.clip.clip_from_bbox(
    bbox_customreg,
    path_ca_dump,
    path_custom_dump,
    kernel='osmosis')
```

```
INFO:osm_flex.download:Skip existing file: /Users/evelynm/osm/osm_bpf/central-america-
↪latest.osm.pbf
INFO:osm_flex.clip:File doesn`t yet exist or overwriting old one.
                   Assembling osmosis command.
INFO:osm_flex.clip:Extracting from larger file…
                   This will take a while
Nov 24, 2023 4:22:44 PM org.openstreetmap.osmosis.core.Osmosis run
INFO: Osmosis Version 0.48.3
Nov 24, 2023 4:22:44 PM org.openstreetmap.osmosis.core.Osmosis run
INFO: Preparing pipeline.
Nov 24, 2023 4:22:44 PM org.openstreetmap.osmosis.core.Osmosis run
INFO: Launching pipeline execution.
Nov 24, 2023 4:22:44 PM org.openstreetmap.osmosis.core.Osmosis run
INFO: Pipeline executing, waiting for completion.
Nov 24, 2023 4:23:34 PM org.openstreetmap.osmosis.core.Osmosis run
INFO: Pipeline complete.
Nov 24, 2023 4:23:34 PM org.openstreetmap.osmosis.core.Osmosis run
INFO: Total execution time: 50140 milliseconds.
```

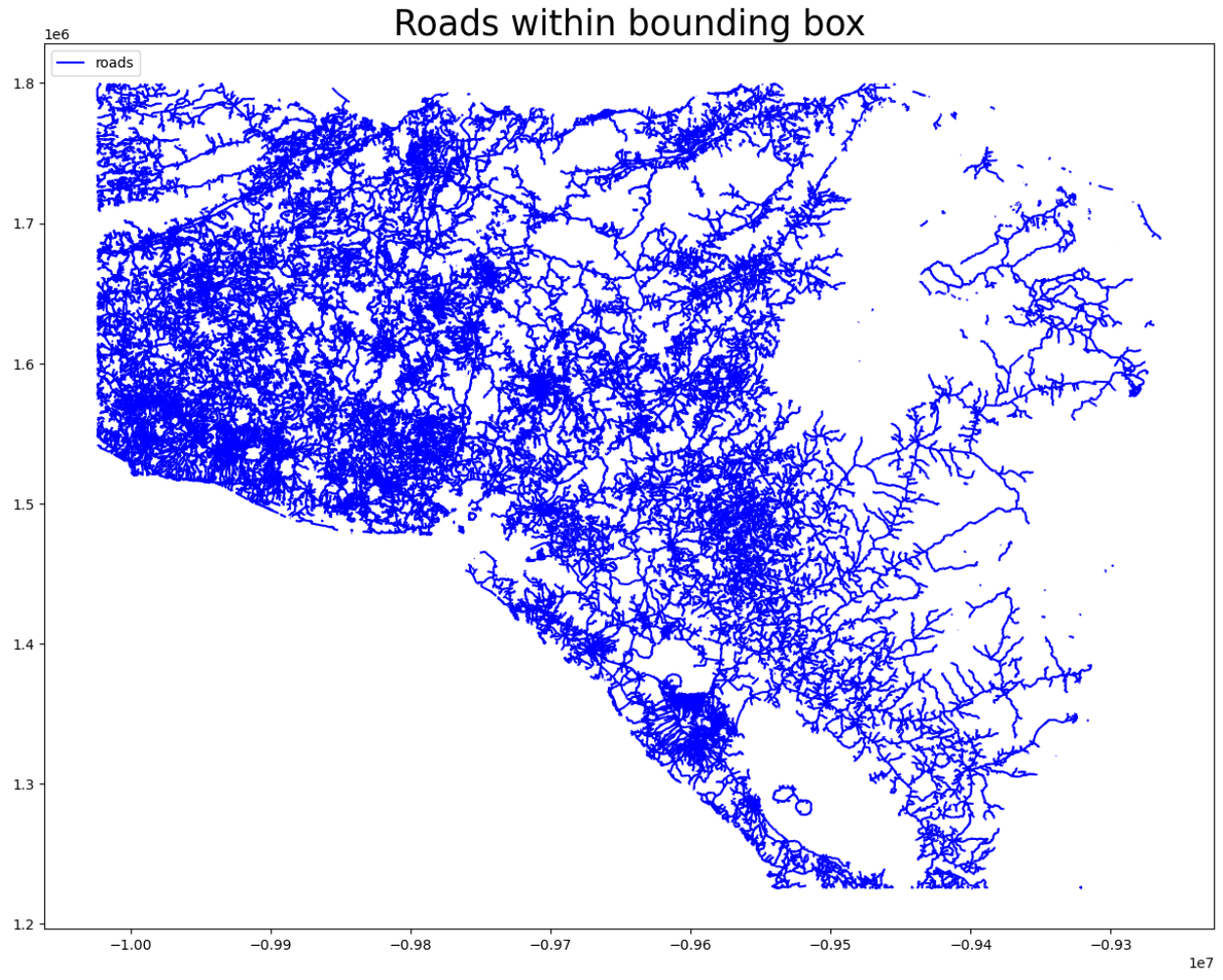
Now we can repeat a similar query (e.g. road data) for our newly cut-out data region.

[23]:
```
# lets extract all roads from the Honduras file, via the wrapper
gdf_roads_customreg = osm_flex.extract.extract_cis(path_custom_dump, 'road')
```

```
INFO:osm_flex.extract:query is finished, lets start the loop
extract points: 0it [00:00, ?it/s]
INFO:osm_flex.extract:query is finished, lets start the loop
extract multipolygons: 100
↪%|
↪3/3 [00:13<00:00,  4.58s/it]
INFO:osm_flex.extract:query is finished, lets start the loop
extract lines: 100
↪%|
↪272481/272481 [00:19<00:00, 14078.86it/s]
```

[25]:
```
# plot results
ax = gdf_roads_customreg.to_crs(epsg=3857).plot(figsize=(15, 15), alpha=1,␣
↪markersize=5, color='blue',
                    edgecolor='blue', label='roads')
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles=handles, loc='upper left')
ax.set_title('Roads within bounding box', fontsize=25)
plt.show()
```

```
/var/folders/jm/4przql117d9gb9g4hb45fd4c0000gp/T/ipykernel_4655/3326432630.py:4:␣
↪UserWarning: Legend does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html
↪#implementing-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
```



Clip data within a custom (multi-)polygon See the tutorial provided by osm-flex.

1.2: Download data from the overpass-API

At the moment, the `OSMApiQuery` class accepts both an *area* and a *query condition* as inputs.

Area can be a bounding box (xmin, ymin, xmax, ymax) or a polygon. The query conditions must be a string in the format `'["key"]'` or `'["key"="value"]'`, etc. For more query syntaxes, have a look at the OverpassQL page.

7.1.1 Example: Buildings & Churches in the city of Zurich

[2]:
```
# reload required modules
import shapely
from climada_petals.entity.exposures.osm_dataloader import OSMApiQuery
from climada import CONFIG

DATA_DIR = CONFIG.exposures.openstreetmap.local_data.dir()
```

[4]:
```
# Two area-formats that are accepted: a bounding box tuple or list (xmin, ymin, xmax,␣
↪ymax) and polygons
area_bbox = (8.5327506, 47.368260, 8.5486078, 47.376877)
area_poly = shapely.geometry.Polygon([(8.5327506, 47.368260), (8.5486078, 47.376877),␣
↪(8.5486078, 47.39)])

# Two examples for query conditions:
condition_church = '["amenity"="place_of_worship"]'
condition_building = '["building"]'

# Initialize OSMApiQuery instances
zrh_churchquery_bbox = OSMApiQuery.from_bounding_box(area_bbox, condition_church)
zrh_buildingquery_poly = OSMApiQuery.from_polygon(area_poly, condition_building)
```

After instantiating the queries with an area and a query condition, the data can now be downloaded from the overpass-API. Depending on the time of the day and the size of the query, this can lead to overloads.

The following request should be small enough that it always works:

[5]:
```
gdf_zrh_churches = zrh_churchquery_bbox.get_data_overpass()
```

[6]:
```
gdf_zrh_buildings = zrh_buildingquery_poly.get_data_overpass()
```

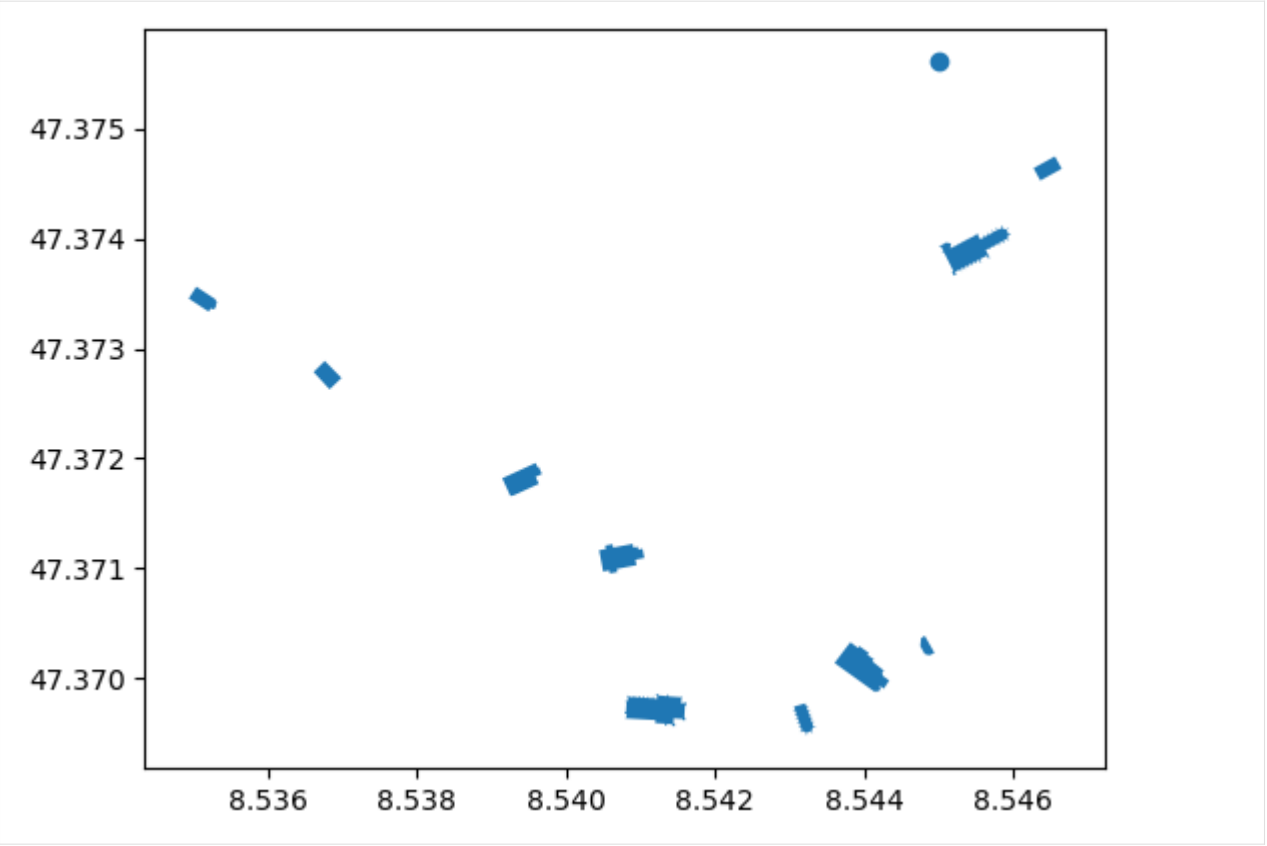
```
2023-12-07 13:31:50,483 - climada_petals.entity.exposures.osm_dataloader - INFO -␣
↪Empty geometry encountered.
```

Let's have look at our downloaded data. The downloaded data is assembled into a geodataframe with an `osm_id` column and a `tags` columns. The latter reports all the tags associated with the respective result.

[7]:
```
gdf_zrh_churches.head()
```

[7]:
```
     osm_id                                              geometry  \
0  24701951  POLYGON ((8.54050 47.37097, 8.54058 47.37098, …
1  33854803  POLYGON ((8.54552 47.37404, 8.54552 47.37404, …
2  36916418  POLYGON ((8.54413 47.37017, 8.54405 47.37021, …
3  40478435  POLYGON ((8.54656 47.37475, 8.54628 47.37464, …
4  80338523  POLYGON ((8.53520 47.37333, 8.53494 47.37345, …

                                                tags
0  {'addr:city': 'Zürich', 'addr:country': 'CH', …
1  {'addr:city': 'Zürich', 'addr:country': 'CH', …
2  {'addr:city': 'Zürich', 'addr:country': 'CH', …
3  {'addr:city': 'Zürich', 'addr:country': 'CH', …
4  {'addr:city': 'Zürich', 'addr:country': 'CH', …
```

[8]:
```
gdf_zrh_churches.plot();
```

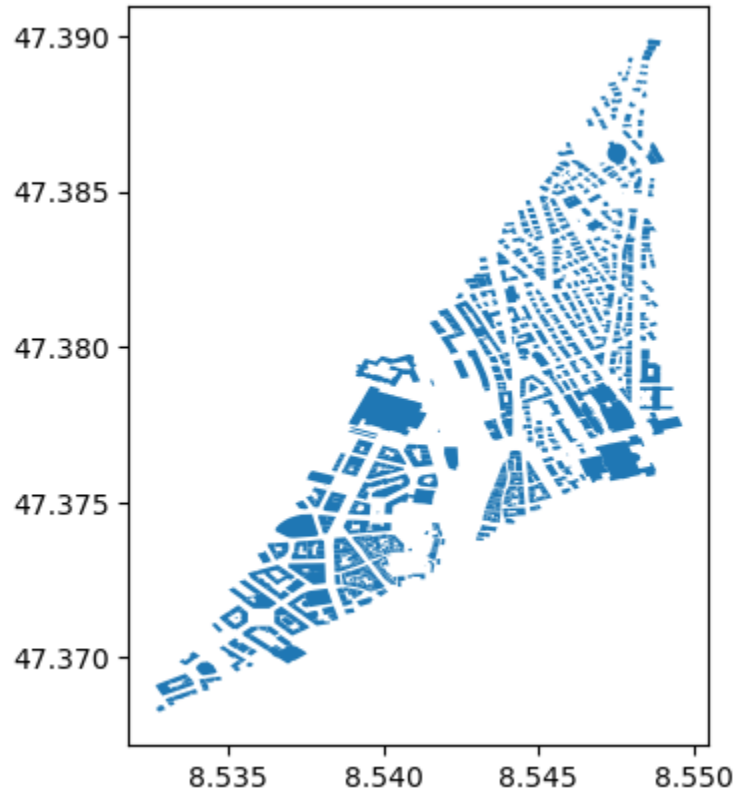


[30]:
```
gdf_zrh_buildings.head()
```

[30]:
```
    osm_id                                           geometry  \
0     4633  POLYGON ((8.54832 47.37961, 8.54831 47.37936, …
1     7565                          GEOMETRYCOLLECTION EMPTY
2   405794  POLYGON ((8.54124 47.37587, 8.54119 47.37583, …
3  4235320  POLYGON ((8.54372 47.37844, 8.54377 47.37804, …
4  4281153  POLYGON ((8.53952 47.37959, 8.53952 47.37960, …

                                                tags
0  {'addr:city': 'Zürich', 'addr:housename': 'ETH…
1  {'building': 'public', 'building:levels': '6',…
2  {'addr:city': 'Zürich', 'addr:country': 'CH', …
3     {'building': 'office', 'type': 'multipolygon'}
4  {'addr:housenumber': '2', 'addr:street': 'Muse…
```

[31]:
```
gdf_zrh_buildings.plot()
```

[31]:
```
<Axes: >
```



Part 2: Risk computation with OSM data as CLIMADA Exposures

Having obtained GeoDataFrame(s) with OSM data in part 1, the following steps for impact calculations will be analogous to any impact calculations on Exposures with point, polygon and / or line geometries. The tutorial in climada_python/doc/tutorial/entity_exposures_lines_polygons gives detailed info on how to generally handle those types within CLIMADA exposures.

For completeness, only a short demo will be given here using OSM data.

2.1: Setting up a high-res CLIMADA exposure from scratch with point, line & polygon data from OSM

The general steps are the following to create exposures from line or polygon data from OSM:

1. Lines / Polygons to points: Specify a distance into which lines are split, or an area into which polygons are split, to get an interpolated gdf.
2. Valuation: Indicate a value per meter or m2, or use LitPop asset values to re-distribute proportionally.
3. Point-Exposure is ready to use
4. Re-aggregation to initial shapes after impact calculation.

CLIMADA exposure from OSM data (roads; lines)

We will take the previously loaded gdf of roads in Honduras. We can set up a CLIMADA point exposure for one point per 500m using the respective utils function. See the respective tutorial climada_python/doc/tutorial/entity_exposures_lines_polygons for details on the interpolation modules.

[45]:
```
from climada.entity.exposures import Exposures
import climada.util.lines_polys_handler as u_lp
```

(continues on next page)

(continued from previous page)

```
exp_rd_hnd = Exposures(gdf_roads_hnd)
res = 500
disagg_val = 500*res
disagg_met = u_lp.DisaggMethod.FIX

# interpolate to point-based exposure
exp_road_pnt = u_lp.exp_geom_to_pnt(exp_rd_hnd, res=res, to_meters=True, disagg_
↪met=disagg_met, disagg_val=disagg_val)
exp_road_pnt.set_lat_lon()
exp_road_pnt.check()

# next step: impact calculation (see 2.3)
```

```
2023-11-27 10:33:54,387 - climada.util.lines_polys_handler - WARNING - 1870 lines␣
↪with a length < 10*resolution were found. Each of these lines is disaggregate to␣
↪one point. Reaggregatint values will thus likely lead to overestimattion. Consider␣
↪chosing a smaller resolution or filter out the short lines.
2023-11-27 10:34:00,589 - climada.util.lines_polys_handler - WARNING - Polygon␣
↪smaller than resolution. Setting a representative point.
2023-11-27 10:34:00,593 - climada.util.lines_polys_handler - WARNING - Polygon␣
↪smaller than resolution. Setting a representative point.
2023-11-27 10:34:00,693 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2023-11-27 10:34:00,946 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2023-11-27 10:34:01,021 - climada.entity.exposures.base - INFO - Setting impf_ to␣
↪default impact functions ids 1.
2023-11-27 10:34:01,023 - climada.entity.exposures.base - INFO - category_id not set.
2023-11-27 10:34:01,024 - climada.entity.exposures.base - INFO - cover not set.
2023-11-27 10:34:01,025 - climada.entity.exposures.base - INFO - deductible not set.
2023-11-27 10:34:01,026 - climada.entity.exposures.base - INFO - region_id not set.
2023-11-27 10:34:01,027 - climada.entity.exposures.base - INFO - centr_ not set.
```

2.2: Refining LitPop - Assigning high-value areas or cutting out low value areas using OSM info

Instead of using OSM exposure data directly, one may want to combine these data with another asset value layer, e.g. `LitPop`. Two options are conceivable: Using OSM to refine the coarse-scale LitPop layers by cutting out "low-value areas", or using spatially resolved asset values from LitPop to assign monetary values to the OSM exposure. The following two examples hint in this direction, but are under development (see older versions of CLIMADA for these functions).

Using asset value layers to valuate OSM exposures

Instead of just assuming an arbitrary m2-value for the asset valuation, we can also use an estimate for asset values from `LitPop`, using the `from_shape_and_countries` method and re-assigning this to our high-resolution exposure optained via openstreetmap:

[55]:
```
from climada.entity.exposures.litpop import LitPop
import geopandas as gpd
import numpy as np

# define a few helper functions
def _ckdnearest(vs_assign, gdf_base, k=1):
    """
    see https://gis.stackexchange.com/a/301935
    Parameters
    ----------
    vs_assign : gpd.GeoDataFrame or Point
    gdf_base : gpd.GeoDataFrame
```

(continues on next page)

(continued from previous page)

```
    """
    if (isinstance(vs_assign, gpd.GeoDataFrame)
        or isinstance(vs_assign, pd.DataFrame)):
        n_assign = np.array(list(vs_assign.geometry.apply(lambda x: (x.x, x.y))))
    else:
        n_assign = np.array([(vs_assign.geometry.x, vs_assign.geometry.y)])
    n_base = np.array(list(gdf_base.geometry.apply(lambda x: (x.x, x.y))))
    btree = cKDTree(n_base)
    dist, idx = btree.query(n_assign, k=k)
    return dist, np.array(gdf_base.iloc[idx.flatten()].index).reshape(dist.shape)

def assign_litpop_values(gdf_buildings, gdf_litpop):
    __, ix_match = _ckdnearest(gdf_buildings, gdf_litpop)
    return gdf_litpop.loc[ix_match]['value'].values

def correct_total_values(gdf_buildings, gdf_litpop):
    corr_factor = gdf_litpop.value.sum()/gdf_buildings.value.sum()
    return gdf_buildings['value']*corr_factor

def distribute_lp_area(exp_osm, val_lp, mode='area'):
    if mode=='area':
        val_tot = exp_osm.gdf.geometry.area.sum()
    elif mode=='length':
        val_tot = exp_osm.gdf.geometry.length.sum()
    else:
        NotImplemented
    exp_osm.gdf['value'] = exp_osm.gdf.apply(lambda row: row.geometry.area/val_
↪tot*val_lp, axis=1)
    return exp_osm
```

[46]:
```
# Get a LitPop exposure for Zurich
poly_zrh = shapely.geometry.Polygon([(8.5327506, 47.368260), (8.5486078, 47.376877),␣
↪(8.5486078, 47.39)])
exp_litpop_zrh = LitPop.from_shape_and_countries(poly_zrh, ["CHE"],
                                                 res_arcsec=30, exponents=(1,1),
                                                 fin_mode='pc')
# total monetary value to be distributed among buildings:
val_zrh = exp_litpop_zrh.gdf['value'].sum()
```

```
2023-11-27 10:40:57,896 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CHE (756)…

2023-11-27 10:40:59,072 - climada.util.finance - INFO - GDP CHE 2014: 7.265e+11.
2023-11-27 10:40:59,733 - climada.util.finance - INFO - GDP CHE 2018: 7.256e+11.
2023-11-27 10:40:59,776 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-11-27 10:40:59,777 - climada.entity.exposures.base - INFO - category_id not set.
2023-11-27 10:40:59,778 - climada.entity.exposures.base - INFO - cover not set.
2023-11-27 10:40:59,779 - climada.entity.exposures.base - INFO - deductible not set.
2023-11-27 10:40:59,779 - climada.entity.exposures.base - INFO - centr_ not set.
2023-11-27 10:40:59,807 - climada.entity.exposures.litpop.litpop - WARNING - Could␣
↪not write attribute meta with ValueError:
2023-11-27 10:40:59,808 - climada.entity.exposures.litpop.litpop - WARNING - zero-
↪size array to reduction operation minimum which has no identity
```

[60]:
```
# Distribute building values based on building footprint area and total LitPop value␣
```

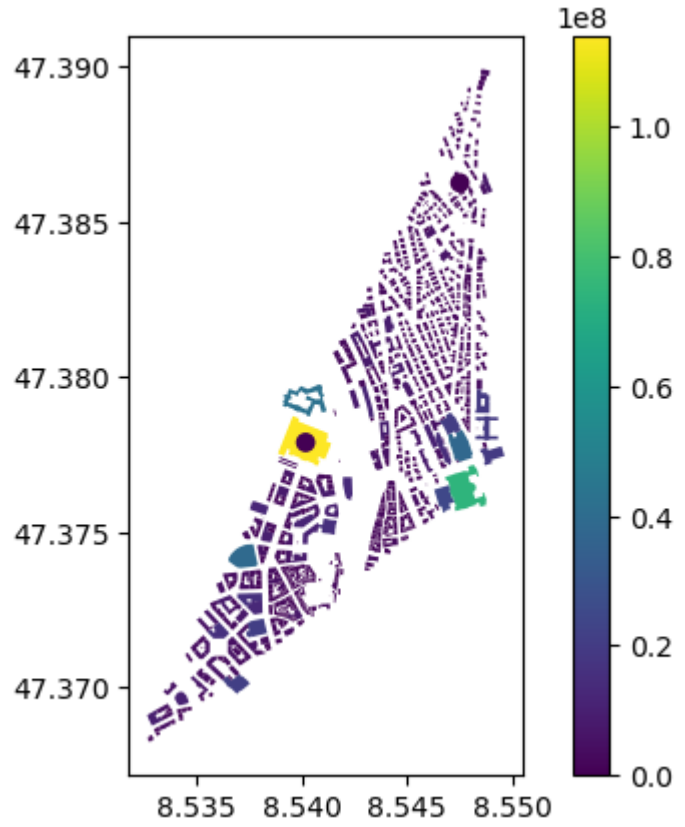
(continues on next page)

(continued from previous page)

```
↪for region
exp_buildings_zrh = distribute_lp_area(Exposures(gdf_zrh_buildings), val_zrh, mode=
↪'area')
exp_buildings_zrh.gdf.plot('value', legend=True)
```

```
/var/folders/jm/4przql117d9gb9g4hb45fd4c0000gp/T/ipykernel_4655/2181821964.py:34:␣
↪UserWarning: Geometry is in a geographic CRS. Results from 'area' are likely␣
↪incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS␣
↪before this operation.

  val_tot = exp_osm.gdf.geometry.area.sum()
```

[60]:
```
<Axes: >
```



Example for cutting out "low-value" features from LitPop cells

The idea behind this "reverse approach" to the one demonstrated before is to use OpenStreetMap info as a "stencil" for downscaling the coarser LitPop asset values (roughly 1x1km2), by defining low-value areas, and allocating all LitPop values to the *remaining* area.

There are in turn several methods on how the allocation is done: * equally spreading out the gridcell value on all the "valid" areas * re-locating LitPop values based on nearest-neighbour centroids

"Low value features" in terms of asset values, i.e. natural areas, etc. can be found on OSM under the following keys and key-value pairs (not extensive list): * 'natural', * 'water', * 'waterway', * 'landuse=forest', * 'landuse=farmland', * 'landuse=grass', * 'wetland'

[39]:
```
# Load required packages:
```

[ ]:
```
# One command does it all (getting LitPop Exp, re-assigning values, converting back␣
↪into exposure format)
```

2.3: Performing impact calculations on OSM-based exposures

[ ]:
```
# now perform a standard impact calc, specifying hazard and vulnerability curves ...
```

[ ]:
```
# EITHER in 3 Steps interpolating exposures to Points, then impact calculation, then␣
↪reaggregation
# OR: all in one go (interpolation, impact calc, re-aggregation ) using imp = u_lp.
↪calc_geom_impact(...)
```

CHAPTER

EIGHT

CROP PRODUCTION RISK BASED ON ISIMIP AND FAO DATA

8.1 Summary

This tutorial gives an overview of the modules in CLIMADA used to compute climate related risks to crop production on a 0.5° grid. The risk calculation is based on yearly crop yield as simulated by global gridded crop models (GGCMs) that are forced with climate variables such as temperature and water availability provided from climate model output or re-analysis data.

The hazard *relative_cropyield* is not a typical natural hazard but rather an aggregation of climatic impacts on the crop yield at each location. *relative_cropyield* intensi is equal to the change in crop_yield in a given year from the long-term averageIt is based on the crop yield simulated by GGCMs. In the CLIMADA framework, we are setting relative_cropyield as "hazard" because it describes the year-by-year climatic influence on agriculture, each year representing one event. This hazard can be applied on any exposure that represents a (mean) amount of crop produced at any location.

Here, we use the exposure *crop_production* (in tonnes or USD per year) that distributes national crop production as extracted from FAO statistics proportional to a gridded distribution of crop production that is based on mean crop yield in tonnes per hectare multiplied with the hectares of harvest area per grid cell.

8.1.1 Example calculation:

At a certain grid cell, the area fraction for non-irrigated rice is 5% and the area of the grid cell is 250,000 ha with an average historical crop yield of 6 tonnes per ha and year.

The exposure (*crop_production*) value is the product of area fraction, area, and average yield = $0.05 * 250,000 ha * 6t/(ha * y) = 75,000 t/y$.

The hazard intensity at this grid cell for non-irrigated rice in a certain year is -20% (*relative_cropyield*), caused by climate variables such as temperature and water availability during that year.

For this specific year, the impact as computed with Impact.calc is the product of hazard and exposure: $75,000 t/y * (-0.20) * 1y = -15,000 t$.

8.2 Data sources

The two classes *RelativeCropyield(Hazard)* and *CropProduction(Exposure)* can be combined to calculate climate impacts on crop production based on simulations data from global gridded crop models (GGCMs) within the Inter-Sectoral Impact Model Intercomparison Project (ISIMIP, https://www.isimip.org/) as well as statistics from the Food and Agriculture Organization of the United Nations (FAO, http://www.fao.org/faostat/en/#home).

From the ISIMIP project, a variety of model runs with yearly crop yield data on a spatial resolution of 0.5° x 0.5° are available. Each run is based on one GGCM forced by a climate model output or re-analysis data. Runs are available for different crop types, model combinations, historical climate and future climate scenarios, and other model parameters. The hazard is generated by the class *RelativeCropyield(Hazard)* that extracts crop yield data simulated by GGCMs. The GGCM runs provided by ISIMIP are forced with the output from climate models (e.g. in ISIMIP2b, ISIMIP3b) or re-analyis data (ISIMIP2a, ISIMIP3a). The driving climate variables for crop yield are temperature, water availability, CO2 concentrations, and nitrogen availability. Additionally, land use data required for *CropProduction(Exposure)* is available from the ISIMIP input data (e.g. histsoc_landuse-15crops_annual_1861_2005.nc for ISIMIP2).

The required ISIMIP data sets are available from https://esg.pik-potsdam.de/search/isimip/ (choose *Variable* = *yield* for crop yield and *landuse-15crops* for land use data).

In this tutorial we show how a *RelativeCropyield* and a *CropProduction* instance can be initiated and translated into socio-economic impacts in the form of (yearly) crop production losses / gains in tonnes or USD.

8.3 RelativeCropyield Hazard

Hazard intensity in the class *RelativeCropyield* is defined as yearly crop yield relative to a historical mean simulated with the same model combination. Each model year represents one event in the hazard instance.

The method *set_from_isimip_netcdf()* generates a *Hazard* instance from one model run, with intensity 'Yearly Yield'. This requires multiple input parameters to specify the model run:

```
input_dir (string): path to input data directory
bbox (list of four floats): bounding box:
    [lon min, lat min, lon max, lat max],
yearrange (int tuple): year range for hazard set, f.i. (1976, 2005)
ag_model (str): abbrev. agricultural model (only when input_dir is selected)
    f.i. 'gepic' etc.
cl_model (str): abbrev. climate model (only when input_dir is selected)
    f.i. 'gfdl-esm2m' etc.
scenario (str): climate change scenario (only when input_dir is selected)
    f.i. 'historical' or 'rcp60'
soc (str): socio-economic trajectory (only when input_dir is selected)
    f.i. '2005soc' or 'histsoc'
co2 (str): CO2 forcing scenario (only when input_dir is selected)
    f.i. 'co2' or '2005co2'
crop (str): crop type, e.g. 'whe', 'mai', 'soy' or 'ric'
irr (str): irrigation type, e.g. 'noirr' or 'irr'
```

In addition to the general attributes of the *Hazard()* class, the class *RelativeCropyield()* has further attributes related to the crop type and intensity definition:

```
crop (str): crop type, e.g. 'whe', 'mai', 'soy', or 'ric';
intensity_def (str): intensity unit definition, either
    'Relative Yield' (unitless), 'Yearly Yield' [t/(y*ha)], or 'Percentile' [t/(y*ha)]
```

To convert intensity to 'Relative Yield', the methods *calc_mean()* and *set_rel_yield_to_int()* can be applied as shown below. Attention: This is required for impact calculations in combination with *CropProduction(Exposure)*.

To initiate one or more hazard files from a variety of model runs, a more convenient function is available: *climada.hazard.relative_cropyield.set_multiple_rc_from_isimip()*, setting intensity to 'Relative Yield' for you. The function *set_multiple_rc_from_isimip()* extracts all model specifications directly from the filenames. Thus, it only requires the following inputs:

```
input_dir (str): path to input data directory
output_dir (str): path to output data directory (hazard sets are saved there in HDF5-
↪format)
return_data (boolean): set to True if you want the function to return a list␣
↪containing the haz. sets
```

Below two examples for initiating an hazard instance and setting intensity to relative yield:

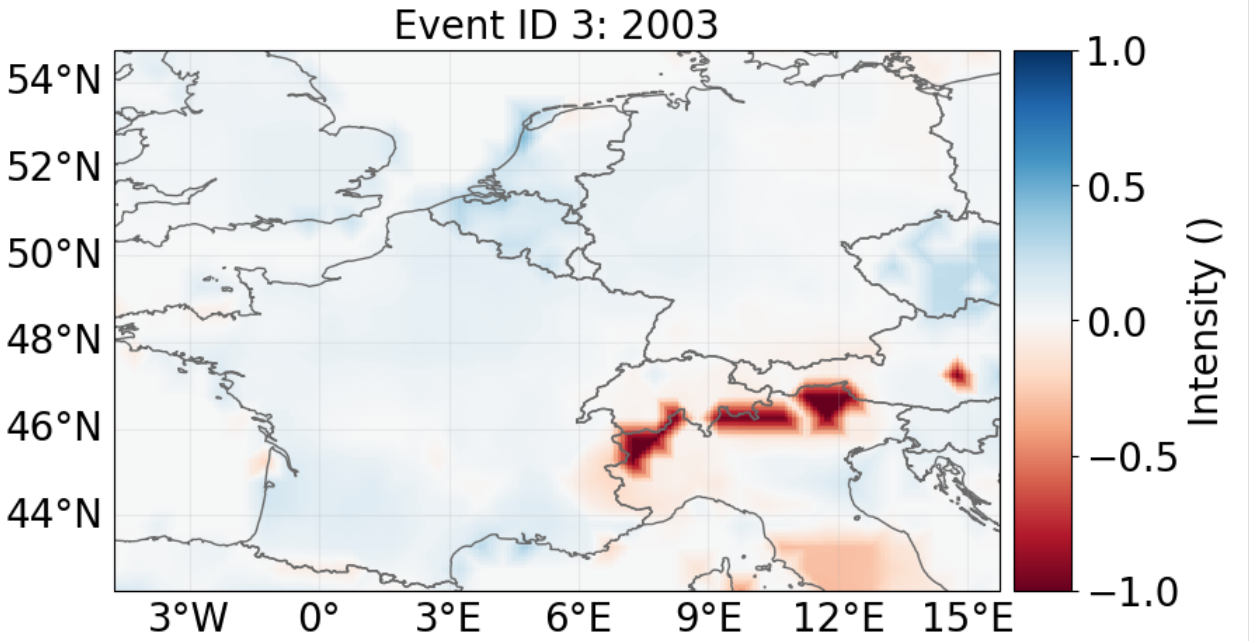
8.3.1 Initiate a single hazard instance and set intensity to relative yield manually (demo data sample):

- using *set_from_isimip_netcdf()*
- using demo data (cropped to France and Germany, 2001-2005)

[3]:
```
import os
from climada_petals.hazard.relative_cropyield import RelativeCropyield
from climada.util.constants import DEMO_DIR as INPUT_DIR

FN_STR_DEMO = 'annual_FR_DE_DEMO'

yearrange_haz = (2001, 2005) # yearrange for hazard (demo data only available from␣
↪2001 to 2005)
yearrange_hist_mean = (2001, 2005) # yearrange for reference historical mean (demo␣
↪data only available from 2001 to 2005)
haz = RelativeCropyield()
haz.set_from_isimip_netcdf(input_dir=INPUT_DIR, yearrange=yearrange_haz, ag_model=
↪'lpjml',
                           cl_model='ipsl-cm5a-lr', scenario='historical', soc='2005soc',
                           co2='co2', crop='whe', irr='noirr', fn_str_var=FN_STR_DEMO)

print("\nBefore calling set_rel_yield_to_int(), intensity is '%s' with unit '%s'.\n"
↪%(haz.intensity_def, haz.units))

hist_mean = haz.calc_mean(yearrange_hist_mean) # requires reference year range as␣
↪input
"""compute historical mean yield per grid cell for reference (base line)"""
haz.set_rel_yield_to_int(hist_mean)
"""set intensity to relative yield by dividing yield/hist_mean"""

print("After calling set_rel_yield_to_int(), intensity is '%s' with unit '%s'.\n"
↪%(haz.intensity_def, haz.units))

haz.plot_intensity_cp(event=3)
"""The map shows relative crop yield in the model year 2003. Positive (negative)␣
↪values correspond to a relative yield surplus (deficit)"""
# please note that the run used here is not based on historical re-analysis data but␣
↪on simulated climate,
# i.e. the climate variables of year "2003" do not correspond to the actual year 2003.
# (only the forcing of the climate models is historical)
```

```
Before calling set_rel_yield_to_int(), intensity is 'Yearly Yield' with unit 't / y / ha'.

After calling set_rel_yield_to_int(), intensity is 'Relative Yield' with unit ''.
```

[3]:
```
'The map shows relative crop yield in the model year 2003. Positive (negative) values correspond to a relative yield surplus (deficit)'
```



8.3.2 Initiate a relative yield hazard set from multiple input files:

- using *set_multiple_rc_from_isimip()*, which creates hazard sets from all NetCDF-files found in the input directory
- in this example, I used the output from GGCM GEPIC forced with GFDL-ESM2M output for rice, both irrigated and non-irrigated. You can however also run the script with other data sets of the same format.

Requires data download from https://esg.pik-potsdam.de/search/isimip/: - gepic_gfdl-esm2m_ewembi_historical_2005soc_co2_yield-ric-firr_global_annual_1861_2005.nc - gepic_gfdl-esm2m_ewembi_historical_2005soc_co2_yield-ric-noirr_global_annual_1861_2005.nc - gepic_gfdl-esm2m_ewembi_rcp60_2005soc_co2_yield-ric-firr_global_annual_2006_2099.nc - gepic_gfdl-esm2m_ewembi_rcp60_2005soc_co2_yield-ric-noirr_global_annual_2006_2099.nc

These files contain historical and RCP6.0 future simulations for *rice* yield, both for fully irrigated ("firr") and no irrigation ("noirr"). Forcing climate model: *gfdl-esm2m*. Crop model: *gepic*.

Please note: when calling *init_full_hazard_set()*. the historical mean (hist_mean) is averaged over all model combinations for each crop and irrigation type. Like this, a cross-model exposure can be initiated using this model average of hist_mean as input.

[ ]:

[5]:
```
from climada import CONFIG
from climada_petals.hazard.relative_cropyield import set_multiple_rc_from_isimip


data_path = CONFIG.local_data.save_dir.dir() / "ISIMIP_crop" # set path of working␣
↪data directory
input_haz_dir = data_path / "Input" / "Hazard_tutorial" # set path where you place␣
↪hazard input data
input_haz_dir.mkdir(parents=True, exist_ok=True)
# (Place crop yield data (.nc) from ISIMIP in input_haz_dir)

output_dir = data_path / "Output" # set output directory
path_hist_mean = output_dir / "Hist_mean" # set output directory for hist_mean
path_hist_mean.mkdir(parents=True, exist_ok=True)

filelist_haz, hazards_list = set_multiple_rc_from_isimip(input_dir=input_haz_dir,␣
↪output_dir=output_dir,
                                                         isimip_run='ISIMIP2b', return_
↪data=True)

print("\nComputed and saved the following files: \n")
print(filelist_haz)

print("\nIntensity of the hazard sets is '%s' with unit '%s'.\n" %(hazards_list[0].
↪intensity_def, hazards_list[0].units))

hazards_list[1].plot_intensity_cp(event=1);
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[5], line 14
     11 path_hist_mean = output_dir / "Hist_mean" # set output directory for hist_mean
     12 path_hist_mean.mkdir(parents=True, exist_ok=True)
---> 14 filelist_haz, hazards_list = set_multiple_rc_from_isimip(input_dir=input_haz_
↪dir, output_dir=output_dir,
     15                                                         isimip_run='ISIMIP2b',␣
↪return_data=True)
     17 print("\nComputed and saved the following files: \n")
     18 print(filelist_haz)

File c:\users\me\git\climada_petals\climada_petals\hazard\relative_cropyield.py:551,␣
↪in set_multiple_rc_from_isimip(input_dir, output_dir, bbox, isimip_run, yearrange_
↪his, yearrange_mean, return_data, save, combine_subcrops)
    547 filename_list = list()
    548 output_list = list()
    550 (his_file_list, file_props, hist_mean_per_crop,
--> 551  scenario_list, _, combi_crop_list) = init_hazard_sets_isimip(filenames,
    552                                                               input_dir=input_
↪dir,
    553                                                               bbox=bbox,␣
↪isimip_run=isimip_run,
    554                                                               yearrange_
↪his=yearrange_his,
    555                                                               combine_
↪subcrops=combine_subcrops)
    557 if (yearrange_mean is None) and (isimip_run == 'ISIMIP2b'):
    558     yearrange_mean = YEARCHUNKS[file_props[his_file_list[0]]['scenario']][
↪'yearrange_mean']
```

(continues on next page)

(continued from previous page)

```
File c:\users\me\git\climada_petals\climada_petals\hazard\relative_cropyield.py:791,␣
↪in init_hazard_sets_isimip(filenames, input_dir, bbox, isimip_run, yearrange_his,␣
↪combine_subcrops)
    785 # generate hazard using the first file to determine the size of the historic␣
↪mean
    786 # file structure: ag_model _ cl_model _ scenario _ soc _ co2 _
    787 #   yield-crop-irr _ fn_str_var _ startyear _ endyear . nc
    788 #e.g. gepic_gfdl-esm2m_ewembi_historical_2005soc_co2_yield-whe-noirr_
    789 #   global_annual_1861_2005.nc
    790 haz_dummy = RelativeCropyield()
--> 791 haz_dummy.set_from_isimip_netcdf(input_dir=input_dir, filename=his_file_
↪list[0], bbox=bbox,
    792                                  scenario=file_props[his_file_list[0]][
↪'scenario'],
    793                                  yearrange=(file_props[his_file_list[0]][
↪'startyear'],
    794                                             file_props[his_file_list[0]][
↪'endyear']))
    796 # initiate the historic mean for each combination of crop and irrigation type
    797 # the idx keeps track of the row in which the hist_mean values are written␣
↪per crop-irr to
    798 # ensure that all files are assigned to the corresponding crop-irr combination
    799 hist_mean_per_crop = dict()

IndexError: list index out of range
```

8.4 CropProduction Exposure

The *CropProduction* exposure data represents the mean crop production per grid cell. For creating an exposure instance, the following main input data are combined: - harvest area fraction (from landuse data): fraction of grid cell area where a crop is grown with / without irrigation, unitless; - total grid cell area $[ha]$: computed from grid; - historical mean yield (hist_mean): simulated crop yield with / without irrigation per grid cell, usually averaged over several model combinations and years, can be initiated with function *set_multiple_rc_from_isimip* $[t/(ha * y)]$; - crop production price from FAO when unit is USD $[USD/t]$.

Crop production = fraction * area * hist_mean * price

$[USD/y = ha * t/(ha * y) * USD/t]$

Unit definitions:

- USD: US dollars
- y: year
- ha: hectar, $1ha = 10000m^2$
- t: tonnnes, $1t = 1000kg$

The method *set_from_isimip_netcdf()* generates a *Exposure* instance for one crop type and irrigation parameter, with unit 't/year' or 'USD/year'. This requires multiple input parameters:

```
input_dir (string): path to input data directory
filename (string): name of the landuse-file to use,
    e.g. "histsoc_landuse-15crops_annual_1861_2005.nc"
```

(continues on next page)

(continued from previous page)

```
hist_mean (array): historic mean crop yield per centroid (from hazard)
bbox (list of four floats): bounding box:
    [lon min, lat min, lon max, lat max]
yearrange (int tuple): year range for exposure set
    f.i. (1990, 2010)
scenario (string): climate change and socio economic scenario
    f.i. 'histsoc' or 'rcp60soc'
cl_model (string): abbrev. climate model (only when landuse data
is future projection)
    f.i. 'gfdl-esm2m' etc.
crop (string): crop type
    f.i. 'mai', 'ric', 'whe', 'soy'
irr (string): irrigation type
    f.i 'firr' (full irrigation), 'noirr' (no irrigation) or 'combined'= firr+noirr
unit (string): unit of the exposure (per year)
    f.i 'USD/y' or 't/y'
fn_str_var (string): FileName STRing depending on VARiable and
    ISIMIP simuation round
```

In addition to the general attributes of the *Exposures()* class, the class *CropProduction()* has one further attribute related:

```
crop (str): crop type, e.g. 'whe', 'mai', 'soy', or 'ric'
```

Below two examples for initiating an Exposures instance:

8.4.1 Initiating a single exposure instance (demo data sample):

[6]:
```
from matplotlib import colors

from climada_petals.entity.exposures.crop_production import CropProduction
from climada.util.constants import DEMO_DIR as INPUT_DIR

FILENAME = 'histsoc_landuse-15crops_annual_FR_DE_DEMO_2001_2005.nc'
FILENAME_MEAN = 'hist_mean_mai-firr_1976-2005_DE_FR.hdf5'

exp = CropProduction()
exp.set_from_isimip_netcdf(input_dir=INPUT_DIR, filename=FILENAME, hist_mean=FILENAME_
↪MEAN,
                                    bbox=[-5, 42, 16, 55], yearrange=(2001, 2005),␣
↪crop='mai',
                                    scenario='flexible', unit='t/y', irr='firr')
"""compute maize crop production..."""
norm=colors.LogNorm(vmin=1e2, vmax=3e5)
exp.plot_basemap(norm=norm, pop_name=False) # warning: slow to plot basemap
# exp.plot_scatter(norm=norm, s=50) # faster

exp.set_value_to_usd(INPUT_DIR)
"""compute USD value (with prices from FAO)..."""
norm=colors.LogNorm(vmin=1e2, vmax=5e7)
exp.plot_basemap(norm=norm, pop_name=False) # warning: slow to plot basemap
# exp.plot_scatter(norm=norm, s=50) # faster
```

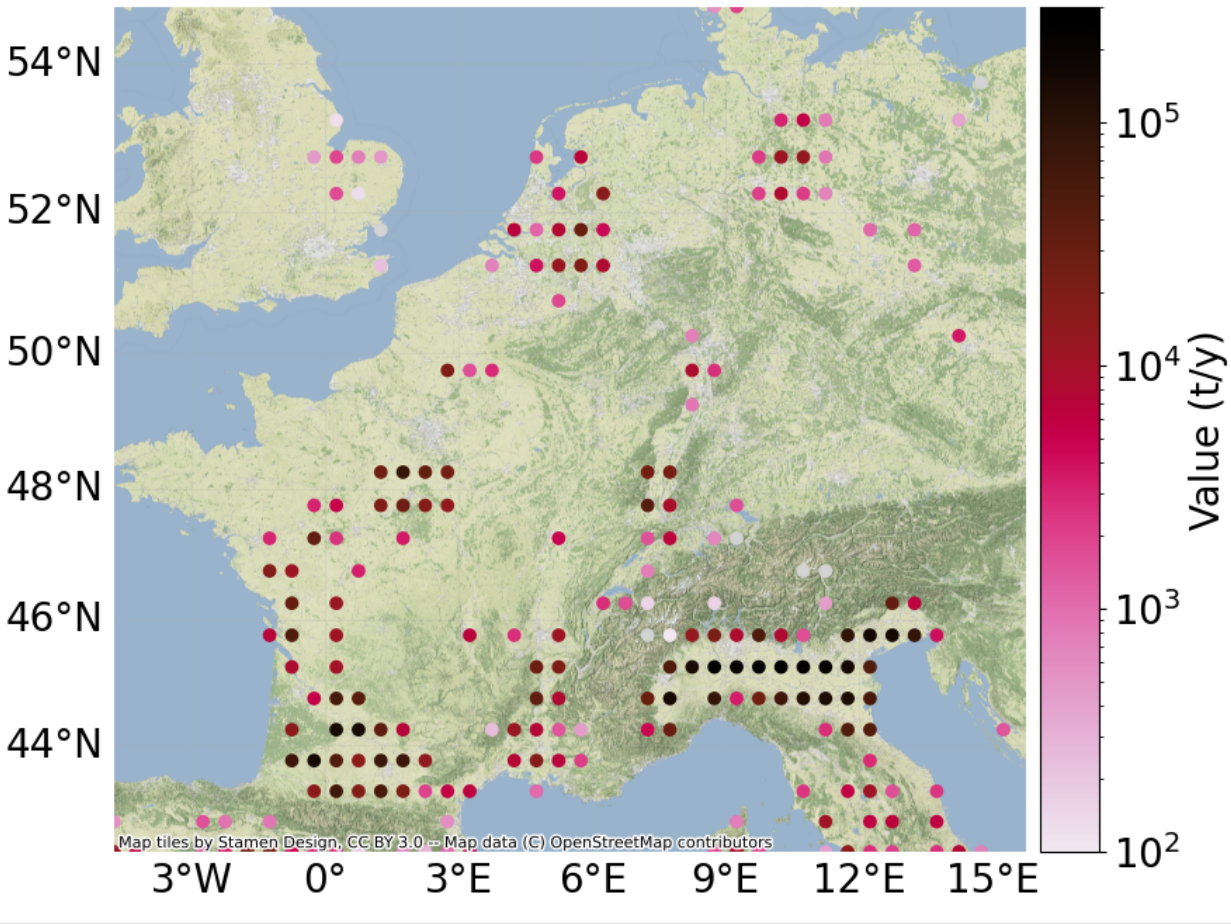
```
c:\users\me\git\climada_python\climada\util\coordinates.py:2747: FutureWarning: You␣
↪are adding a column named 'geometry' to a GeoDataFrame constructed without an␣
↪active geometry column. Currently, this automatically sets the active geometry␣
```

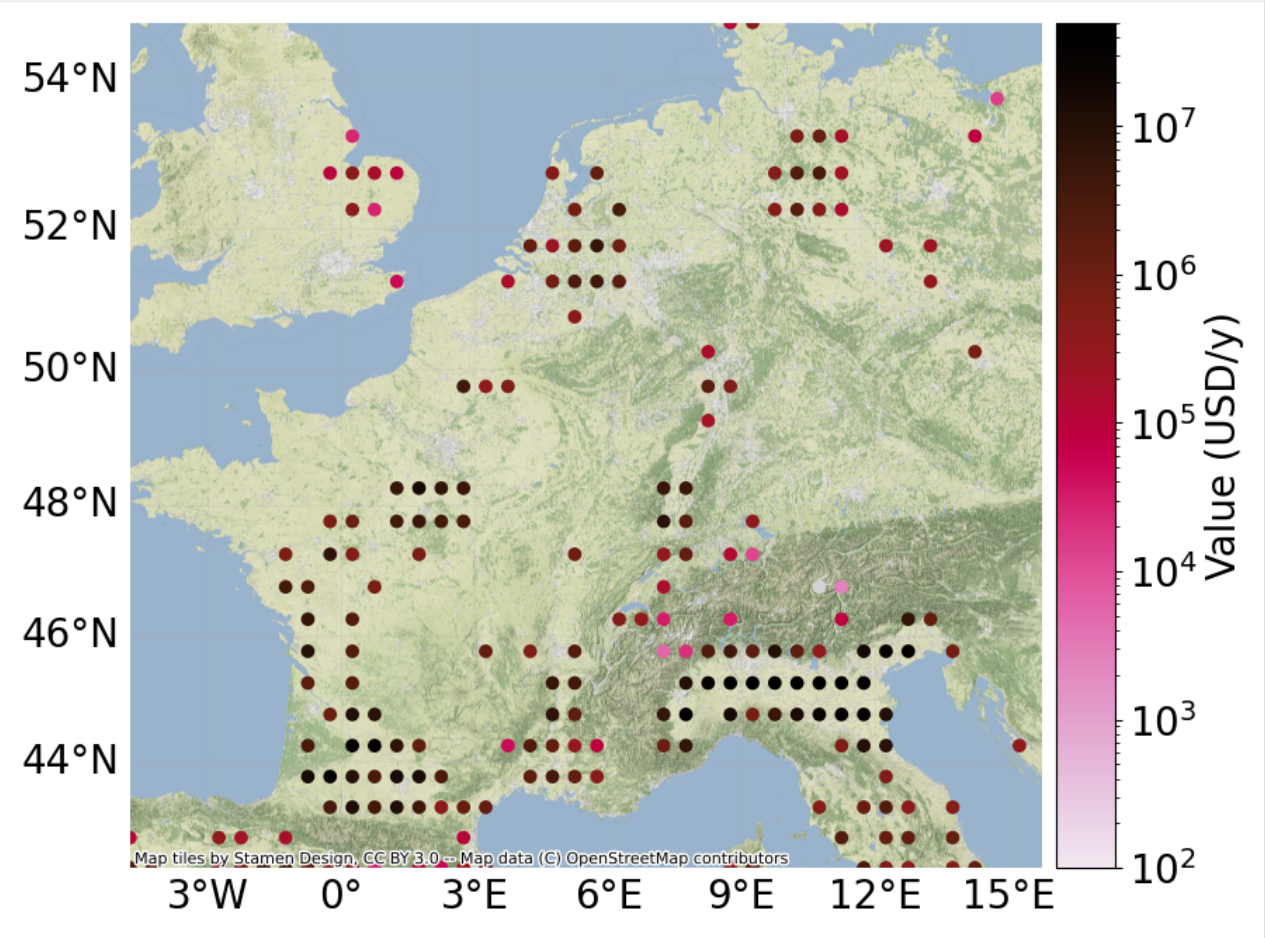
(continues on next page)

(continued from previous page)

```
↪column to 'geometry' but in the future that will no longer happen. Instead, either␣
↪provide geometry to the GeoDataFrame constructor (GeoDataFrame(…␣
↪geometry=GeoSeries()) or use `set_geometry('geometry')` to explicitly set the␣
↪active geometry column.
  df_val['geometry'] = gpd.GeoSeries(
```

[6]: <GeoAxes: >





8.4.2 Initiating an exposure set from several model runs:

Requires data download from https://esg.pik-potsdam.de/search/isimip/: - gepic_gfdl-esm2m_ewembi_historical_2005soc_co2_yield-ric-firr_global_annual_1861_2005.nc - gepic_gfdl-esm2m_ewembi_historical_2005soc_co2_yield-ric-noirr_global_annual_1861_2005.nc - histsoc_landuse-15crops_annual_1861_2005.nc

Requires data download from http://www.fao.org/faostat/en/#data/QC: - Countries: all; Items: "Rice, paddy"; Elements: "Production Quantity", Years: 2008 to 2018. - save as FAOSTAT_data_production_quantity.csv in your local input data directory (*input_exp_dir*)

Please note: when calling *set_multiple_rc_from_isimip()*. the historical mean (hist_mean) required here is averaged over all model combinations for each crop and irrigation type. This means that the exposure per crop type and irrigation type represents an average over all model ciombinations used.

Normalization:

It is possible to normalize the crop production per country with FAO data using *normalize_with_fao_cp()* and *normalize_several_exp()* for multiple exposure pairs. The normalization follows three steps: 1. Total crop production per crop type (full irrigation + no irrigation) is summed for each country (model crop production). 2. An average crop production per crop type and country is extracted from FAO statistics (reported crop production). 3. Crop production is normalized by multiplying the values of each grid cell with the ratio of reported over model crop production.

As a result of normalization, crop production summed over a country and both irrigation types is equal to the avaerage reported crop production.

[ ]:
```
import numpy as np
from pathlib import Path
import matplotlib.pyplot as plt

from climada.util.constants import CONFIG
import climada.util.coordinates as u_coord
from climada_petals.hazard.relative_cropyield import set_multiple_rc_from_isimip
from climada_petals.entity.exposures.crop_production import init_full_exp_set_isimip,␣
↪normalize_several_exp

data_path = CONFIG.local_data.save_dir.dir() / 'ISIMIP_crop' # set path of working␣
↪data directory
input_haz_dir = data_path / "Input" / "Hazard_tutorial" # set path where you place␣
↪hazard input data
# (Place crop yield data (.nc) from ISIMIP in input_haz_dir)
input_exp_dir = data_path / "Input" / "Exposure" # save FAO data and histsoc_landuse-
↪15crops_annual_1861_2005.nc here.

output_dir = data_path / "Output_tutorial" # set output directory
path_hist_mean = output_dir / 'Hist_mean' # set output directory for hist_mean

# # only required if hazard set has not yet been initiated above:
#-----------------------------------------------------------------
# set_multiple_rc_from_isimip(input_dir=input_haz_dir, output_dir=output_dir)
# """compute historical mean yield for all runs available in input_haz directory"""
#-----------------------------------------------------------------

filelist_exp, exposures = init_full_exp_set_isimip(input_dir=input_exp_dir, hist_mean_
↪dir=path_hist_mean, \
                                                   output_dir=output_dir, return_
↪data=True)
"""create exposures for all hist_mean files available in path_hist_mean directory"""
print("\nExposure files created:\n")
print(filelist_exp)

norm=colors.LogNorm(vmin=1e2, vmax=3e5)
exposures[0].plot_scatter(norm=norm, s=20, pop_name=False)
exposures[1].plot_scatter(norm=norm, s=20, pop_name=False)
"""For each crop type, an exposure with full irrigation crop production and one with␣
↪no irrigation is created."""

crop_list, countries_list, ratio_list, exp_firr_norm, exp_noirr_norm, fao_cp_list,␣
↪exp_tot_cp_list = \
    normalize_several_exp(input_dir=input_exp_dir, output_dir=output_dir,
                          yearrange=(2008, 2018),
                          unit='t/y', returns='all')
"""normalize crop production per country using FAO data"""
```

(continues on next page)

(continued from previous page)

```
exp_noirr_norm[0].plot_scatter(norm=norm, s=20, pop_name=False)
exp_firr_norm[0].plot_scatter(norm=norm, s=20, pop_name=False)

fig_scatter = plt.figure(facecolor='w', figsize=(7, 7))
ax_s = fig_scatter.add_subplot(1,1,1)
ax_s.scatter(exp_tot_cp_list, fao_cp_list)
ax_s.plot([0,2e8], [0,2e8], alpha=.5)
index_max_fao = np.where(fao_cp_list[0]==np.nanmax(fao_cp_list[0]))[0][0]
index_max_isimip = np.where(exp_tot_cp_list[0]==np.nanmax(exp_tot_cp_list[0]))[0][0]
# print(ratio_list[0])

ax_s.text(exp_tot_cp_list[0][index_max_fao], fao_cp_list[0][index_max_fao],
          u_coord.country_to_iso(countries_list[0][index_max_fao], 'name'))
ax_s.text(exp_tot_cp_list[0][index_max_isimip], fao_cp_list[0][index_max_isimip],
          u_coord.country_to_iso(countries_list[0][index_max_isimip], 'name'))
ax_s.set_title('Rice: total crop production (CP) per country')
ax_s.set_xlabel('CP before normalization [t/y]')
ax_s.set_ylabel('FAO statistics (used for normalization) [t/y]')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[2], line 24
     16 path_hist_mean = output_dir / 'Hist_mean' # set output directory for hist_mean
     18 # # only required if hazard set has not yet been initiated above:
     19 #-----------------------------------------------------------------
     20 # set_multiple_rc_from_isimip(input_dir=input_haz_dir, output_dir=output_dir)
     21 # """compute historical mean yield for all runs available in input_haz␣
↪directory"""
     22 #-----------------------------------------------------------------
---> 24 filelist_exp, exposures = init_full_exp_set_isimip(input_dir=input_exp_dir,␣
↪hist_mean_dir=path_hist_mean, \
     25                                                    output_dir=output_dir,␣
↪return_data=True)
     26 """create exposures for all hist_mean files available in path_hist_mean␣
↪directory"""
     27 print("\nExposure files created:\n")

File c:\users\me\git\climada_petals\climada_petals\entity\exposures\crop_production.
↪py:896, in init_full_exp_set_isimip(input_dir, filename, hist_mean_dir, output_dir,␣
↪bbox, yearrange, unit, isimip_version, return_data)
    893 if not unit:
    894     unit = 't/y'
--> 896 filenames = [f.name for f in hist_mean_dir.iterdir()
    897              if f.is_file() and not f.name.startswith('.')]
    899 # generate output directory if it does not exist yet
    900 target_dir = output_dir / 'Exposure'

File c:\users\me\git\climada_petals\climada_petals\entity\exposures\crop_production.
↪py:896, in <listcomp>(.0)
    893 if not unit:
    894     unit = 't/y'
--> 896 filenames = [f.name for f in hist_mean_dir.iterdir()
    897              if f.is_file() and not f.name.startswith('.')]
    899 # generate output directory if it does not exist yet
```

(continues on next page)

(continued from previous page)

```
    900 target_dir = output_dir / 'Exposure'

File c:\Users\me\miniconda3\envs\env_climada\lib\pathlib.py:1160, in Path.
↪iterdir(self)
   1156 def iterdir(self):
   1157     """Iterate over the files in this directory.  Does not yield any
   1158     result for the special paths '.' and '..'.
   1159     """
-> 1160     for name in self._accessor.listdir(self):
   1161         if name in {'.', '..'}:
   1162             # Yielding a path object for these makes little sense
   1163             continue

FileNotFoundError: [WinError 3] The system cannot find the path specified: 'c:\\Users\
↪\me\\git\\climada_petals\\doc\\tutorial\\results\\ISIMIP_crop\\Output_tutorial\\
↪Hist_mean'
```

8.5 Impact: Deviation in yearly crop production

8.5.1 Computing impact from single file (end-to-end; demo data):

Relative crop yield and historical crop production is combined to calculate the deviation of crop production from the historical mean production.

The impact function (*ImpfRelativeCropyield*) corresponds to a simple multiplication of hazard intensity (relative yield) with exposure (baseline production). As a result, the impact represents the deviation of yearly production from the exposure value.

There are positive and negative impact values. Positive values represent a crop production surplus. Negative values represent a deficit.

[ ]:
```
import os
import matplotlib.pyplot as plt
from matplotlib import colors
from climada_petals.entity.exposures.crop_production import CropProduction
from climada_petals.hazard.relative_cropyield import RelativeCropyield
from climada.util.constants import DEMO_DIR as INPUT_DIR
from climada.entity import ImpactFuncSet, ImpfRelativeCropyield
from climada.engine import Impact

FN_STR_DEMO = 'annual_FR_DE_DEMO'
FILENAME_LU = 'histsoc_landuse-15crops_annual_FR_DE_DEMO_2001_2005.nc'
FILENAME_MEAN = 'hist_mean_mai-firr_1976-2005_DE_FR.hdf5'

yearrange_haz = (2001, 2005) # yearrange for hazard (demo data only available from␣
↪2001 to 2005)
yearrange_hist_mean = (2001, 2005) # yearrange for reference historical mean (demo␣
↪data only available from 2001 to 2005)
haz = RelativeCropyield()
haz.set_from_isimip_netcdf(input_dir=INPUT_DIR, yearrange=yearrange_haz, ag_model=
↪'lpjml',
                           cl_model='ipsl-cm5a-lr', scenario='historical', soc='2005soc',
                           co2='co2', crop='whe', irr='noirr', fn_str_var=FN_STR_DEMO)
hist_mean = haz.calc_mean(yearrange_hist_mean) # requires reference year range as␣
```

(continues on next page)

(continued from previous page)

```
↪input
"""compute historical mean yield per grid cell for reference (base line)"""
haz.set_rel_yield_to_int(hist_mean)

exp = CropProduction()
exp.set_from_isimip_netcdf(input_dir=INPUT_DIR, filename=FILENAME_LU, hist_
↪mean=FILENAME_MEAN,
                                    bbox=[-5, 42, 16, 55], yearrange=(2001, 2005),
                                    scenario='flexible', unit='t/y', irr='firr')
exp.set_value_to_usd(INPUT_DIR) # convert exposure from t/y to USD/y using FAO␣
↪statistics
exp.assign_centroids(haz, threshold=20) # assign exposure points to centroids
"""Init hazard and exposure"""

impf_cp = ImpactFuncSet()
impf_def = ImpfRelativeCropyield()
impf_def.set_relativeyield()
impf_cp.append(impf_def)
impf_cp.check()
impf_def.plot()
"""Import impact function"""

impact_demo= Impact()
impact_demo.calc(exp, impf_cp, haz)
"""Calculate impact"""

fig_imp_demo = plt.figure(facecolor='w')
ax_imp_demo = fig_imp_demo.add_subplot(1,1,1)
ax_imp_demo.plot(impact_demo.event_id, impact_demo.at_event, 'g', lw=3)
ax_imp_demo.hlines(0, xmin=1, xmax=5, alpha=.85, ls=':')
ax_imp_demo.set_xticks(impact_demo.event_id)
ax_imp_demo.set_xticklabels(impact_demo.event_name)
ax_imp_demo.set_title('Impact: Maize production deviation (demo data)')
ax_imp_demo.set_xlabel('model year')
ax_imp_demo.set_ylabel('$\Delta$ Crop Production [%s]' %(exp.value_unit))
```

[ ]:

CHAPTER

NINE

TUTORIAL WARN MODULE

This tutorial shows how to use the `Warn` module to generate a warning, i.e., 2D map of coordinates with assigned warn levels. Operations, their order, and their influence (operations sizes, gradual decrease of warn levels, and changing of too small regions to surrounding) can be selected to generate the warning. The functionality of the `Warn` module of reducing heterogeneity in a map can be applied to different inputs, e.g., MeteoSwiss windstorm data (COSMO data), `TC`s, `Impact`s, etc. The Warn module can also be used to cluster data visualized on a map. The master's thesis corresponding to this module (more information about operations) can be found here: 10.3929/ethz-b-000548850

[1]:
```
import numpy as np
import xarray as xr

from climada_petals.engine.warn import Warn, Operation
from climada.util.plot import geo_bin_from_array
from climada.entity import ImpfTropCyclone, ImpactFuncSet
from climada.util.api_client import Client


plotting_parameters = dict()
plotting_parameters['cmap'] = 'Wistia'
```

CHAPTER

TEN

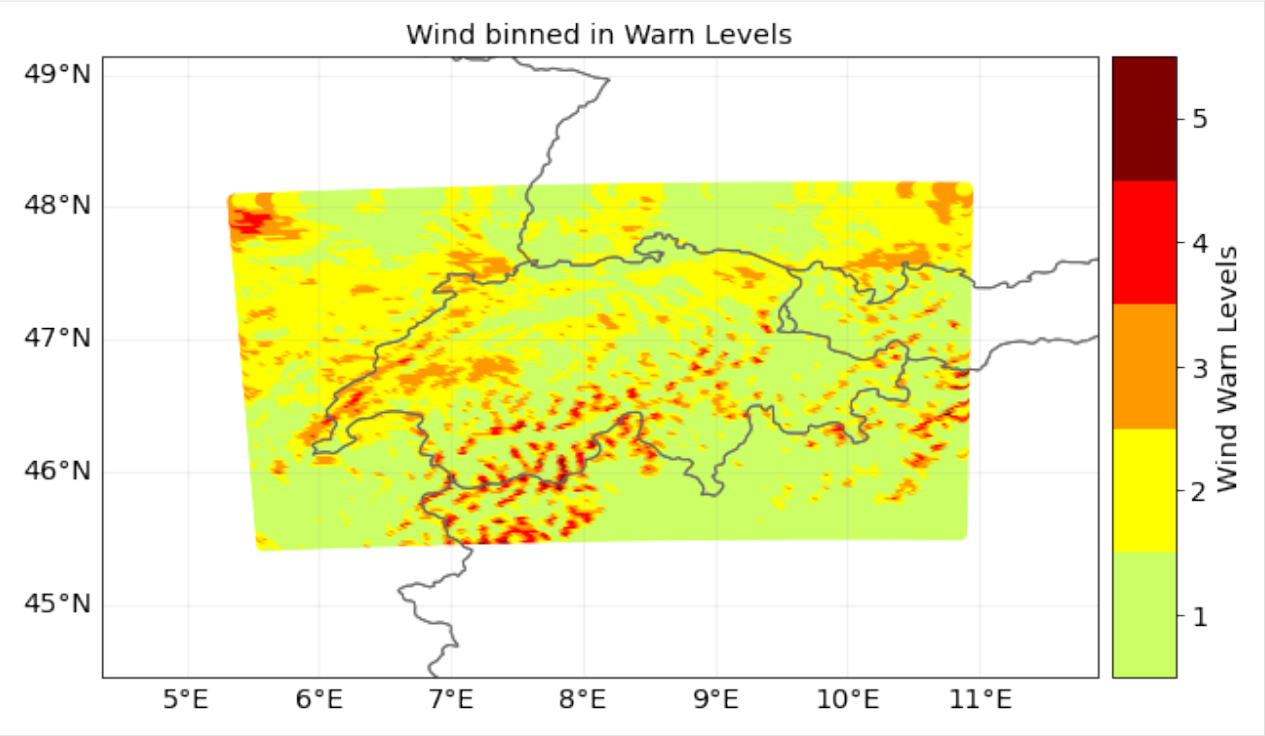
METEOSWISS STORM EXAMPLE

The first example is to generate a warn map of numerical weather predictions, in this case a wind storm prognose computed by MeteoSwiss.

[2]:
```
# Load MeteoSwiss storm example via client (COSMO2-E model)
client = Client()
dataset = client.get_dataset_info(name="cosmo2_2019121400")
path_files = client.download_dataset(dataset)[1][0]
ncdf = xr.open_dataset(path_files)
wind_matrix = ncdf.VMAX_10M.values[0, :, :]  # take one ensemble member of the␣
↪numerical weather model COSMO

lon = ncdf.lon_1.values
lat = ncdf.lat_1.values
coord = np.vstack((lat.flatten(), lon.flatten())).transpose()
```

To show the raw data first, no operations are applied. This shows then the binned data only.

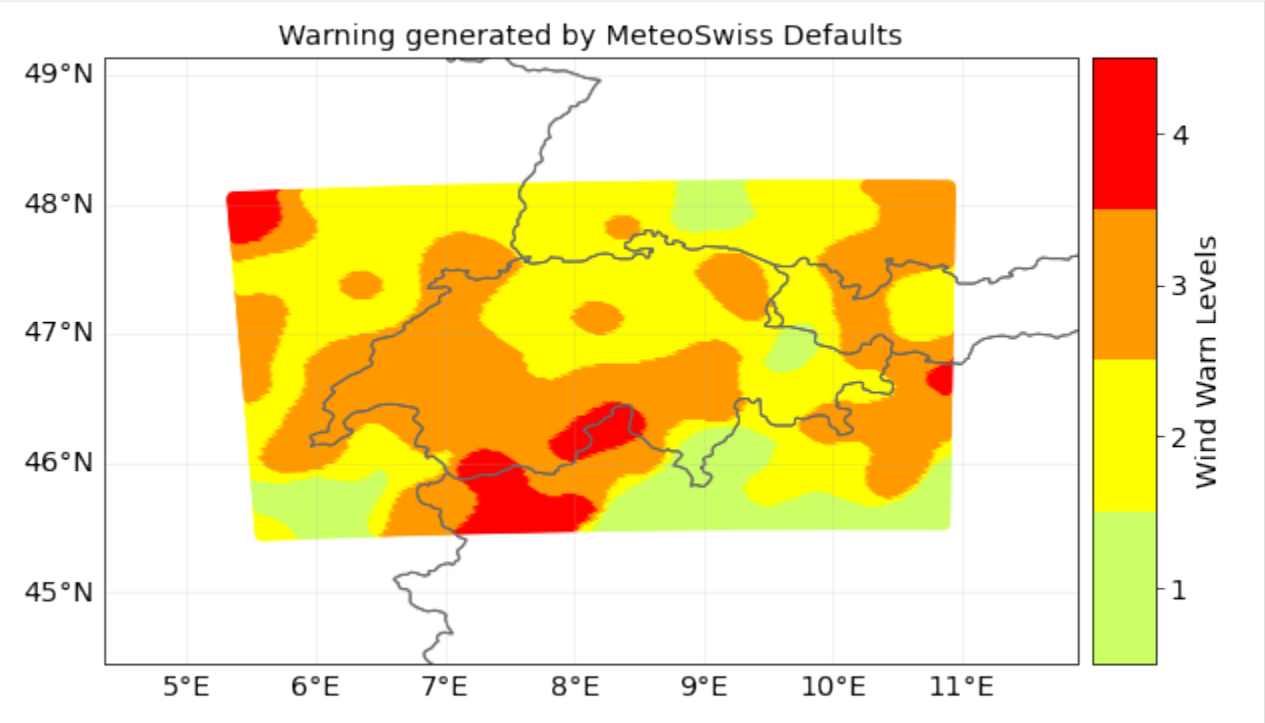
[3]:
```
# Define warn levels for this example
warn_levels = np.array([0.0, 19.44, 25.0, 30.55, 38.88, 300.0])

# Define warning parameters, such that the data is only binned in warn levels␣
↪(without operations)
warn_params_only_binning = Warn.WarnParameters(warn_levels, operations=[])
binned_map = Warn.from_map(wind_matrix, coord, warn_params_only_binning)
binned_map.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title='Wind␣
↪binned in Warn Levels');
```



10.1 Demonstrate Warning Generation

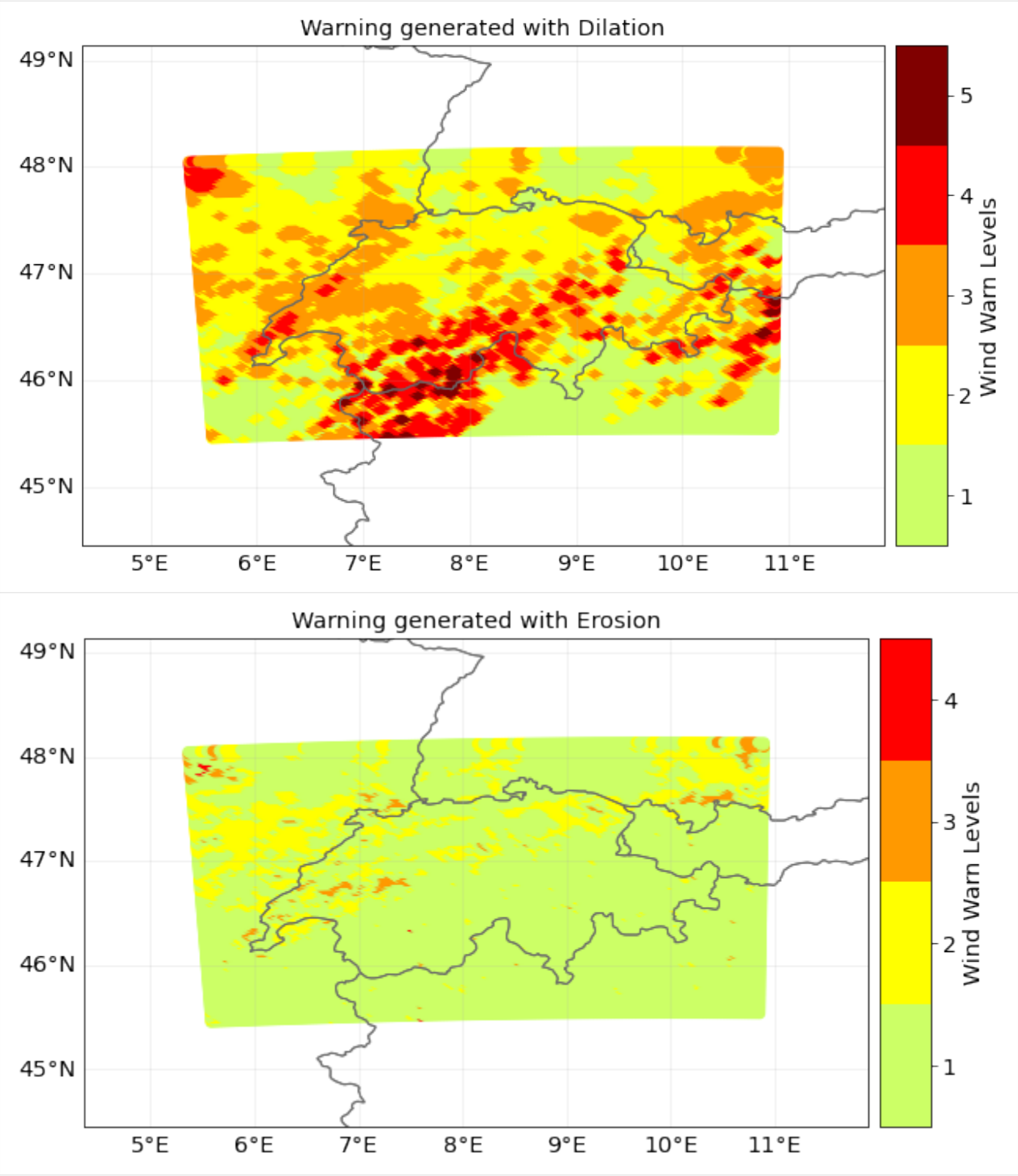
The first example shows a default selection of operations and their properties.

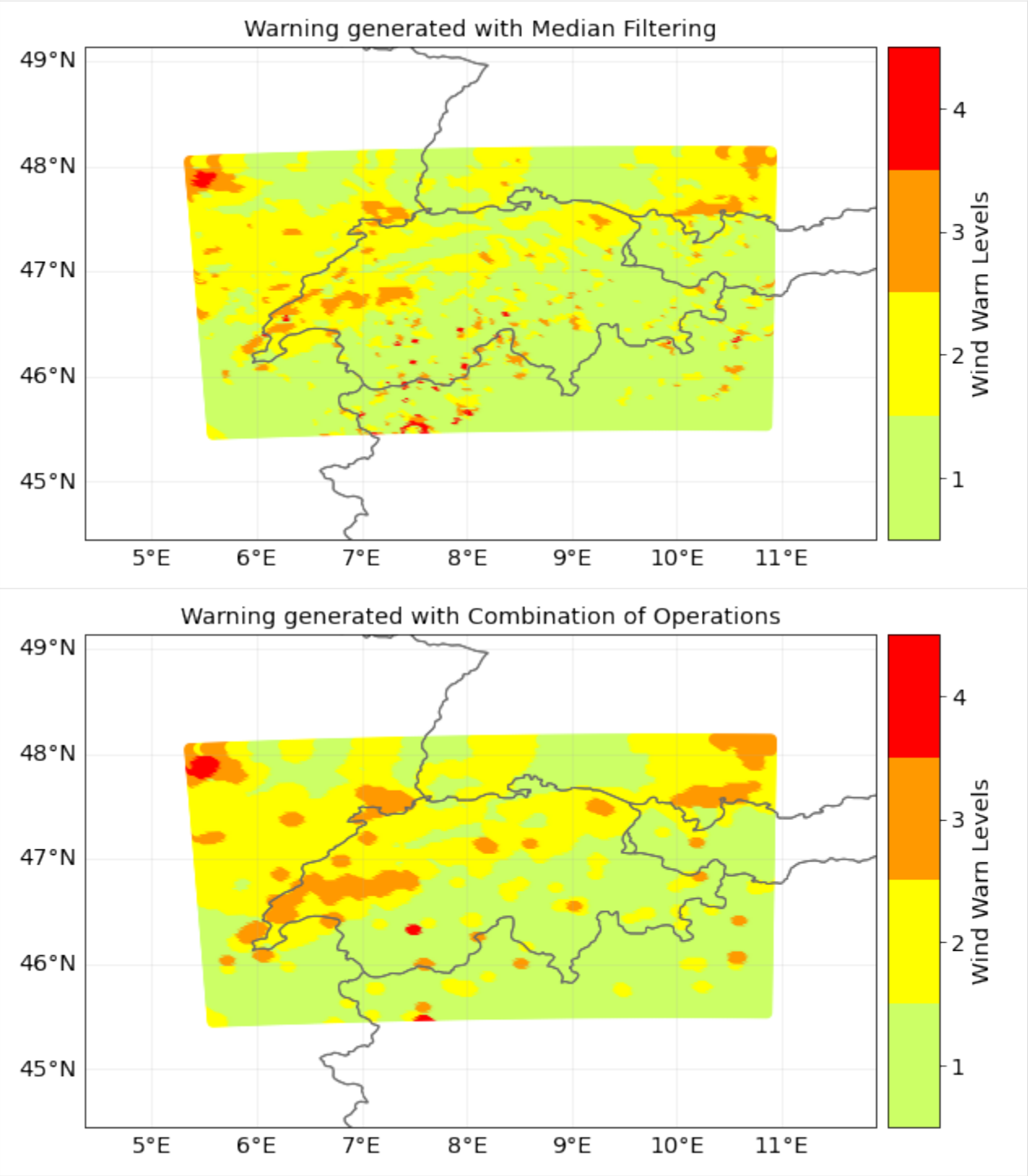
[4]:
```
# Define warning parameters, such that the data is generated with default parameters
warn_params_default = Warn.WarnParameters(warn_levels)
default_op_warning = Warn.from_map(wind_matrix, coord, warn_params_default)
default_op_warning.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title=
↪'Warning generated by '
                                                 'MeteoSwiss␣
↪Defaults');
```



When generating a warning, operations and their sizes can be selected by the users. This has an impact on what the warning looks like. The different operations and a possible combination of them can be seen below.

[5]:
```
# only dilation - expands areas in warn levels above 0
warn_params = Warn.WarnParameters(warn_levels, operations=[(Operation.dilation, 2)])
warn_dilation_only = Warn.from_map(wind_matrix, coord, warn_params)
warn_dilation_only.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title=
↪'Warning generated with '
                                                 'Dilation');
# only erosion - reduces areas in warn levels above 0 and heterogeneity
warn_params = Warn.WarnParameters(warn_levels, operations=[(Operation.erosion, 1)])
warn_erosion_only = Warn.from_map(wind_matrix, coord, warn_params)
warn_erosion_only.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title=
↪'Warning generated with '
                                                'Erosion');
# only median filtering - smooths out all areas (best applied after forming regions)
warn_params = Warn.WarnParameters(warn_levels, operations=[(Operation.median_
↪filtering, 3)])
warn_median_only = Warn.from_map(wind_matrix, coord, warn_params)
warn_median_only.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title=
↪'Warning generated with Median '
                                               'Filtering');
# first ersoion, then dilation, then median filtering
warn_params = Warn.WarnParameters(warn_levels, operations=[(Operation.erosion, 1),
                                                           (Operation.dilation, 3),
                                                           (Operation.median_
↪filtering, 3)])
warn_combination = Warn.from_map(wind_matrix, coord, warn_params)
warn_combination.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title=
↪'Warning generated with '
                                               'Combination of␣
↪Operations');
```


Warning generated with Dilation
49°N
48°N
47°N
46°N
45°N
5°E
6°E
7°E
8°E
9°E
10°E
11°E
Wind Warn Levels
1
2
3
4
5
Warning generated with Erosion
49°N
48°N
47°N
46°N
45°N
5°E
6°E
7°E
8°E
9°E
10°E
11°E
Wind Warn Levels
1
2
3
4



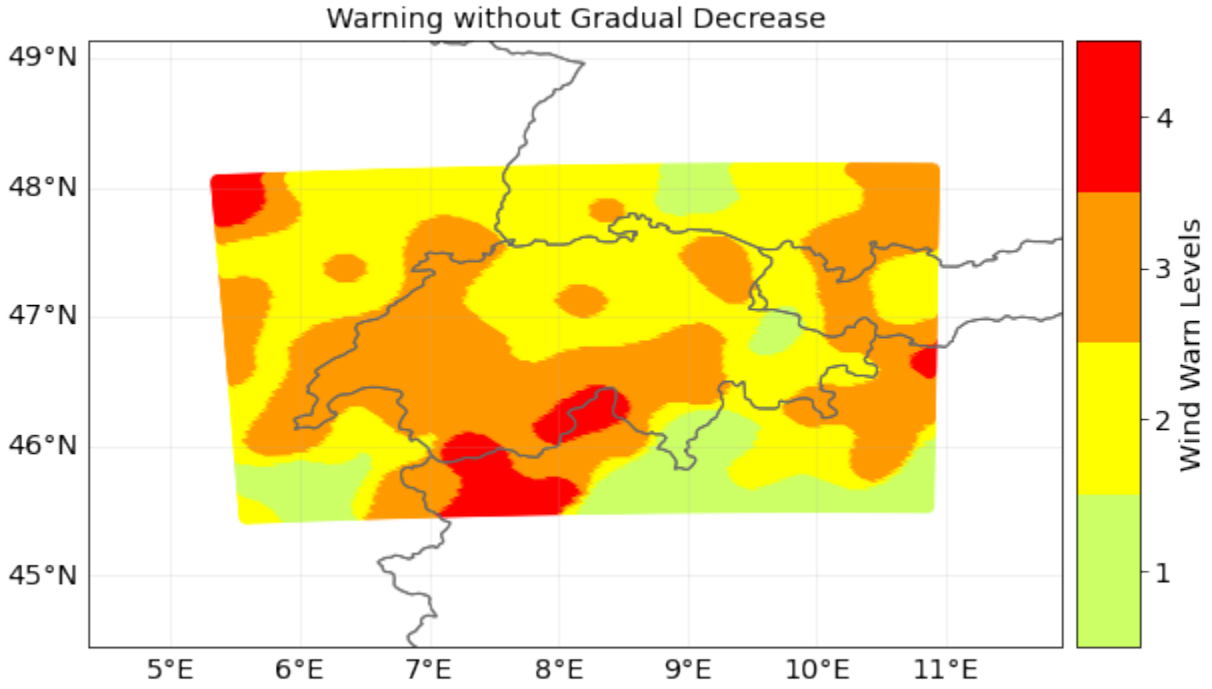
Next to selecting the operations, one can select whether the highest warn levels should be gradually decreased by its neighboring regions (if True) to the lowest level (e.g., level 3, 2, 1, 0) or larger steps are allowed (e.g., from level 5 directly to 2). Since this parameter doesn't have a large influence here, study also the next example.

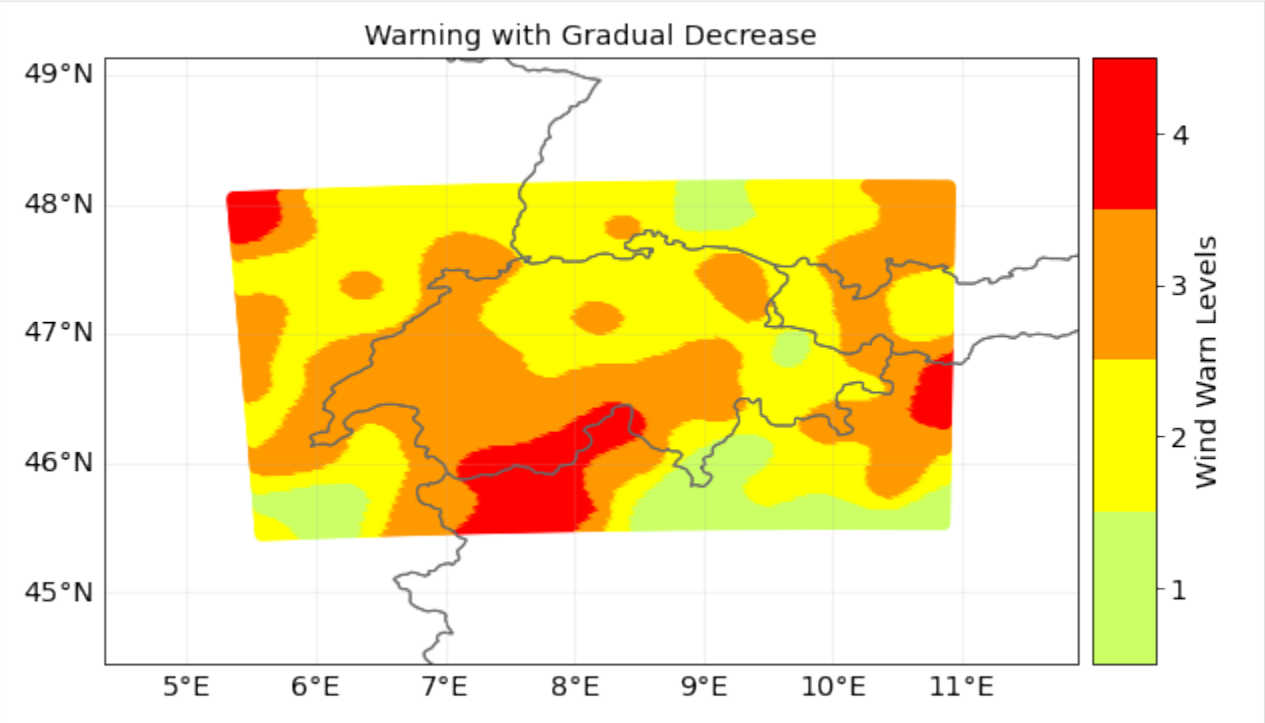
[6]:
```
# Warning without and with gradual decrease of regions of higher level to lower␣
↪levels to see difference to default parameters plot
warn_params = Warn.WarnParameters(warn_levels)
```

(continues on next page)

(continued from previous page)

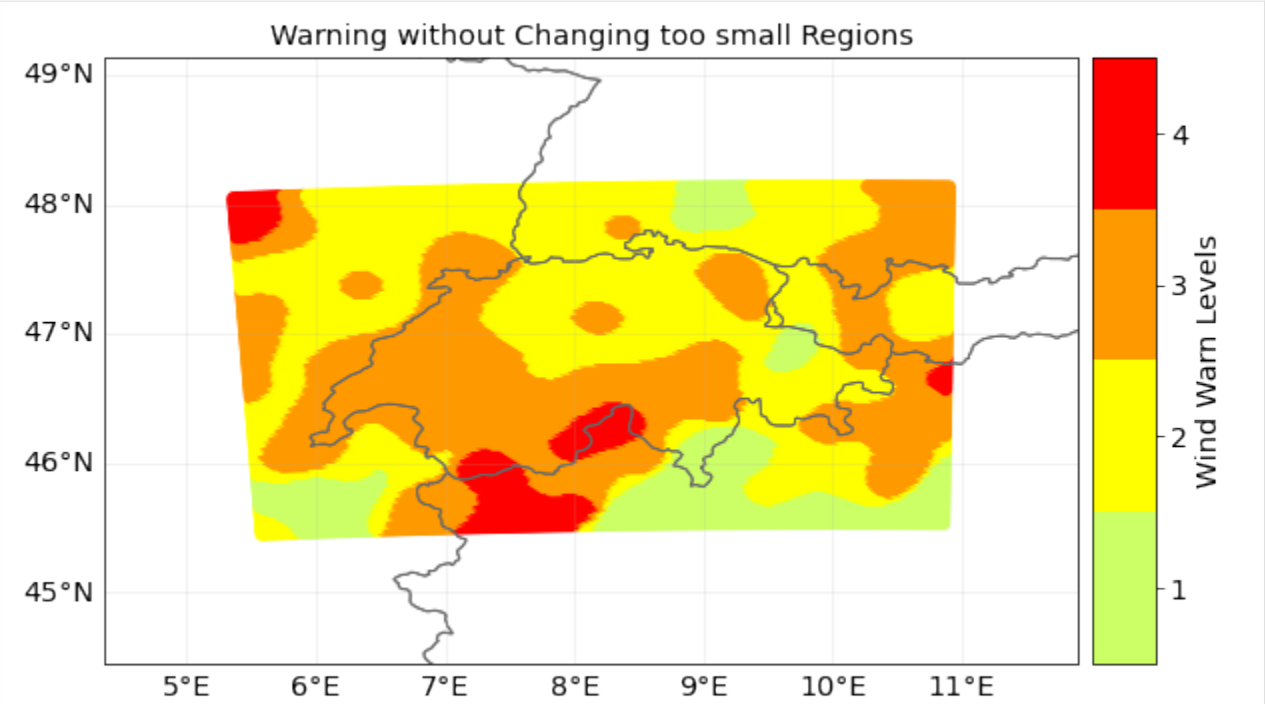
```
warn_wo_grad = Warn.from_map(wind_matrix, coord, warn_params)
warn_wo_grad.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title=
↪'Warning without Gradual Decrease');

warn_params = Warn.WarnParameters(warn_levels, gradual_decr=True)
warn_wo_grad = Warn.from_map(wind_matrix, coord, warn_params)
warn_wo_grad.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title=
↪'Warning with Gradual Decrease');
```

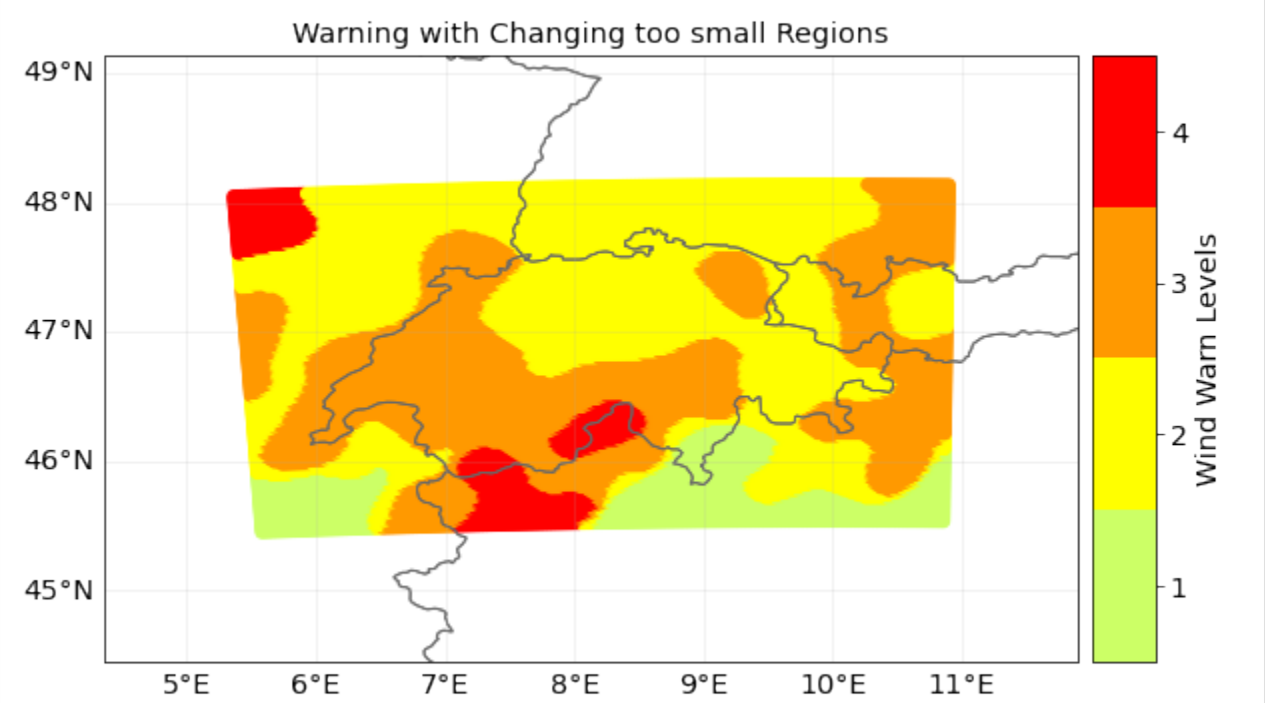




Further, small regions of a warn level formed by the algorithm can be changed to the surrounding warn level. Therefore, the parameter `change_sm` needs to be set to the number of coordinate below which a region is regarded as to small.

[7]:
```
# Warning without and with change too small regions by setting levels newly.
warn_params = Warn.WarnParameters(warn_levels)
warn_not_changed = Warn.from_map(wind_matrix, coord, warn_params)
warn_not_changed.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title=
↪'Warning without Changing too '
                                                'small Regions');

warn_params = Warn.WarnParameters(warn_levels, change_sm=350)
warn_changed = Warn.from_map(wind_matrix, coord, warn_params)
warn_changed.plot_warning_meteoswiss_style(var_name='Wind Warn Levels', title=
↪'Warning with Changing too small '
                                                'Regions');
```


Warning without Changing too small Regions
49°N
48°N
47°N
46°N
45°N
5°E
6°E
7°E
8°E
9°E
10°E
11°E
Wind Warn Levels
1
2
3
4


Warning with Changing too small Regions
49°N
48°N
47°N
46°N
45°N
5°E
6°E
7°E
8°E
9°E
10°E
11°E
Wind Warn Levels
1
2
3
4

CHAPTER

ELEVEN

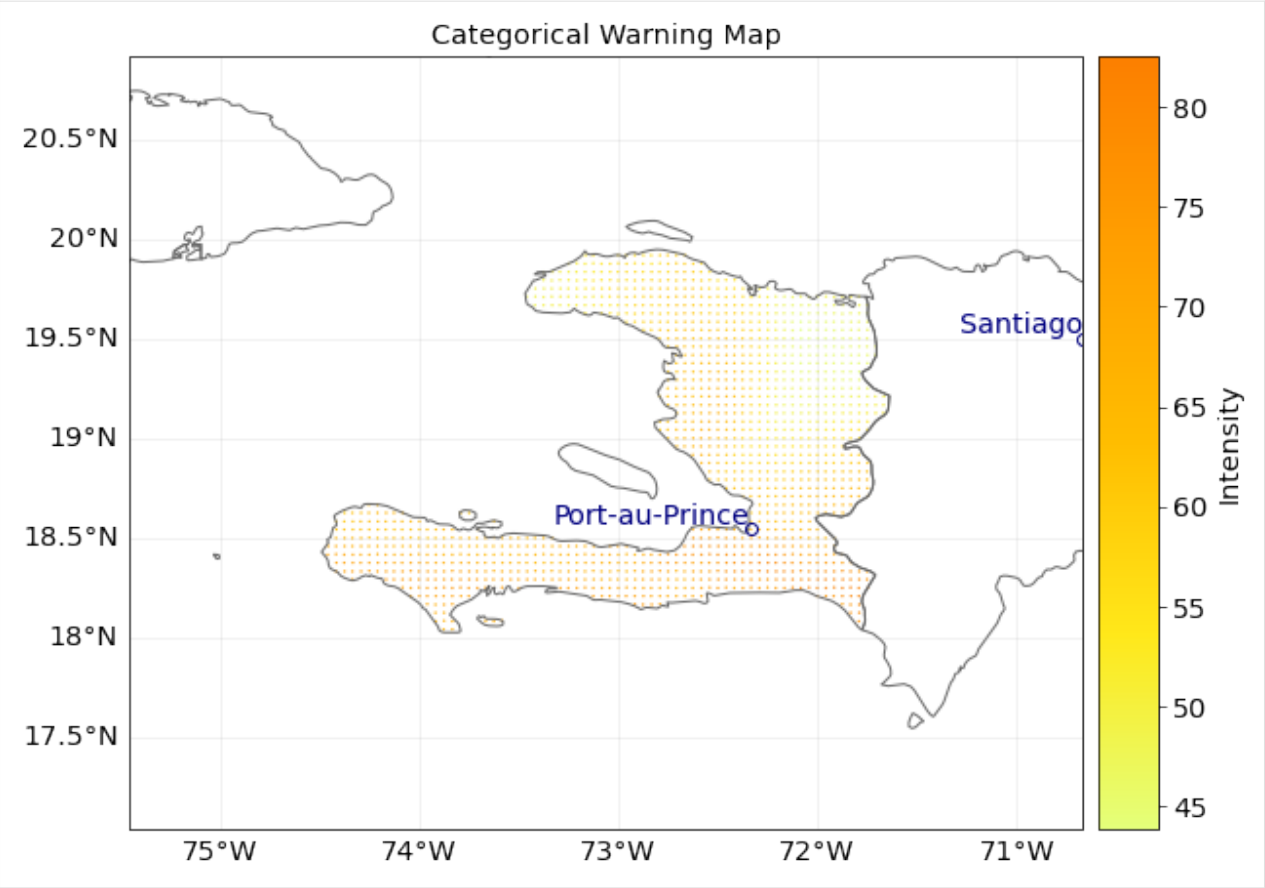
HAZARD EXAMPLE - TC HAITI

This example shows how the Warn module can be used to generate warnings of natural catastrophes, e .g., tropical cyclons.

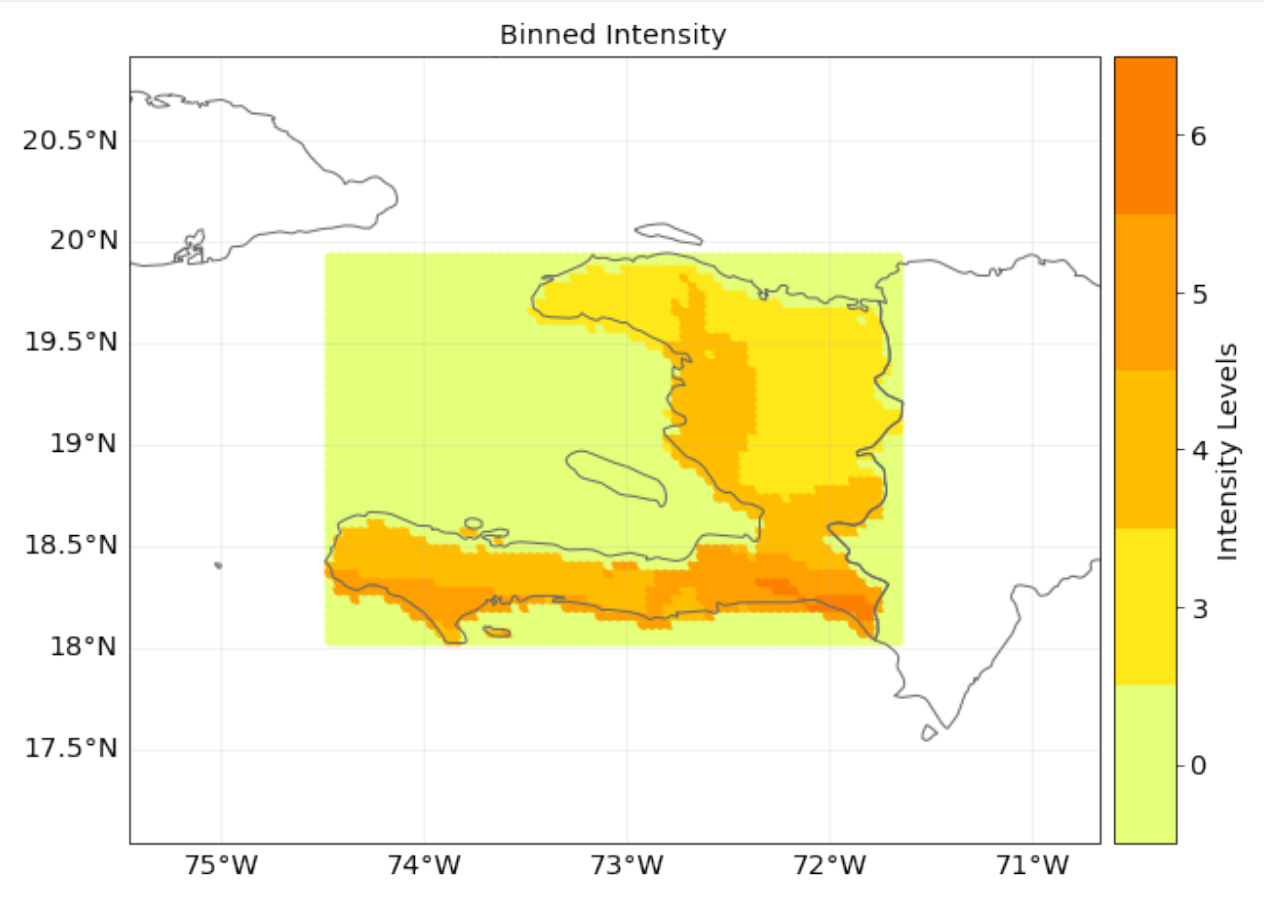
[8]:
```
tc_dataset_infos = client.list_dataset_infos(data_type='tropical_cyclone')
client.get_property_values(tc_dataset_infos, known_property_values = {'country_name':
↪'Haiti'});
```

[9]:
```
# Read hazard
tc_haiti = client.get_hazard('tropical_cyclone', properties={'country_name': 'Haiti',
↪'climate_scenario': 'rcp45', 'ref_year':'2040', 'nb_synth_tracks':'10'})
tc_haiti.intensity = tc_haiti.intensity.max(axis=0)

lon = tc_haiti.centroids.lon
lat = tc_haiti.centroids.lat
coord_haiti = np.vstack((lat.flatten(), lon.flatten())).transpose()

geo_bin_from_array(tc_haiti.intensity.todense().transpose(), coord_haiti, 'Intensity',
↪ 'Categorical Warning Map', **plotting_parameters);
```

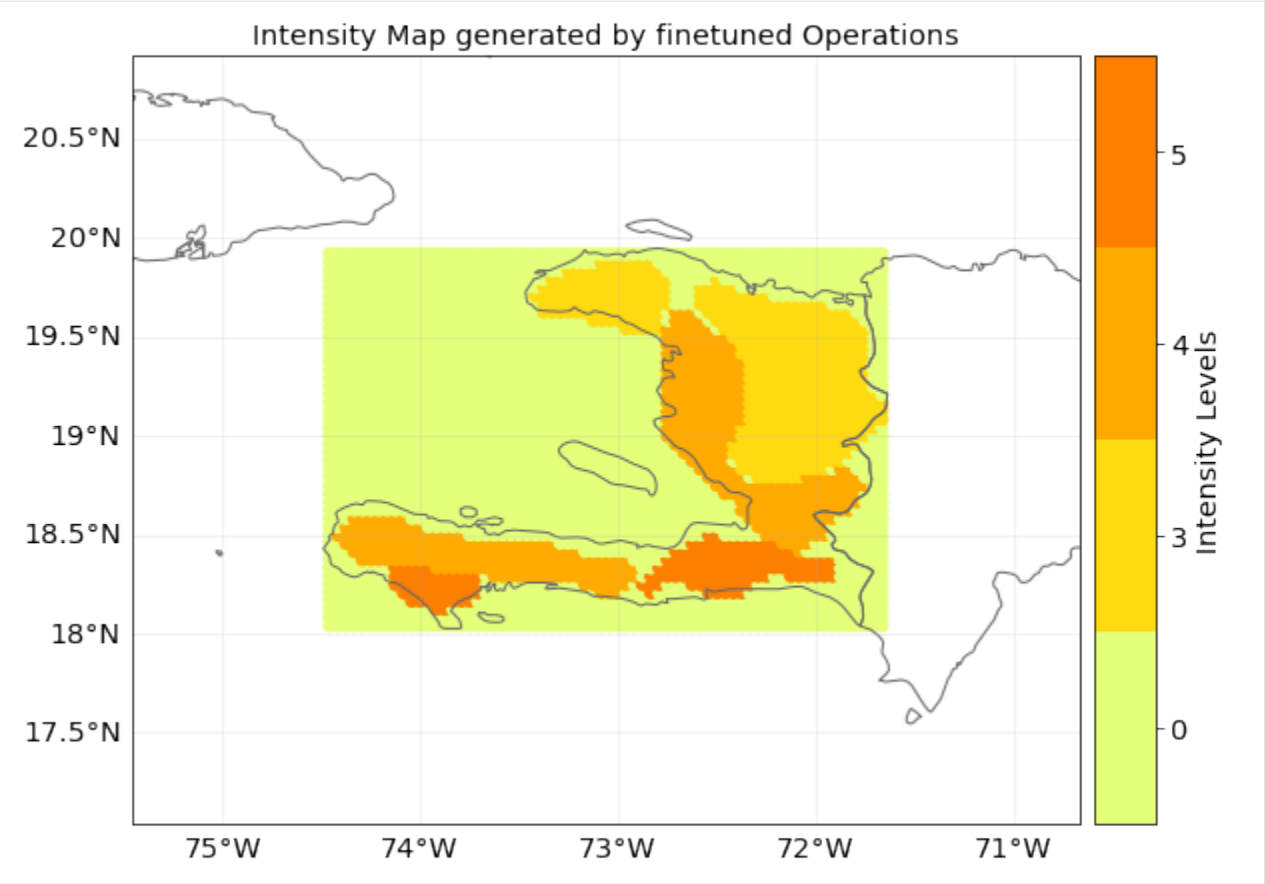
```
2022-05-25 14:25:10,374 - climada.hazard.base - INFO - Reading /Users/robertblass/
↪climada/data/hazard/tropical_cyclone/tropical_cyclone_10synth_tracks_150arcsec_
↪rcp45_HTI_2040/v1/tropical_cyclone_10synth_tracks_150arcsec_rcp45_HTI_2040.hdf5
```



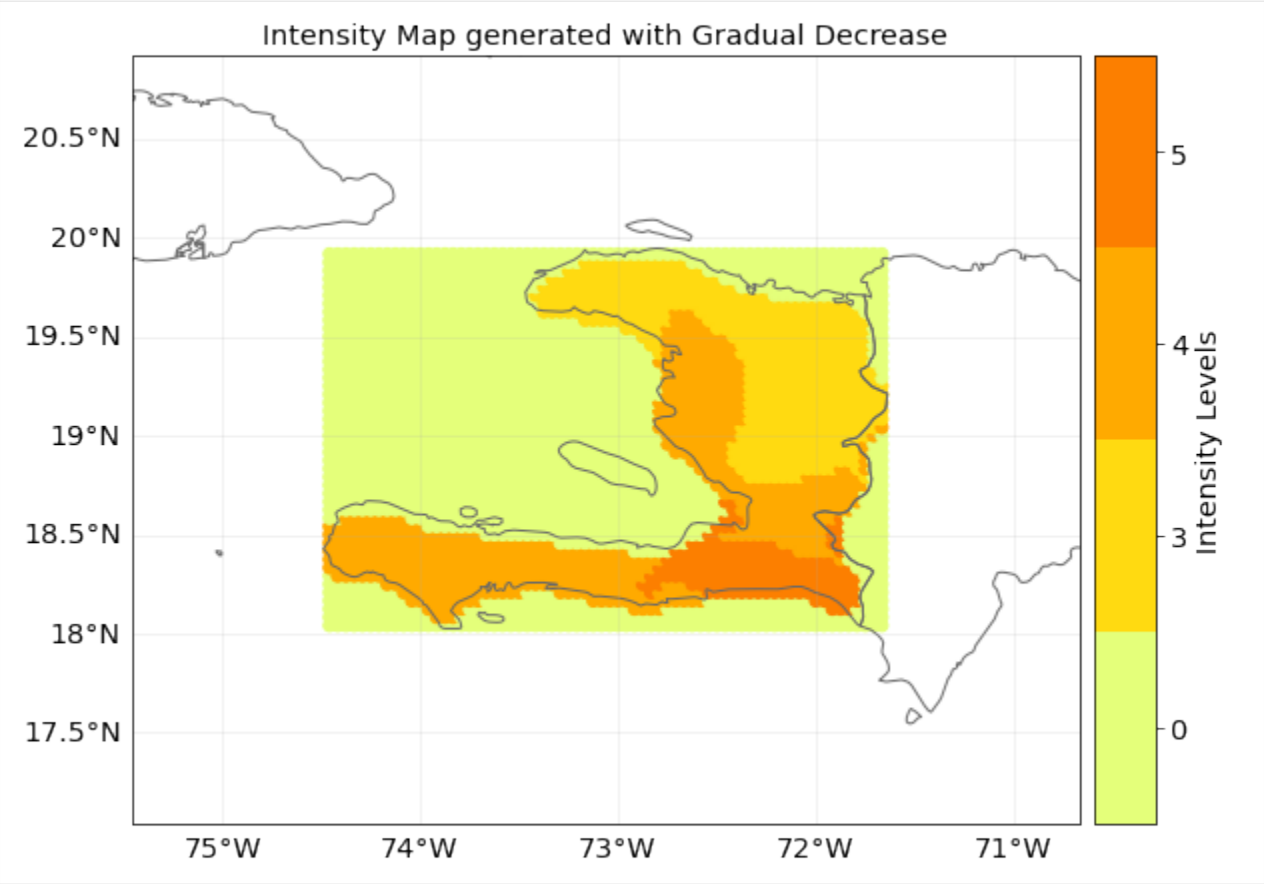
[10]:
```
# Plot binned intensities
# warn Levels
warn_levels = [0, 20, 30, 40, 60, 70, 80, 1000]
grid, coord_haiti = Warn.zeropadding(tc_haiti.centroids.lat, tc_haiti.centroids.lon,␣
↪tc_haiti
                                     .intensity.todense())

# no filtering operations applied - only binning as reference
warn_params_only_binning = Warn.WarnParameters(warn_levels, operations=[])
binned_only = Warn.from_map(grid, coord_haiti, warn_params_only_binning)
binned_only.plot_warning(var_name='Intensity Levels', title='Binned Intensity',␣
↪**plotting_parameters);
```



[11]:
```
# Apply fine tuned filtering operations
warn_params = Warn.WarnParameters(warn_levels, operations=[(Operation.erosion, 1),
                                                           (Operation.dilation, 1),
                                                           (Operation.median_
↪filtering, 5)])
warn_def = Warn.from_map(grid, coord_haiti, warn_params)
warn_def.plot_warning(var_name='Intensity Levels', title='Intensity Map generated by␣
↪finetuned Operations', **plotting_parameters);
```



[12]:
```
# Apply same operations with gradual decrease and changing too small regions to␣
↪surrounding
warn_params = Warn.WarnParameters(warn_levels, operations=[(Operation.erosion, 1),
                                                           (Operation.dilation, 1),
                                                           (Operation.median_
↪filtering, 5)],
                                  gradual_decr=True,
                                  change_sm=100)
warn_def = Warn.from_map(grid, coord_haiti, warn_params)
warn_def.plot_warning(var_name='Intensity Levels', title='Intensity Map generated␣
↪with Gradual Decrease', **plotting_parameters);
```


Intensity Map generated with Gradual Decrease
20.5°N
20°N
19.5°N
19°N
18.5°N
18°N
17.5°N
75°W
74°W
73°W
72°W
71°W
Intensity Levels
5
4
3
0

CHAPTER

TWELVE

IMPACT EXAMPLE - VALUE (USD) HAITI

This example shows how the Warn module can be used on any 2D map of values (here impact cost in USD). It can cluster and smooth the given data map.

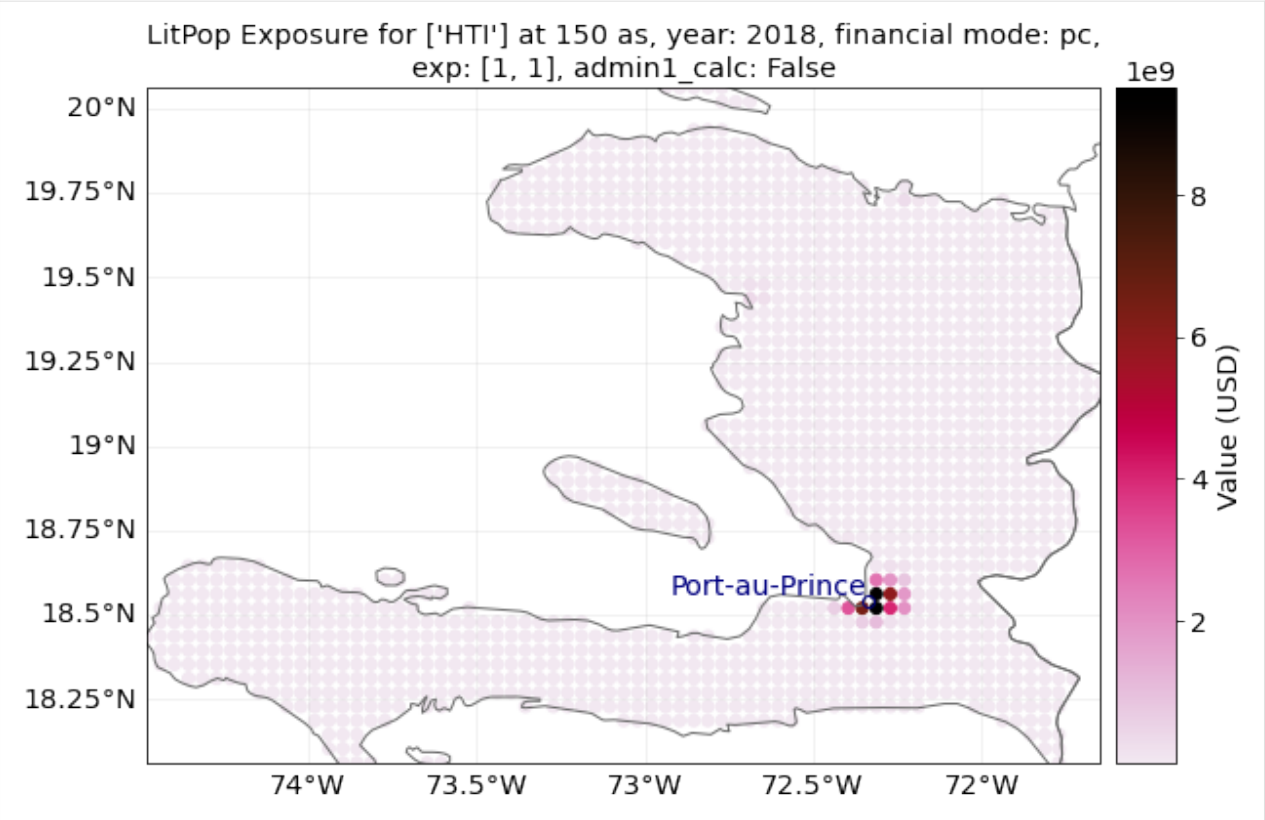
[13]:
```
# Get data
exp_haiti = client.get_litpop_default(country="Haiti");
```

```
2022-05-25 14:25:18,668 - climada.entity.exposures.base - INFO - Reading /Users/
↪robertblass/climada/data/exposures/litpop/LitPop_150arcsec_HTI/v1/LitPop_150arcsec_
↪HTI.hdf5
```

[14]:
```
impf = ImpfTropCyclone().from_emanuel_usa()
impf_set = ImpactFuncSet()
impf_set.append(impf)
```

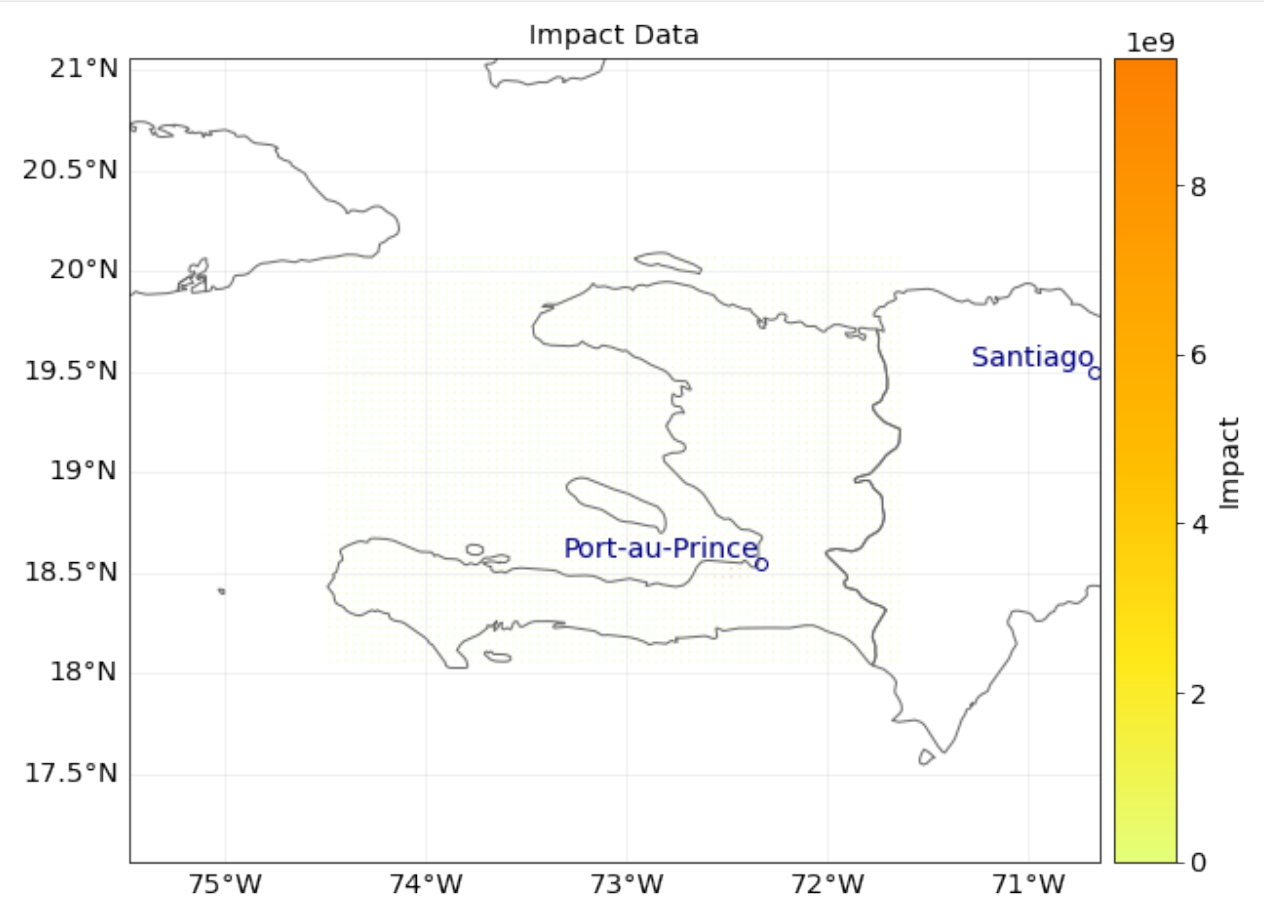
[15]:
```
# Plot exposures
exp_haiti.plot_scatter();
```

```
/Users/robertblass/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↪crs/crs.py:1256: UserWarning: You will likely lose important projection information␣
↪when converting to a PROJ string from another format. See: https://proj.org/faq.html
↪#what-is-the-best-format-for-describing-coordinate-reference-systems
  return self._crs.to_proj4(version=version)
```

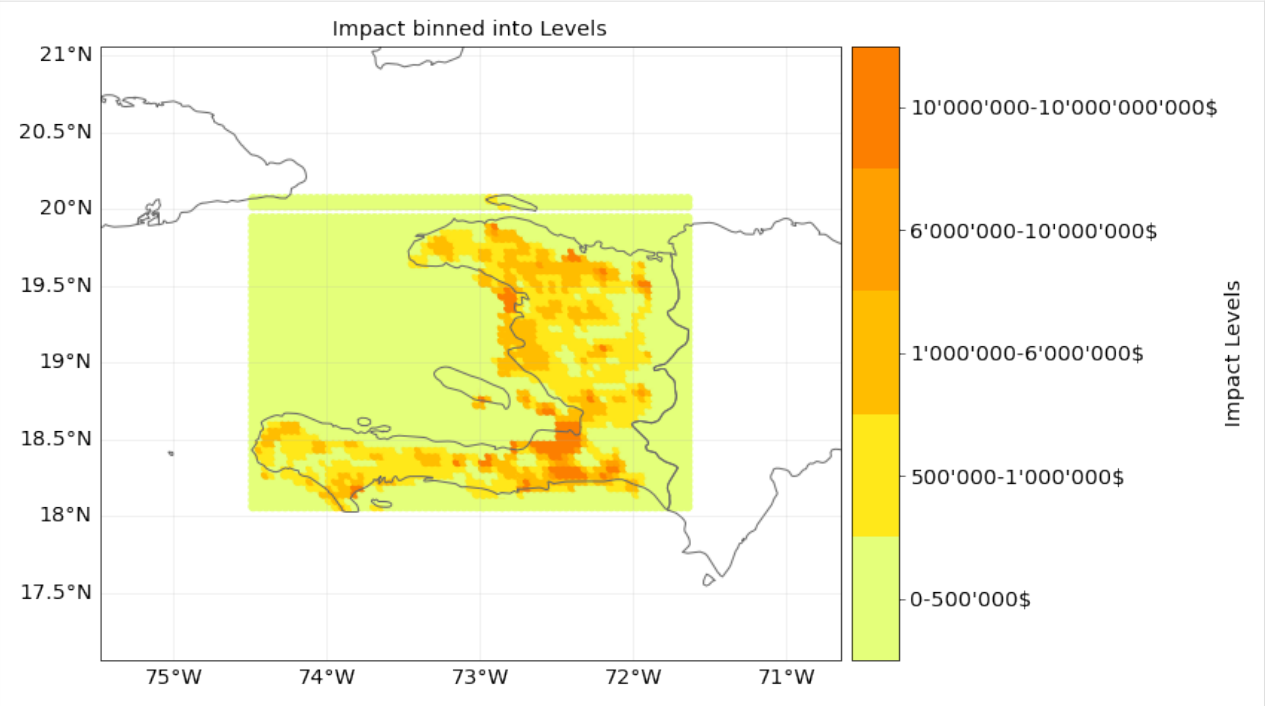


[16]:
```
# Project values to rectangle
lat, lon, values = exp_haiti.gdf.latitude.to_numpy(), exp_haiti.gdf.longitude.to_
↪numpy(), exp_haiti.gdf.value.to_numpy()
grid, coord_impf = Warn.zeropadding(lat, lon, values)
```

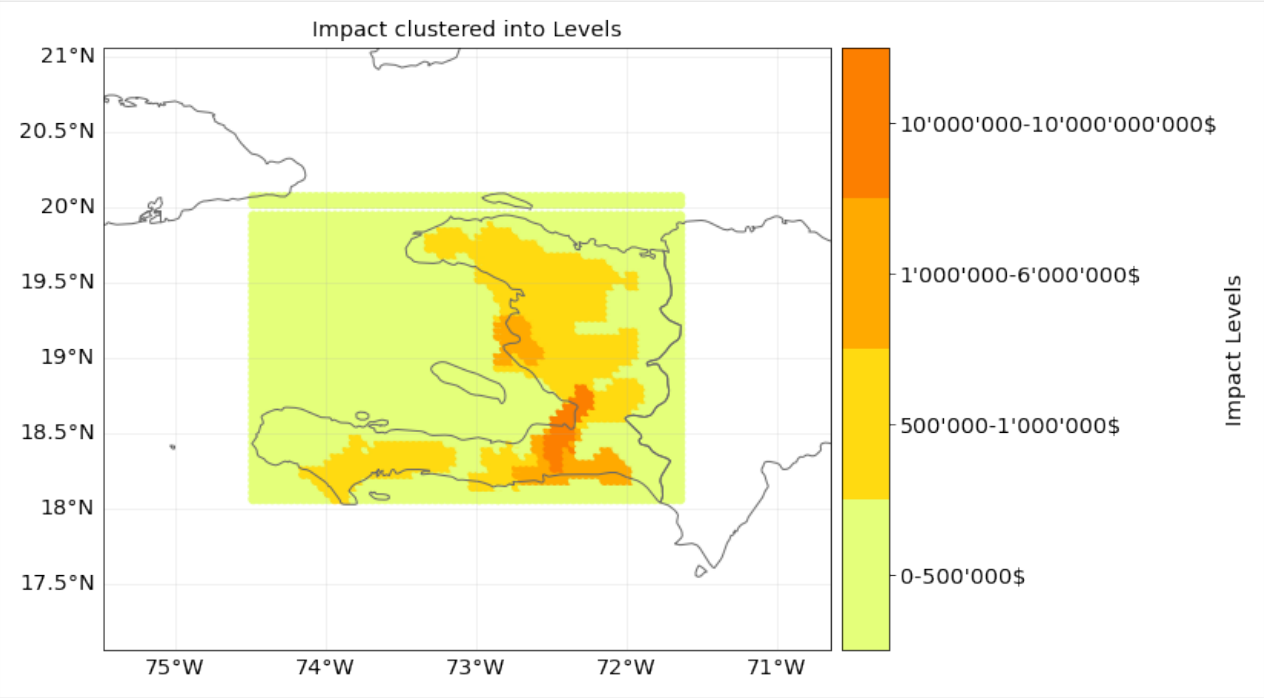
[17]:
```
# Plot raw data
geo_bin_from_array(grid.flatten(), coord_impf, 'Impact', 'Impact Data', **plotting_
↪parameters);
```



[31]:
```
# Bin data into levels and plot binned data
levels = [0, 500000, 1000000, 6000000, 10000000, 10000000000]
cat_names = {
    0: "0-500'000$",
    1: "500'000-1'000'000$",
    2: "1'000'000-6'000'000$",
    3: "6'000'000-10'000'000$",
    4: "10'000'000-10'000'000'000$",
}
warn_params = Warn.WarnParameters(levels, operations=[])
warn_def = Warn.from_map(grid, coord_impf, warn_params)
warn_def.plot_warning(var_name='Impact Levels', title='Impact binned into Levels',
                      cat_name=cat_names, **plotting_parameters);
```



[32]:
```
# Apply fine tuned filtering operations with gradual decrease and changing too small␣
↪regions
warn_levels = [0, 500000, 1000000, 6000000, 10000000,   10000000000]
warn_params = Warn.WarnParameters(warn_levels, operations=[(Operation.erosion, 1),␣
↪(Operation.dilation,
                                                                                ␣
↪1),
                                                                (Operation.median_
↪filtering, 3)], gradual_decr=True,
                                      change_sm=50)
warn_def = Warn.from_map(grid, coord_impf, warn_params)
warn_def.plot_warning(var_name='Impact Levels', title='Impact clustered into Levels',
                      cat_name=cat_names, **plotting_parameters);
```


Impact clustered into Levels
21°N
20.5°N
20°N
19.5°N
19°N
18.5°N
18°N
17.5°N
75°W
74°W
73°W
72°W
71°W
10'000'000-10'000'000'000$
1'000'000-6'000'000$
500'000-1'000'000$
0-500'000$
Impact Levels

[19]:

CHAPTER

THIRTEEN

CLIMADA

CLIMADA stands for **CLIM**ate **ADA**ptation and is a probabilistic natural catastrophe impact model, that also calculates averted damage (benefit) thanks to adaptation measures of any kind (from grey to green infrastructure, behavioural, etc.).

As of today, CLIMADA provides global coverage of major climate-related extreme-weather hazards at high resolution via a data API, namely (i) tropical cyclones, (ii) river flood, (iii) agro drought and (iv) European winter storms, all at 4km spatial resolution - wildfire to be added soon. For all hazards, historic and probabilistic event sets exist, for some also under select climate forcing scenarios (RCPs) at distinct time horizons (e.g. 2040). See also papers for details.

CLIMADA is divided into two parts (two repositories):

1. the core climada_python contains all the modules necessary for the probabilistic impact, the averted damage, uncertainty and forecast calculations. Data for hazard, exposures and impact functions can be obtained from the data API. Litpop is included as demo Exposures module, and Tropical cyclones is included as a demo Hazard module.
2. the petals climada_petals contains all the modules for generating data (e.g., TC_Surge, WildFire, OpenStreeMap, …). Most development is done here. The petals builds-upon the core and does not work as a stand-alone.

It is recommend for new users to begin with the core (1) and the tutorials therein.

This is the Python (3.8+) version of CLIMADA - please see https://github.com/davidnbresch/climada for backward compatibility (MATLAB).

13.1 Getting started

CLIMADA runs on Windows, macOS and Linux. The released versions of CLIMADA are available from conda-forge. Use the Mamba package manager to install it:

```
mamba install -c conda-forge climada-petals
```

It is **highly recommended** to install CLIMADA into a **separate** Conda environment. See the installation guide for further information.

Follow the tutorial `climada_python-x.y.z/doc/tutorial/1_main_climada.ipynb` in a Jupyter Notebook to see what can be done with CLIMADA and how.

13.2 Documentation

Documentation is available on Read the Docs:

Note that all the documentations has two versions,'latest' and 'stable', and explicit version numbers, such as 'v3.1.1', in the url path. 'latest' is created from the 'develop' branch and has the latest changes by developers, 'stable' from the latest release. For more details about documentation versions, please have a look at here.

CLIMADA python:

- online (recommended)
- PDF file

CLIMADA petals:

- online (recommended)
- PDF file
- petals Tutorials on GitHub

The documentation can also be built locally.

13.3 Citing CLIMADA

If you use CLIMADA please cite (in general, in particular for academic work) :

The used version

and/or the following published articles:

Aznar-Siguan, G. and Bresch, D. N., 2019: CLIMADA v1: a global weather and climate risk assessment platform, Geosci. Model Dev., 12, 3085–3097, https://doi.org/10.5194/gmd-12-3085-2019

Bresch, D. N. and Aznar-Siguan, G., 2021: CLIMADA v1.4.1: towards a globally consistent adaptation options appraisal tool, Geosci. Model Dev., 14, 351-363, https://doi.org/10.5194/gmd-14-351-2021

Please see all CLIMADA-related scientific publications in our repository of scientific publications and cite according to your use of select features, be it hazard set(s), exposure(s) …

In presentations or other graphical material, as well as in reports etc., where applicable, please add the logo as follows:

As key link, please use https://wcr.ethz.ch/research/climada.html, as it will last and provides a bit of an intro, especially for those not familiar with GitHub - plus a nice CLIMADA infographic towards the bottom of the page

13.4 Contributing

See the Contribution Guide.

13.5 Versioning

We use SemVer for versioning. For the versions available, see the releases on this repository.

13.6 License

CHAPTER

FOURTEEN

CHANGELOG

14.1 4.1.0

Release date: 2024-02-19

14.1.1 Dependency Changes

Updated:

- `climada` >=4.0 → ==4.1

Added:

- `overpy` >=0.7
- `osm-flex` >=1.1.1
- `pymrio` >=0.5

14.1.2 Changed

- Restructured `Supplychain` module, which now uses `pymrio` to download and handle multi-regional input output tables #66
- Restructured `openstreetmap` module to draw functionalities from external package osm-flex #103
- As part of `climada_petals.hazard.tc_rainfield`, implement a new, physics-based TC rain model ("TCR") in addition to the existing implementation of the purely statistical R-CLIPER model (#85)
- Conda environment now avoids `default` channel packages, as these are incompatible to conda-forge #110

14.2 4.0.2

Release date: 2023-09-27

14.2.1 Dependency Changes

- `pandas` >=1.5,<2.0 → >=1.5 (compatibility with pandas 2.x)

14.2.2 Changed

- improved integration tests for notebooks and external data apis

14.2.3 Fixed

- implicit casting from `DataArray` to `int` in reading mehtods made explicit #95

14.3 4.0.1

Release date: 2023-09-06

14.3.1 Fixed

- `TCForecast` now skips "untrackable" TCs when reading multi-message `.bufr` files #91

14.4 4.0.0

Release date: 2023-09-01

14.4.1 Dependency Changes

Upgraded:

- shapely `1.8` -> `2.0` (#80)

14.4.2 Changed

- refactored `climada_petals.river_flood.RiverFlood.from_nc`, removing calls to `set_raster` (#80)
- Replace `tag` attribute with string `description` in classes derived from `Exposure` #89

14.4.3 Removed

- `tag` attribute from hazard classes #88

14.5 v3.3.2

Release date: 2023-08-25

14.5.1 Description

Patch release

14.6 v3.3.1

Release date: 2023-08-24

14.6.1 Description

Rearranged file-system structure: `data` subdirectory of `climada_petals`.

14.7 v3.3.0

Release date: 2023-05-08

14.7.1 Description

Release aligned with climada (core) 3.3.

14.7.2 Added

- Changelog and PR description template based on the Climada Core repository #72

14.7.3 Changed

- Rework docs and switch to Book theme *#63*

14.7.4 Fixed

- fix issue #69 Warn.zeropadding for islands

CHAPTER

FIFTEEN

CLIMADA PETALS LIST OF AUTHORS

- Gabriela Aznar-Siguan
- David N. Bresch
- Samuel Eberenz
- Jan Hartman
- Marine Perus
- Thomas Röösli
- Dario Stocker
- Veronica Bozzini
- Tobias Geiger
- Carmen B. Steinmann
- Evelyn Mühlhofer
- Rachel Bungerer
- Inga Sauer
- Samuel Lüthi
- Pui Man Kam
- Simona Meiler
- Alessio Ciullo
- Thomas Vogt
- Benoit P. Guillod
- Chahan Kropf
- Emanuel Schmid
- Chris Fairless
- Jan Wüthrich
- Zélie Standhanske
- Lukas Riedel

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

E

F

G

H

P

R

S

T

V

W

Y