
CLIMADA petals documentation

Release 6.2.0

CLIMADA contributors

Jun 15, 2026

CONTENTS

1	River Flood Hazards from GloFAS Discharge Data	3
1.1	Overview	3
1.2	Preparations	3
1.3	Computing a Flood Footprint	4
2	Software documentation per package	9
2.1	climada_petals.engine package	9
2.2	climada_petals.entity package	31
2.3	climada_petals.hazard package	47
2.4	climada_petals.util package	92
3	Hazard Tutorials	93
3.1	Hazard Emulator	93
3.2	Hazard: Landslides	104
3.3	Hazard: RiverFlood	118
3.4	ECMWF OPERATIONAL FORECAST TRACKS	140
3.5	Hazard: Tropical cyclone rain from R-CLIPER or TCR model	142
3.6	Hazard: Tropical cyclone surge from linear wind-surge relationship and a bathtub model	152
3.7	Surge from tropical cyclones modeled with GeoClaw	164
3.8	Hazard: WildFire	171
3.9	River Flood based on GloFAS Discharge Data	187
4	supplychain module	203
4.1	Introduction	203
4.2	Goal of this tutorial	203
4.3	What approaches are available?	203
4.4	Tutorial	204
5	CAT Bond Tutorial	221
5.1	Import packages	222
5.2	Set CAT Bond Basic Settings	222
5.3	Define Exposure, Hazard, and Vulnerability per Country	223
5.4	Initialise Subareas Class	225
5.5	Subarea Calculations	228
5.6	Single-Country CAT Bond Simulation	229
5.7	Multi-Country CAT Bond Simulation	232
5.8	Earthquake CAT Bond Simulation	238
6	BlackMarble class	243
6.1	Input data:	243
6.2	Import BlackMarble()	243

6.3	Possible settings	243
6.4	Modeling exposure of Sovereign States	244
6.5	Model specific territories of Sovereign States	254
6.6	Change exponents of the polynomial transformation used for nighlight intensity:	257
7	CLIMADA & OpenStreetMap	261
7.1	Introduction & Overview on OSM data	261
7.2	Downloading, clipping and parsing OSM data	262
7.3	Risk computation with OSM data as CLIMADA Exposures	271
8	Crop production risk based on ISIMIP and FAO data	277
8.1	Summary	277
8.2	Data sources	277
8.3	RelativeCropyield Hazard	278
8.4	CropProduction Exposure	282
8.5	Impact: Deviation in yearly crop production	288
9	Tutorial Warn module	291
10	MeteoSwiss Storm Example	293
10.1	Demonstrate Warning Generation	294
11	Hazard Example - TC Haiti	301
12	Impact Example - Value (USD) Haiti	307
13	CLIMADA	313
13.1	Getting started	313
13.2	Documentation	313
13.3	Citing CLIMADA	314
13.4	Contributing	314
13.5	Versioning	314
13.6	License	314
14	Changelog	315
14.1	6.2.0	315
14.2	6.1.0	315
14.3	6.0.1	316
14.4	6.0.0	316
14.5	5.0.0	316
14.6	4.1.0	317
14.7	4.0.2	318
14.8	4.0.1	318
14.9	4.0.0	318
14.10	v3.3.2	318
14.11	v3.3.1	319
14.12	v3.3.0	319
15	CLIMADA List of Authors	321
	Python Module Index	323



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.

Copyright Notice

Copyright (C) 2017 ETH Zurich, CLIMADA contributors listed in [AUTHORS.md](#).

CLIMADA is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

CLIMADA is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with CLIMADA. If not, see <https://www.gnu.org/licenses/>.

RIVER FLOOD HAZARDS FROM GLOFAS DISCHARGE DATA

This tutorial will guide you through the GloFAS River Flood module of CLIMADA Petals. Its purpose is to download river discharge data from the Global Flood Awareness System (GloFAS) and compute flood depths from it. The data is stored by the [Copernicus Data Store \(CDS\)](#) and will be automatically downloaded in the process.

1.1 Overview

Instead of employing a computationally expensive hydrological model to compute inundation depths, this module uses a simplified statistical approach to compute flooded areas. As an input, the approach uses river discharge data and river flood hazard maps. These hazard maps contain flood footprints for specific return periods. The idea is to compute equivalent return periods for the discharge data at every pixel and then use the flood hazard maps to compute a flood hazard. For computing these return periods, we require an extreme value distribution at every point. In practice, we fit such distributions from the historical discharge data.

Depending on your area and time series of interest the computational cost and the amount of data produced can be immense. For a larger country, however, a single flood inundation footprint can be computed within few minutes on a decently modern machine.

1.2 Preparations

We need to prepare three things: The flood hazard maps, the extreme value distributions, and access to the CDS API.

Copernicus Data Store API Access

1. Register at the [Copernicus Data Store \(CDS\)](#) and log in.
2. Check out the [CDS API HowTo](#). In the section “Install the CDS API key”, copy the content of the black box on the right.
3. Create a file called `.cdsapirc` in your home directory and paste the contents of the black box into it.

If you are unsure where to put the file and you are working on a Linux or macOS system, open a terminal and execute

```
cd $HOME
touch .cdsapirc
```

Now the file is created and can be opened with your favorite text editor.

1.2.1 Use Prepared Datasets

The Gumbel distribution fit parameter data has been uploaded to the [ETH Research Collection](#) for your convenience: [Gumbel distribution fit parameters for historical GloFAS river discharge data \(1979–2015\)](#)

This dataset and the global flood hazard maps will be automatically downloaded when executing

```
from climada_petals.hazard.rf_glofas import setup_all

setup_all()
```

Alternatively, you can download the data yourself or specify custom paths to datasets on your machine.

After this step, you should have the following files in your `<climada-dir>/data/river-flood-computation`:

- `gumbel-fit.nc`: A NetCDF file containing `loc`, `scale` and `samples` variables with dimensions `latitude` and `longitude` on a grid matching the input discharge data (here: GloFAS).
- `flood-maps.nc`: A NetCDF file giving the `flood_depth` with dimensions `latitude`, `longitude`, and `return_period`. The grid is allowed to differ from that of the discharge and the Gumbel fit parameters and is expected to have a higher resolution.
- `FLOPROS_shp_V1/FLOPROS_shp_V1.shp`: A shapefile containing flood protection standards for the entire world, encoded as return period against which the local measures are protecting against.

1.3 Computing a Flood Footprint

With the required data in place and access to the Copernicus Data Store, we can proceed to compute a flood footprint. In the end, we want to arrive at a `Hazard` object we can use for computations in CLIMADA.

The overall procedure is as follows:

1. Instantiate an object of `RiverFloodInundation`.
2. Use it to download discharge data (either ensemble forecasts or historical reanalysis) from the CDS.
3. Compute flood inundation footprints from the downloaded data with `compute()`.
4. Create a series of hazard objects (or a single object) from the data using `hazard_series_from_dataset()`.

```
from climada_petals.hazard.rf_glofas import (
    RiverFloodInundation,
    hazard_series_from_dataset,
)

forecast_date = "2023-08-01"
rf = RiverFloodInundation()
rf.download_forecast(
    countries="Switzerland",
    forecast_date=forecast_date,
    lead_time_days=5,
    preprocess=lambda x: x.max(dim="step"),
)
ds_flood = rf.compute()
hazard = hazard_series_from_dataset(ds_flood, "flood_depth", "number")
```

1.3.1 Step-By-Step Instructions

The `compute()` method is a shortcut for the steps of the flood model algorithm that compute flood depth from the discharge input.

The single steps are as follows:

1. Computing the return period from the input discharge with `return_period()`. To that end, the fitted Gumbel distributions are used and a return period is computed by $r(q) = (1 - \text{cdf}(q))^{-1}$, where `cdf` is the cumulative distribution function of the fitted Gumbel distribution and `q` is the input discharge.

```
discharge = rf.download_forecast(
    countries="Switzerland",
    forecast_date=forecast_date,
    lead_time_days=5,
    preprocess=lambda x: x.max(dim="step"),
)
return_period = rf.return_period(discharge)
```

Alternatively, bootstrap sampling can be employed to represent the statistical uncertainty in the return period computation with `return_period_resample()`. In bootstrap sampling, we draw random samples from the fitted Gumbel distribution and fit a new distribution from them. This process can be repeated an arbitrary number of times. The resulting distribution quantifies the uncertainty in the original fit. The first argument to the method is the number of samples to draw while bootstrapping (i.e., how many samples the resulting distribution should have).

```
return_period = rf.return_period_resample(10, discharge)
```

2. Regridding the return period onto the higher resolution grid of the flood hazard maps with `regrid()`:

```
return_period_regrid = rf.regrid(return_period)
```

3. *Optional:* Applying the protection level with `apply_protection()`:

```
return_period_regrid_protect = rf.apply_protection(return_period_regrid)
```

4. Computing the flood depth from the regridded return period by interpolating between flood hazard maps for various return periods with `flood_depth()`

```
flood_depth = rf.flood_depth(return_period_regrid)
flood_depth_protect = rf.flood_depth(return_period_regrid_protect)
```

If `compute()` was executed with `apply_protection="both"` (default), it will merge the data arrays for flood depth without protection applied and with protection applied, respectively, into a single dataset and return it.

1.3.2 Passing Keyword-Arguments to `compute`

If you want to pass custom arguments to the methods called by `compute()` without calling each method individually, you can do so via the `resample_kws` and `regrid_kws` arguments.

If you add `resample_kws`, `compute()` will call `return_period_resample()` instead of `return_period()` and pass the mapping as keyword arguments.

Likewise, `regrid_kws` will be passed as keyword arguments to `regrid()`.

```
ds_flood = rf.compute(
    resample_kws=dict(num_bootstrap_samples=20, num_workers=4),
```

(continues on next page)

```
regrid_kws=dict(reuse_regridder=True)
)
```

1.3.3 Creating Hazards from the Data

With the computation successful, we now want to create `Hazard` objects. The resulting data is usually multi-dimensional, which is why we typically create multiple `Hazard` objects from it. Two obvious dimensions are the spatial ones, longitude and latitude. Ignoring these (as they must persist into the `Hazard` object), we can decide on one more dimension to merge into a single hazard.

If we use an ensemble forecast like in the above example, and decide *not* to compute the maximum in time, the dataset has four coordinates: `latitude`, `longitude`, `step`, and `number`, with the latter two indicating the lead time step and the ensemble member, respectively. Employing bootstrap sampling would add another dimension `sample`. To create hazard objects, we would have to decide which of these dimensions should encode the “event” dimension in the `Hazard`. For each combination of the remaining dimension coordinates, a new `Hazard` object would then be created.

The task of splitting and concatenating along particular dimensions of the dataset and creating `Hazard` objects is performed by `climada_petals.hazard.rf_glofas.rf_glofas.hazard_series_from_dataset()`. We put in the data as file path or `xarray Dataset` and receive a `pandas.Series` with the hazard objects as values and the remaining dimension coordinates as `MultiIndex`. The dimension name which is to be considered the event dimension in a `Hazard` instance must be specified as the `event_dim` argument.

Tip

If the dataset is three-dimensional, `climada_petals.hazard.rf_glofas.rf_glofas.hazard_series_from_dataset()` will return a single `Hazard` object instead of a `pandas.Series`.

```
discharge = rf.download_forecast(
    countries="Switzerland",
    forecast_date="2023-08-01",
    lead_time_days=5,
)

# Compute flood for maximum over lead time
ds_flood = rf.compute(discharge.max(dim="step"))

# Single hazard return (no remaining dimensions)
hazard = hazard_series_from_dataset(ds_flood, "flood_depth", "number")

# Compute flood for each lead time day *and* bootstrap sample
ds_flood_multidim = rf.compute(discharge, resample_kws=dict(num_bootstrap_samples=20))

# Series with MultiIndex: step, member
# Each hazard with 20 events (samples)
hazard_series = hazard_series_from_dataset(ds_flood, "flood_depth", "sample")
```

1.3.4 Storing Data

Use `climada_petals.hazard.rf_glofas.transform_ops.save_file()` to store `xarray Datasets` or `DataArrays` conveniently.

 Tip

Storing your result is important for two reasons:

1. Computing flood footprints for larger areas or multiple events can take a lot of time.
2. Loading flood footprints into `Hazard` objects requires transpositions that do not commute well with the lazy computations and evaluations by `xarray`. Storing the data and re-loading it before plugging it into `hazard_series_from_dataset()` will likely increase performance.

By default, data is stored without compression and encoded in 32-bit floats. This maintains a reasonable accuracy while reducing file size by half even though no compression is applied. Compression will drastically reduce the storage space needed for the data. However, it also creates a heavy burden on the CPU and especially multiprocessing and multithreading tasks suffer heavily. If storage space permits, it is therefore recommended to store the data without compression.

 Warning

Saving results of computations **with** compression is **not** recommended, because performance might be impeded a **lot!**

To enable compression, add `zlib=True` as argument to `save_file()`. The default compression level is `complevel=4`. The compression level may range from 1 to 9.

Because storing without compression does not compromise multiprocessing performance, it might be feasible to first write *without* compression after computing the result, and then to re-write *with* compression separately to save storage space. The reason for this is that `xarray` uses `dask` to perform computations lazily. Only when data is required, `dask` will compute it according to the transformations applied on the data. This does not commute well with compression.

The following code will likely run much faster than directly writing `ds_flood` with compression, especially when the data is large. However, it requires the space to once store the entire dataset without compression.

```
from pathlib import Path
import xarray as xr
from climada_petals.hazard.rf_glofas import save_file

rf.download_forecast(
    countries="Switzerland",
    forecast_date="2023-08-01",
    lead_time_days=5,
)
ds_flood = rf.compute()

# Save without compression (default)
outpath = Path("out.nc")
save_file(ds_flood, outpath)
ds_flood.close() # Release data

# Re-open, and save with compression into "out-comp.nc"
with xr.open_dataset(outpath, chunks="auto") as ds:
    save_file(ds, outpath.with_stem(outpath.stem + "-comp"), zlib=True)

# Delete initial file
outpath.unlink()
```

1.3.5 Storing Intermediate Data

By default, *RiverFloodInundation* stores the result of each computation step in a cache directory and reloads it for the next step. The reason for this is similar to the issue with compression: To perform our computations, the data has to be transposed often. Multiple transpositions of a dataset in memory are costly, but storing data and reopening it transposed is fast. Especially for larger data that do not fit into memory at once, *store_intermediates* should therefore be set to `True` (default).

The intermediate data is stored in a cache directory which is deleted after the *RiverFloodInundation* instance is closed or deleted. While it exists, the cached data can be accessed via the *cache_paths* after the computation:

```
import xarray as xr

rf.download_forecast(
    countries="Switzerland",
    forecast_date="2023-08-01",
    lead_time_days=5,
)
ds_flood = rf.compute()

# Plot regridded return period
with xr.open_dataarray(rf.cache_paths.return_period_regrid, chunks="auto") as da_rp:
    da_rp.isel(step=0).max(dim="member").plot()
```

SOFTWARE DOCUMENTATION PER PACKAGE

2.1 climada_petals.engine package

2.1.1 climada_petals.engine.cat_bonds package

climada_petals.engine.cat_bonds.helpers_tutorial_cat_bonds module

`climada_petals.engine.cat_bonds.helpers_tutorial_cat_bonds.create_earthquake_mmi_impact_function` (*impf*, *haz_*

Create a stylized earthquake impact function for MMI-based hazards.

The curve uses separate smooth transitions for the affected asset share (PAA) and the mean damage degree conditional on being affected (MDD).

Parameters

- **impf_id** (*int, optional*) – Impact function identifier.
- **haz_type** (*str, optional*) – Hazard type associated with the impact function.

Returns

Impact function set containing a single MMI-based impact function.

Return type

ImpactFuncSet

`climada_petals.engine.cat_bonds.helpers_tutorial_cat_bonds.remap_event_years_even` (*hazard*, *start_year=1980*, *n_years=1000*)

This function is intended for tropical cyclone data sets based on STORM which can be downloaded from the CLIMADA data API. In these data sets, all events are assigned the same year which is not suitable for CAT bond simulations that rely on the distribution of events over time. This function remaps the event years to be evenly distributed over a specified number of years while preserving the original month/day seasonality of the events.

climada_petals.engine.cat_bonds.mlt_bond_simulation module

climada_petals.engine.cat_bonds.pooling_functions module

climada_petals.engine.cat_bonds.premium_class module

class `climada_petals.engine.cat_bonds.premium_class.PremiumCalculations` (*bond_simulation_class*)

Bases: `object`

Premium calculation utilities for catastrophe bond simulations.

This class provides premium estimation methods based on Chatoro regression, IBRD bond curve fitting, and Sharpe ratio benchmarks.

Parameters

bond_simulation_class (*object*) – Bond simulation instance exposing *loss_metrics*, *df_loss_month*, and *term* attributes.

__init__ (*bond_simulation_class*)

Initialize with a bond simulation object.

Parameters

bond_simulation_class (*object*) – Bond simulation instance exposing *loss_metrics*, *df_loss_month*, and *term* attributes.

calc_chatoro_premium (*peak_multi: int, investment_graded: int, hybrid_trigger: int, GCIndex=None, BBSpread=None*)

Calculate premium using the Chatoro et al. (2022) regression model.

Parameters

- **peak_multi** (*float*) – Indicator whether the cat bond is considered peak peril or if it is covering multiple perils. If either is true = 1, else 0.
- **investment_graded** (*int*) – Indicator for investment-grade status (e.g., 1 for yes, 0 for no).
- **hybrid_trigger** (*int*) – Indicator for hybrid trigger structure (e.g., 1 for yes, 0 for no).
- **GCIndex** (*float, optional*) – Guy Carpenter Global Property Catastrophe Rate on Line Index value.
- **BBSpread** (*float, optional*) – ICE BofA BB US High Yield Index OAS value.
- **Sets**
- **_____**
- **chatoro_prem_rate** (*float*) – Calculated Chatoro premium rate.

monoExp (*x, a, k, b*)

Exponential function for risk multiple curve fitting.

Parameters

- **x** (*float or np.ndarray*) – Input value(s) for the expected loss.
- **a** (*float*) – Amplitude parameter.
- **k** (*float*) – Decay rate parameter.
- **b** (*float*) – Offset parameter.

Returns

Evaluated exponential function values.

Return type

float or np.ndarray

calc_ibrd_premium (*peril=None, year=None*)

Fits a monotonic exponential curve to catastrophe bond data for bonds issued by the World Bank to estimate premium parameters. This function loads IBRD bond data from a csv file, optionally filters the data by peril type or issuing year, and fits a monotonic exponential function to the relationship between expected loss and risk multiple. The fitted parameters are returned.

Parameters

- **peril** (*str, optional*) – Peril type to filter the bonds. If None, no filtering by peril is applied.

- **year** (*list[int] or int, optional*) – Issuing year(s) to filter the bonds. If None, no filtering by year is applied.
- **Sets**
- _____
- **ibrd_prem_rate** (*float*) – The estimated premium rate based on the fitted curve and the bond's expected annual loss.

find_sharpe (*premium: float, monthly_losses: DataFrame, target_sharpe: float*)

Calculates the squared difference between the Sharpe ratio of a cat bond cash flow and a target Sharpe ratio. The function simulates the annual cash flows of a catastrophe bond investment, adjusting for losses and premium payments. It computes the average return and standard deviation of the net cash flows, then calculates the Sharpe ratio and returns the squared difference from the target Sharpe ratio.

Parameters

- **premium** (*float*) – Annual premium rate paid to the investor.
- **monthly_losses** (*pandas.DataFrame*) – DataFrame with columns 'losses' (list of loss amounts per event) and 'months' (list of event months) for each year.
- **target_sharpe** (*float*) – Target Sharpe ratio to compare against.

Returns

Squared difference between the calculated Sharpe ratio and the target Sharpe ratio.

Return type

float

calc_benchmark_premium (*target_sharpe: float*)

Calculates the initial premium required to achieve a target Sharpe ratio for a given set of annual losses. This function uses numerical optimization to find the premium value that results in the desired Sharpe ratio.

Parameters

target_sharpe (*float*) – Desired Sharpe ratio to be achieved.

Returns

Optimal premium value that achieves the target Sharpe ratio.

Return type

float

climada_petals.engine.cat_bonds.sng_bond_simulation module

```
class climada_petals.engine.cat_bonds.sng_bond_simulation.SingleCountryBondSimulation (subarea_calc,
                                                                                       term:
                                                                                       int,
                                                                                       start_year:
                                                                                       int,
                                                                                       num-
                                                                                       ber_of_terms:
                                                                                       int)
```

Bases: object

Single-country catastrophe bond simulation utilities.

This class simulates losses and returns for a single country's bond based on per-event payouts and damages.

`__init__` (*subarea_calc*, *term*: int, *start_year*: int, *number_of_terms*: int)

Initialize with subarea data and simulation horizon.

Parameters

- **subarea_calc** (*object*) – Subarea calculation instance.
- **term** (*int*) – Bond term in years for each simulated period.
- **start_year** (*int*) – The starting year of the simulation.
- **number_of_terms** (*int*) – Number of consecutive terms to simulate.

`init_bond_loss` (*events_per_year*: list[DataFrame])

Calculates the expected losses for a catastrophe bond over its term. This function simulates the bond’s loss experience given a sequence of event data per year and month, tracking payouts, damages, remaining principal value, and the timing of losses. It returns the relative losses, the total payouts and damages per term, and a DataFrame detailing losses and their corresponding months.

Parameters

events_per_year (*list[pandas.DataFrame]*) – A list where each element is a DataFrame representing events in a year. Each DataFrame must contain at least ‘month’ and ‘pay’ columns, where ‘pay’ is the payout for each event.

Returns

- **rel_annual_losses** (*numpy.ndarray*) – Array of relative payouts/losses per year (losses divided by principal).
- **rel_monthly_loss** (*pandas.DataFrame*) – DataFrame with columns ‘losses’ and ‘months’, detailing the losses and their corresponding months for each year.
- **summed_payouts** (*float*) – The total summed payouts over the bond’s term.
- **summed_damages** (*float*) – The total summed damages over the bond’s term.

`init_loss_simulation` (*confidence_levels*: list[float] = [0.95, 0.99])

Simulate losses, payouts, damages, and risk metrics for a catastrophe bond.

Parameters

- **confidence_levels** (*list[float]*, *optional*) – Confidence levels for VaR and ES calculation.
- **Sets**
- **_____**
- **df_loss_month** (*pd.DataFrame*) – Monthly loss data for all simulations.
- **loss_metrics** (*dict*) – Expected loss, attachment probability, total payouts/damages, VaR and ES metrics for given confidence levels.

`init_return_simulation` (*premium*: float)

Simulates the performance of a catastrophe bond over the simulation period, premiums and returns. This function models the bond’s payouts, premiums, and returns over a series of simulated years. It aggregates annual and total returns and computes Sharpe ratios.

Parameters

premium (*float*) – Annual premium rate for the bond.

Returns

return_metrics – Dictionary with annual premiums, annual returns, total returns, total premiums, and Sharpe ratio.

Return type

dict

climada_petals.engine.cat_bonds.subareas module

class `climada_petals.engine.cat_bonds.subareas.Subareas` (*hazard, vulnerability, exposure, subareas_gdf: GeoDataFrame*)

Bases: object

Handle subareas for CAT bond spatial aggregation.

Parameters

- **hazard** (*climada.Hazard*) – Hazard object containing event data. Attention: The date attribute is used in the CAT bond simulation. Date must include a year and month.
- **vulnerability** (*climada.Vulnerability*) – Vulnerability object containing vulnerability functions.
- **exposure** (*climada.Exposure*) – Exposure object containing monetary data and geometry.
- **subareas_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing subarea polygons. Must cover the exposure perimeter and include (or will be assigned) a *subarea_letter* column.

hazard

Hazard object containing event data. Attention: The date attribute is used in the CAT bond simulation. Date must include a year and month.

Type

climada.Hazard

vulnerability

Vulnerability object containing vulnerability functions.

Type

climada.Vulnerability

exposure

Exposure object containing monetary data and geometry.

Type

climada.Exposure

subareas_gdf

GeoDataFrame of subarea polygons with *subarea_letter* labels.

Type

geopandas.GeoDataFrame

__init__ (*hazard, vulnerability, exposure, subareas_gdf: GeoDataFrame*)

Initialize a Subareas instance.

Parameters

- **hazard** (*climada.Hazard*) – Hazard object containing event data. Attention: The date attribute is used
- **in the CAT bond simulation. Date must include a year and month.**
- **vulnerability** (*climada.Vulnerability*) – Vulnerability object containing vulnerability functions.
- **exposure** (*climada.Exposure*) – Exposure object containing monetary data and geometry.

- **subareas_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing subarea polygons.

classmethod from_resolution (*hazard, vulnerability, exposure, resolution: float, subareas_gdf: GeoDataFrame | None = None*)

Create subareas from a grid resolution and an exposure layer.

Parameters

- **hazard** (*climada.Hazard*) – Hazard object containing event data. Attention: The date attribute is used
- **in the CAT bond simulation. Date must include a year and month.**
- **vulnerability** (*climada.Vulnerability*) – Vulnerability object containing vulnerability functions.
- **exposure** (*climada.Exposure*) – Exposure object containing monetary data and geometry.
- **resolution** (*float*) – Grid cell size used to generate subareas.
- **subareas_gdf** (*geopandas.GeoDataFrame, optional*) – Unused placeholder for compatibility.

Returns

Initialized instance with generated subareas.

Return type

Subareas

classmethod from_geodataframe (*hazard, vulnerability, exposure, gdf: GeoDataFrame*)

Create subareas from an existing GeoDataFrame.

Parameters

- **hazard** (*climada.Hazard*) – Hazard object containing event data. Attention: The date attribute is used
- **in the CAT bond simulation. Date must include a year and month.**
- **vulnerability** (*climada.Vulnerability*) – Vulnerability object containing vulnerability functions.
- **exposure** (*climada.Exposure*) – Exposure object containing monetary data and geometry.
- **gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame of polygon subareas covering the exposure perimeter.

Returns

Initialized instance with the provided subareas.

Return type

Subareas

property exposure

Return the exposure object.

Returns

Exposure object containing monetary data and geometry.

Return type

climada.Exposure

plot()

Plot exposure raster with subarea boundaries overlaid.

Return type

None

count_subareas()

Return the number of subareas.

Returns

Number of subarea polygons.

Return type

int

climada_petals.engine.cat_bonds.subarea_calculations module

```
class climada_petals.engine.cat_bonds.subarea_calculations.SubareaCalculations (subareas,
                                                                    index_stat:
                                                                    float | str,
                                                                    ini-
                                                                    tial_guess:
                                                                    tuple[float,
                                                                    float] |
                                                                    None =
                                                                    None)
```

Bases: object

Subarea-level calculations for parametric catastrophe bond payout setup.

This class computes impacts, parametric indices, payout thresholds, and pay-versus-damage tables for a set of subareas.

Parameters

- **subareas** (*object*) – Container with *exposure*, *hazard*, *vulnerability*, and *subareas_gdf*.
- **index_stat** (*str or float*) – Statistic for the parametric index (e.g., “mean” or a percentile).
- **initial_guess** (*tuple[float, float], optional*) – Initial guess for minimum and maximum payout trigger thresholds.

subareas

Input container holding exposure, hazard, vulnerability, and subareas.

Type

object

index_stat

Statistic used to summarize hazard intensity per subarea which is used as parametric index.

Type

str or float

initial_guess

Initial guess for minimum/maximum thresholds for the payout function.

Type

tuple[float, float]

`__init__` (*subareas*, *index_stat*: float | str, *initial_guess*: tuple[float, float] | None = None)

Initialize the subarea calculation helper.

Parameters

- **subareas** (*object*) – Container with *exposure*, *hazard*, *vulnerability*, and *subareas_gdf*.
- **index_stat** (*str or float*) – Statistic used for the parametric index (“mean” or a percentile).
- **initial_guess** (*tuple[float, float]*, *optional*) – Initial guess for min/max payout trigger thresholds. When omitted, the 30th and 60th percentiles of hazard intensity are used.

`create_pay_vs_dam` (*attachment_point*: float, *exhaustion_point*: float, *methods_attachment_point*: str | None = None, *methods_exhaustion_point*: str | None = None)

Build the pay-versus-damage table for the bond structure.

This runs impact calculation, parametric index creation, payout calibration, and computes pay/damage per event.

Parameters

- **attachment_point** (*float*) – Attachment point value or parameter for the selected method. Is either a monetary amount, a fraction of total exposure, or a return period in years.
- **exhaustion_point** (*float*) – Exhaustion point value or parameter for the selected method. Is either a monetary amount, a fraction of total exposure, or a return period in years.
- **methods_attachment_point** (*str, optional*) – Interpretation method for attachment (e.g., “Exposure_Share” or “Return_Period”).
- **methods_exhaustion_point** (*str, optional*) – Interpretation method for exhaustion (e.g., “Exposure_Share” or “Return_Period”).

Returns

Sets *principal*, *attachment*, *results*, *opt_min_thresh*, *opt_max_thresh*, and *pay_vs_dam* on the instance.

Return type

None

`climada_petals.engine.cat_bonds.subarea_calculations.calc_payout` (*min_trig*: float, *max_trig*: float, *haz_int*: DataFrame, *max_pay*: float)

Calculates payout values based on a linear payout function using hazard intensities and trigger thresholds.

Parameters

- **min_trig** (*float*) – The minimum trigger threshold for hazard intensity.
- **max_trig** (*float*) – The maximum trigger threshold for hazard intensity.
- **haz_int** (*pandas.DataFrame*) – DataFrame containing hazard intensity values in the first column.
- **max_pay** (*float*) – The maximum payout value.

Returns

payouts – Array of calculated payout values corresponding to each hazard intensity.

Return type

numpy.ndarray

climada_petals.engine.cat_bonds.utils_cat_bonds module

`climada_petals.engine.cat_bonds.utils_cat_bonds.multi_level_es` (*losses: Series, confidence_levels: list[float]*)

Calculate Value at Risk (VaR) and Expected Shortfall (ES) for multiple confidence levels.

Parameters

- **losses** (*pandas.Series*) – Loss samples used to estimate tail risk metrics.
- **confidence_levels** (*list[float]*) – Confidence levels (e.g., [0.95, 0.99]).

Returns

- **var_list** (*list[float]*) – VaR values in the order of the given confidence levels.
- **es_list** (*list[float]*) – ES values in the order of the given confidence levels.

`climada_petals.engine.cat_bonds.utils_cat_bonds.allocate_single_payout` (*payout: float, nominals: ndarray*)

Allocate a single payout across tranche nominals.

Parameters

- **payout** (*float*) – Payout amount to allocate.
- **nominals** (*array-like*) – 1D array of tranche nominal values.

Returns

- **remaining_nominals** (*numpy.ndarray*) – Remaining nominal values after the payout.
- **payout_per_tranche** (*numpy.ndarray*) – Amount allocated to each tranche.

2.1.2 climada_petals.engine.supplychain module

`class climada_petals.engine.supplychain.DirectShocksSet` (*mriot_name: str, mriot_sectors: Index, mriot_regions: Index, exposure_assets: Series, impacted_assets: DataFrame, event_dates: ndarray, monetary_factor: int, shock_name: str = 'unnamed'*)

Bases: `object`

`DirectShocksSet` class

The `DirectShocksSet` class provides methods for ‘translating’ Impact and Exposure objects into economic shocks, corresponding to a specific MRIOT table, from which indirect economic costs can be computed.

mriot_name

The name of the MRIOT defining the typology of assets.

Type

`str`

mriot_sectors

The list of possible sectors.

Type

`pd.Index`

mriot_regions

The list of possible regions.

Type

pd.Index

name

A name to identify the object. For convenience only.

Type

str, default “unnamed”

monetary_factor

The monetary factor of the corresponding MRIOT.

Type

int

exposure_assets

Exposure translated in the region/sector typology of the MRIOT. The index of the Series are the possible (region, sector).

Type

pd.Series

impacted_assets

Impact translated in the region/sector typology of the MRIOT. The columns index are the possible (region, sector), and the row index correspond to event_ids of the Impact.

Type

pd.DataFrame

event_dates

Series of the dates of the events, as ordinals, with event ids as index.

Type

pd.Series

__init__ (*mriot_name: str, mriot_sectors: Index, mriot_regions: Index, exposure_assets: Series, impacted_assets: DataFrame, event_dates: ndarray, monetary_factor: int, shock_name: str = 'unnamed'*) → None

classmethod combine (*direct_shocks: list[DirectShocksSet], kind: Literal['merge', 'concat'] = 'merge', combine_name: str = 'unnamed_combine'*)

Combine multiple *DirectShocksSet* instances into a single instance.

This method supports two modes of combination: merging and concatenating. - In “merge” mode, the direct shocks are summed together for each matching event

and per region/sector.

- In “concat” mode, the direct shocks are assumed to originate from independent events and are concatenated along the event axis.

Parameters

- **direct_shocks** (*list of DirectShocksSet*) – A list of *DirectShocksSet* instances to be combined.
- **kind** (*{“merge”, “concat”}, optional (Default: “merge”)*) – The type of combination to apply. - “merge”: Sum the shocks across all events. - “concat”: Concatenate shocks assuming independent events.

- **combine_name** (*str, optional*) – Name for the combined *DirectShocksSet*. Default is “unnamed_combine”.

Returns

A new *DirectShocksSet* instance representing the combination of the input shocks.

Return type

DirectShocksSet

Raises

ValueError – If the *DirectShocksSet* instances do not share the same *mriot_name*.

Notes

- All *DirectShocksSet* instances must share the same *mriot_name*.
- Merging sums the shocks and combines the corresponding impacted and exposed assets.
- Concatenating assumes independent events and stacks the shocks along the event axis.

Examples

Assuming: - *mriot* is an *IOSystem* object representing a MRIOT with (at least) Agriculture and Manufacture sectors. - *exp_agri* contains exposure data for the Agriculture sector - *exp_manu* contains exposure data for the for the Manufacture sector. - *imp_TC_agri* is an *Impact* object with impacts on the agriculture sector from different tropical cyclones - *imp_TC_manu* is an *Impact* object with impacts on the manufacture sector from different tropical cyclones - *imp_FL_agri* is an *Impact* object with impacts on the manufacture sector from different floods on the agriculture sector - *imp_TC_surges_manu* is an *Impact* object with impacts from surges that occurred during the same tropical cyclones as defined in *imp_TC_manu*

```
>>> shocks_TC_agri = DirectShocksSet.from_assets_and_imp(mriot=mriot,
↳ exposure=exp_agri, impact=imp_TC_agri, affected_sectors=["Agriculture"])
>>> shocks_TC_manu = DirectShocksSet.from_assets_and_imp(mriot=mriot,
↳ exposure=exp_manu, impact=imp_TC_manu, affected_sectors=["Manufacture"])
>>> shocks_FL_agri = DirectShocksSet.from_assets_and_imp(mriot=mriot,
↳ exposure=exp_agri, impact=imp_FL_agri, affected_sectors=["Agriculture"])
>>> shocks_TC_surges_manu = DirectShocksSet.from_assets_and_imp(mriot=mriot,
↳ exposure=exp_manu, impact=imp_TC_surges_agri, affected_sectors=["Manufacture
↳ "])
```

Assume you want to compute the impacts from TCs on both Agriculture and Manufacture sectors, you can “merge” *shocks_TC_agri* and *shocks_TC_manu* for that purpose:

```
>>> combined_set = DirectShocksSet.combine([shock_TC_agri, shock_TC_manu],
↳ kind="merge")
```

If you want to evaluate the perturbations from TCs and floods on the Agriculture sectors, you can “concat” *shocks_TC_agri* and *shocks_FL_agri* for that purpose:

```
>>> combined_set = DirectShocksSet.combine([shocks_TC_agri, shocks_FL_agri],
↳ kind="concat", combine_name="impact from floods and TCs")
```

Finally if you want to look at the shocks of “co-occurrence” of surges and TCs for which the impacts were computed independently, you can merge *shocks_TC_manu* and *shocks_TC_surge_manu*.

Of course you can use more that one sectors at once per individual *DirectShocksSet*. The purpose of this *combine* method is to allow flexibility with employing different exposure layers and impact computation.

property event_ids_with_impact: Index

Get the IDs of events that have a non-zero impact on assets.

This property filters the *impacted_assets* DataFrame to retrieve the indices of rows where at least one asset has a non-zero impact.

Returns

Index of events with at least one impacted asset.

Return type

pandas.Index

property exposure_assets_not_null: Series

Get a subset of the *exposure_assets* DataFrame containing only non-null exposure.

This property filters the *exposure_assets* DataFrame to include only rows with non-zero values.

Returns

A Series containing the impacted assets with non-zero values.

Return type

pandas.Series

classmethod from_assets_and_imp (*mriot: IOSystem, exposure_assets: Series, impact: Impact, shock_name: str | None, affected_sectors: Iterable[str] | dict[str, float] | Series | Literal['all'], impact_distribution: dict[str, float] | Series | None, custom_mriot: bool = False*)

Build a DirectShocksSet from an MRIOT, assets Series, and Impact objects.

This method translates the given Impact object to the MRIOT typology (see *from_exp_and_imp()*).

Parameters

- **mriot** (*pymrio.IOSystem*) – The MRIOT to use for the typology of region and sectors.
- **exposure_assets** (*pd.Series*) – A pandas *Series* with (region,sector) as index and assets value.
- **impact** (*Impact*) – The Impact object to derive the impact on assets from.
- **shock_name** (*str or None*) – An optional name to identify the object.
- **affected_sectors** (*Iterable[str] | dict[str, float] | pd.Series | Literal["all"]*) – The sectors of the MRIOT that are impacted. If given as a collection of string, or “all”, then the total assets of the region are distributed proportionally to each sectors gross output. A dictionary *sector:share* can also be passed to specify which share of the total regional assets should be distributed to each sector.
- **impact_distribution** (*dict[str, float] | pd.Series, optional*) – This argument specify how the impact per region should be distributed to the impacted sectors. Using *None* will distribute proportionally to each sectors gross output in the MRIOT. A dictionary in the form *sector:share* or similarly a *Series* can be used to specify a custom distribution.
- **custom_mriot** (*bool*) – Whether to consider the MRIOT as a custom one (skips name checking), defaults to False. Note that its regions has to be ISO3 countries name for it to work.

classmethod from_exp_and_imp (*mriot: IOSystem, exposure: Exposures, impact: Impact, affected_sectors: Iterable[str] | dict[str, float] | Series | Literal['all'], impact_distribution: dict[str, float] | Series | None, shock_name: str | None = None, exp_value_col: str = 'value', custom_mriot: bool = False*)

Build a DirectShocksSet from MRIOT, Exposure and Impact objects.

This method translates both the given Exposure and Impact objects to the MRIOT typology.pd.Series First, it aggregates exposure values by *region_id* and then maps them to the regions of the given mriot. Assets are then distributed to the different sectors specified by *affected_sectors*. It proceeds with a similar process for the Impact object, using *impact_distribution* to distribute the impact across sectors.

Parameters

- **mriot** (*pymrio.IOSystem*) – The MRIOT to use for the typology of region and sectors.
- **exposure** (*Exposures*) – The Exposures object to use to derive total assets from.
- **impact** (*Impact*) – The Impact object to derive the impact on assets from.
- **shock_name** (*str* | *None*) – An optional name to identify the object, defaults to “unnamed”.
- **affected_sectors** (*Iterable[str]* | *dict[str, float]* | *pd.Series* | *Literal[“all”]*) – The sectors of the MRIOT that are impacted. If given as a collection of string, or “all”, then the total assets of the region are distributed proportionally to each sectors gross output. A dictionary *sector:share* can also be passed to specify which share of the total regional assets should be distributed to each sector.
- **impact_distribution** (*dict[str, float]* | *pd.Series, optional*) – This argument specify how the impact per region should be distributed to the impacted sectors. Using *None* will distribute proportionally to each sectors gross output in the MRIOT. A dictionary in the form *sector:share* or similarly a *Series* can be used to specify a custom distribution.
- **exp_value_col** (*str*) – The name of the column of the Exposure data representing the value of assets in each centroids.
- **custom_mriot** (*bool*) – Whether to consider the MRIOT as a custom one (skips name checking), defaults to False. Note that its regions has to be ISO3 countries name for it to work.

property impacted_assets_not_null: DataFrame

Get a subset of the *impacted_assets* DataFrame containing only non-null impacts.

This property filters the *impacted_assets* DataFrame to include only rows and columns where at least one non-zero impact exists.

Returns

A DataFrame containing the impacted assets with non-zero values.

Return type

pandas.DataFrame

property relative_impact: DataFrame

The ratio of impacted assets over total assets (0. if total assets are 0.).

property relative_impact_not_null: DataFrame

Get a subset of the *relative_impact* DataFrame containing only non-null values.

This property filters the *relative_impact* DataFrame to include only rows and columns where at least one non-zero impact exists.

Returns

A DataFrame containing the impacted assets with non-zero values.

Return type

pandas.DataFrame

class climada_petals.engine.supplychain.IndirectCostModel (*mriot: IOSystem, direct_shocks: DirectShocksSet* | *None = None*)

Bases: ABC

Abstract base class for modeling indirect costs using a Multi-Regional Input-Output Table (MRIOT).

This class provides a structure for incorporating direct shocks into an indirect cost model.

Parameters

- **mriot** (*pymrio.IOSystem*) – The Multi-Regional Input-Output System used in the model.
- **direct_shocks** (*DirectShocksSet or None, optional*) – The direct shocks to apply to the model. If provided, the model will initialize with these shocks. Default is None.

mriot

The Multi-Regional Input-Output System used in the model.

Type

pymrio.IOSystem

direct_shocks

The direct shocks applied to the model. None if no shocks have been applied.

Type

DirectShocksSet or None

__init__ (*mriot: IOSystem, direct_shocks: DirectShocksSet | None = None*) → None

shock_model_with (*direct_shocks: DirectShocksSet | list[DirectShocksSet], combine_mode: Literal['concat', 'merge'] = 'concat'*)

Apply direct shocks to the model, combining them if multiple shocks are provided.

Parameters

- **direct_shocks** (*DirectShocksSet or list of DirectShocksSet*) – The direct shocks to apply to the model. If a list is provided, the shocks will be combined based on the *combine_mode*.
- **combine_mode** (*{“concat”, “merge”}, optional*) – The method for combining multiple shocks. Default is “concat”. - “concat”: Concatenate shocks assuming independent events. - “merge”: Merge shocks by summing their impacts for matching (event,region,sector).

Raises

ValueError – If the MRIOT name of the provided shocks does not match the model’s MRIOT name.

Notes

- The *combine_mode* is only applicable if *direct_shocks* is provided as a list.
- The shocks must correspond to the same MRIOT used in the model.

class `climada_petals.engine.supplychain.StaticIOModel` (*mriot: IOSystem, direct_shocks: DirectShocksSet | None = None*)

Bases: *IndirectCostModel*

Static Input-Output Model for analyzing indirect economic impacts.

Extends and makes concrete the *IndirectCostModel* abstract class by providing methods to calculate degraded economic metrics and indirect impacts using Leontief and Ghosh models.

Parameters

- **mriot** (*pymrio.IOSystem*) – The Multi-Regional Input-Output System used in the model.
- **direct_shocks** (*DirectShocksSet or None, optional*) – The direct shocks to apply to the model. Default is None.

mriot

The Multi-Regional Input-Output System used in the model.

Type

`pymrio.IOSystem`

direct_shocks

The direct shocks applied to the model. None if no shocks have been applied.

Type

`DirectShocksSet` or `None`

`__init__` (*mriot*: `IOSystem`, *direct_shocks*: `DirectShocksSet` | `None` = `None`) → `None`

`calc_ghosh` (*event_ids*: `list[int]` | `Index` | `None` = `None`) → `DataFrame`

Computes indirect impacts using the Ghosh model.

Parameters

event_ids (`list[int]` or `pd.Index` or `None`, optional) – A list of event IDs to calculate impacts for. If `None`, all events are used.

Returns

A `DataFrame` of indirect outputs for the specified events.

Return type

`pd.DataFrame`

`calc_indirect_impacts` (*event_ids*: `list[int]` | `Index` | `Literal['with_impact']` | `None` = `'with_impact'`) → `DataFrame` | `None`

Calculates detailed indirect impacts using both Leontief and Ghosh models.

Parameters

event_ids (`"with_impact"` or `list[int]` or `pd.Index` or `None`, default `"with_impact"`) – A list of event IDs to calculate impacts for. Only events with non null impacts by default. If `None`, all events are used.

Returns

A `DataFrame` containing indirect impacts with various metrics and methods, or `None` if no shocks are defined for the model.

Return type

`pd.DataFrame` or `None`

`calc_leontief` (*event_ids*: `list[int]` | `Index` | `None` = `None`) → `DataFrame`

Computes indirect impacts using the Leontief model.

Parameters

event_ids (`list[int]` or `pd.Index` or `None`, optional) – A list of event IDs to calculate impacts for. If `None`, all events are used.

Returns

A `DataFrame` of indirect outputs for the specified events.

Return type

`pd.DataFrame`

`property degraded_final_demand`: `DataFrame` | `Series`

Calculates and returns the degraded final demand considering the direct shocks.

Returns

A series representing the degraded final demand by sector.

Return type
pd.Series

property degraded_value_added: DataFrame | Series

Calculates and returns the degraded value added considering the direct shocks.

Returns
A series representing the degraded value added by sector.

Return type
pd.Series

property final_demand: Series

Retrieves the (total) final demand addressed to each sectors from the MRIOT.

Returns
A series representing the final demand by sector.

Return type
pd.Series

property value_added: Series

Calculates and returns the value added from the MRIOT.

Returns
A series representing the value added by sector.

Return type
pd.Series

class climada_petals.engine.supplychain.**BoARIOModel** (*mriot: IOSystem, direct_shocks: DirectShocksSet, model_kwargs: Dict[str, Any] | None = None, simulation_kwargs: Dict[str, Any] | None = None, event_kwargs: Dict[str, Any] | None = None*)

Bases: *IndirectCostModel*

BoARIO Model for simulating the economic impacts of shocks using the ARIOT model.

Extends and makes concrete the IndirectCostModel abstract class by integrating the ARIOPsiModel and Simulation classes to simulate the effects of direct shocks over time, with customizable event parameters.

Parameters

- **mriot** (*pymrio.IOSystem*) – The Multi-Regional Input-Output System used in the model.
- **direct_shocks** (*DirectShocksSet*) – The direct shocks to apply to the model.
- **model_kwargs** (*dict, optional*) – A dictionary of model parameters to be passed to the ARIOPsiModel.
- **simulation_kwargs** (*dict, optional*) – A dictionary of simulation parameters to be passed to the Simulation class.
- **event_kwargs** (*dict, optional*) – A dictionary of event-specific parameters (e.g., recovery time) to be passed to the event creation logic.

mriot

The Multi-Regional Input-Output System used in the model.

Type
pymrio.IOSystem

model

The ARIIO model used to represent the economy.

Type

ARIOPsiModel

sim

The simulation object used to run the model.

Type

Simulation

direct_shocks

The direct shocks applied to the model.

Type

DirectShocksSet or None

Notes

We highly recommend users to go through *BoARIO's documentation* <<https://spjuhel.github.io/BoARIO>>

`__init__` (*mriot*: *IOSystem*, *direct_shocks*: *DirectShocksSet*, *model_kwargs*: *Dict[str, Any]* | None = None, *simulation_kwargs*: *Dict[str, Any]* | None = None, *event_kwargs*: *Dict[str, Any]* | None = None) → None

run_sim()

Run the simulation for the model.

Executes the simulation loop and returns the simulation object.

Returns

The simulation object after running the simulation loop.

Return type

Simulation

`shock_model_with` (*direct_shocks*: *DirectShocksSet* | *list[DirectShocksSet]*, *combine_mode*: *Literal['concat', 'merge']* = 'concat', *event_kwargs*: *Dict[str, Any]* | None = None)

Apply direct shocks to the model, adding events for recovery if necessary.

Parameters

- **direct_shocks** (*DirectShocksSet* or *list of DirectShocksSet*) – The direct shocks to apply to the model. If a list is provided, the shocks will be combined based on the *combine_mode*.
- **combine_mode** (*{“concat”, “merge”}*, *optional*) – The method for combining multiple shocks. Default is “concat”. - “concat”: Concatenate shocks assuming independent events. - “merge”: Merge shocks by summing their impacts.
- **event_kwargs** (*dict*, *optional*) – A dictionary of event-specific parameters to be passed to event creation. See BoARIO documentation on *Events* <<https://spjuhel.github.io/BoARIO/tutorials/boario-events.html>>

`climada_petals.engine.supplychain.get_mriot` (*mriot_type*: *str*, *mriot_year*: *int*, *redownload*: *bool* = False, *save*: *bool* = True)

Retrieves and optionally saves the Multi-Regional Input-Output Table (MRIOT) for a specified type and year. It handles downloading, parsing, and managing the directory structure for MRIOT files.

Files are stored in the climada data folder, within a “MRIOT” folder by default.

Parameters

- **mriot_type** (*str*) – The type of MRIOT to retrieve (e.g., “OECD23”).
- **mriot_year** (*int*) – The specific year of the MRIOT to retrieve.
- **redownload** (*bool, optional*) – Indicates whether to force redownload (and parsing) of the data (default is False).
- **save** (*bool, optional*) – Indicates whether to save the processed MRIOT data (default is True).

Returns

The loaded or newly parsed MRIOT represented as an IOSystem.

Return type

pymrio.IOSystem

Notes

The function manages both the downloading of the MRIOT data and the parsing of the data into an appropriate format. If the data for the specified year and type is already available, it will load the existing data unless redownload is set to True. If the data must be parsed, it can optionally be saved to the designated location.

2.1.3 climada_petals.engine.warn module

`climada_petals.engine.warn.dilation` (*bin_map, size*)

Dilate binary input map. The operation is based on a convolution. During translation of the filter, a point is included to the region (changed or kept to 1), if one or more elements correspond with the filter. Else, it is 0. This results in more and larger regions of interest. Larger filter sizes - more area of interest. For more information: https://scikit-image.org/docs/stable/auto_examples/applications/plot_morphology.html

Parameters

- **bin_map** (*np.ndarray*) – Rectangle 2d map of values which are used to generate the warning.
- **size** (*int*) – Size of filter.

Returns

Generated binary map with enlarged regions of interest.

Return type

`np.ndarray`

`climada_petals.engine.warn.erosion` (*bin_map, size*)

Erode binary input map. The operation is based on a convolution. During translation of the filter, a point is included to the region (changed or kept to 1), if all elements correspond with the filter. Else, it is 0. This results in less and smaller regions of interest and reduces heterogeneity in map. Larger sizes - more reduction. For more information: https://scikit-image.org/docs/stable/auto_examples/applications/plot_morphology.html

Parameters

- **bin_map** (*np.ndarray*) – Rectangle 2d map of values which are used to generate the warning.
- **size** (*int*) – Size of filter.

Returns

Generated binary map with reduced regions of interest.

Return type

`np.ndarray`

`climada_petals.engine.warn.median_filtering` (*bin_map*, *size*)

Smooth binary input map. The operation is based on a convolution. During translation of the filter, a point is included to the region (changed or kept to 1), if the median of the filter is 1. Else, it is 0. This results in smoother regions of interest and reduces heterogeneity in map. Larger sizes - smoother regions.

Parameters

- **bin_map** (*np.ndarray*) – Rectangle 2d map of values which are used to generate the warning.
- **size** (*int*) – Size of filter.

Returns

Generated binary map with smoothed regions of interest.

Return type

`np.ndarray`

class `climada_petals.engine.warn.Operation` (*value*)

Bases: `Enum`

Available Operations. Links operations to functions. More operations can be added.

dilation

Links to dilation operation.

Type

function

erosion

Links to erosion operation.

Type

function

median_filtering

Links to median filtering operation.

Type

function

dilation = `functools.partial`(<function dilation>)

erosion = `functools.partial`(<function erosion>)

median_filtering = `functools.partial`(<function median_filtering>)

class `climada_petals.engine.warn.Warn` (*warning*, *coord*, *warn_params*)

Bases: `object`

Warn definition. Generate a warning, i.e., 2D map of coordinates with assigned warn levels. Operations, their order, and their influence (filter sizes) can be selected to generate the warning. Further properties can be chosen which define the warning generation. The functionality of reducing heterogeneity in a map can be applied to different inputs, e.g. MeteoSwiss windstorm data (COSMO data), TCs, impacts, etc.

warning

Warning generated by warning generation algorithm. Warn level for every coordinate of map.

Type

`np.ndarray`

coord

Coordinates of warning map.

Type

np.ndarray

warn_levels

Warn levels that define the bins in which the input_map will be classified in. E.g., for windspeeds: [0, 10, 40, 80, 150, 200.0]

Type

list

```
class WarnParameters (warn_levels: ~typing.List[float], operations:  
    ~typing.List[~typing.Tuple[~climada_petals.engine.warn.Operation, int]] =  
    <factory>, gradual_decr: bool = False, change_sm: bool = False)
```

Bases: object

WarnParameters data class definition. It stores the relevant information needed during the warning generation. The operations and its sizes, as well as the algorithms properties \$(gradual decrease of warning levels and changing of small warning regions formed) are saved.

warn_levels

Warn levels that define the bins in which the input_map will be classified in.

Type

list

operations

Tuples saving operations and their filter sizes to be applied in filtering algorithm.

Type

list

gradual_decr

Defines 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 warn level 5 directly to 1).

Type

bool

change_sm

If strictly larger than 1, the levels of too small regions are changed to its surrounding levels. If 0 or None, the levels are not changed.

Type

int

```
warn_levels: List[float]
```

```
operations: List[Tuple[Operation, int]]
```

```
gradual_decr: bool = False
```

```
change_sm: bool = False
```

```
__init__ (warn_levels: ~typing.List[float], operations:  
    ~typing.List[~typing.Tuple[~climada_petals.engine.warn.Operation, int]] = <factory>,  
    gradual_decr: bool = False, change_sm: bool = False) → None
```

`__init__` (*warning, coord, warn_params*)

Initialize Warn.

Parameters

- **warning** (*np.ndarray*) – Warn level for every coordinate of input map.
- **coord** (*np.ndarray*) – Coordinates of warning map.
- **warn_params** (*WarnParameters*) – Contains information on how to generate the warning (operations and details).

classmethod `from_map` (*input_map, coord, warn_params*)

Generate Warn object from map (value (e.g., windspeeds at coordinates)).

Parameters

- **input_map** (*np.ndarray*) – Rectangle 2d map of values which are used to generate the warning.
- **coord** (*np.ndarray*) – Coordinates of warning map. For every value of the map exactly one coordinate is needed.
- **warn_params** (*WarnParameters*) – Contains information on how to generate the warning (operations and details).

Returns

warn – Generated Warn object including warning, coordinates, warn levels, and metadata.

Return type

Warn

classmethod `wind_from_cosmo` (*path_to_cosmo, warn_params, lead_time, quant_nr=0.7*)

Generate Warn object from COSMO windspeed data. The warn object is computed for the given date and time. The ensemble members of that date and time are grouped together to a single windspeed map.

Parameters

- **path_to_cosmo** (*string*) – Path including name to cosmo file.
- **warn_params** (*WarnParameters*) – Contains information on how to generate the warning (operations and details).
- **lead_time** (*datetime*) – Lead time when warning should be generated.
- **quant_nr** (*float*) – Quantile number to group ensemble members of COSMO wind speeds.

Returns

warn – Generated Warn object including warning, coordinates, warn levels, and metadata.

Return type

Warn

classmethod `from_hazard` (*hazard, warn_params*)

Generate Warn object from hazard object. The intensity map is used therefore. It needs to be transferable to a dense matrix, else the computation of the warning is impossible.

Parameters

- **hazard** (*Hazard*) – Contains the information of which to generate a warning from.
- **warn_params** (*WarnParameters*) – Contains information on how to generate the warning (operations and details).

Returns

warn – Generated Warn object including warning, coordinates, warn levels, and metadata.

Return type

Warn

static bin_map (*input_map*, *levels*)

Bin every value of input map into given levels.

Parameters

- **input_map** (*np.ndarray*) – Array containing data to generate binned map of.
- **levels** (*list*) – List with levels to bin input map.

Returns

binned_map – Map of binned values in levels, same shape as input map.

Return type

np.ndarray

static zeropadding (*lat*, *lon*, *val*, *res_rel_error=0.01*)

Produces a rectangular shaped map from a non-rectangular map (e.g., country). For this, a regular grid is created in the rectangle enclosing the non-rectangular map. The values are mapped onto the regular grid according to their coordinates. The regular grid is filled with zeros where no values are defined. This only works if the lat lon values of the non-rectangular map can be accurately represented on a grid with a regular resolution.

Parameters

- **lat** (*list*) – Latitudes of values of map.
- **lon** (*list*) – Longitudes of values of map.
- **val** (*list*) – Values of quantity of interest at every coordinate given.
- **res_rel_error** (*float*) – defines the relative error in terms of grid resolution by which the lat lon values of the returned *coord_rec* can maximally differ from the provided lat lon values. Default: 0.01

Returns

- **map_rec** (*np.ndarray*) – Rectangular map with a value for every grid point. Padded with zeros where no values in input map.
- **coord_rec** (*np.ndarray*) – Longitudes and Latitudes of every value of the map.

plot_warning (*var_name='Warn Levels'*, *title='Categorical Warning Map'*, *cat_name=None*, *adapt_fontsize=True*, ***kwargs*)

Map plots for categorical data defined in array(s) over input coordinates. The categories must be a finite set of unique values as can be identified by *np.unique()* (mix of int, float, strings, ...).

The categories are shared among all subplots, i.e. are obtained from *np.unique(array_sub)*. Eg.:

```
>>> array_sub = [[1, 2, 1.0, 2], [1, 2, 'a', 'a']]
>>> -> category mapping is [[0, 2, 1, 2], [0, 2, 3, 3]]
```

Same category: 1 and '1' Different categories: 1 and 1.0

This method wraps around *util.geo_scatter_from_array* and uses all its args and kwargs.

Parameters

- **var_name** (*str or list(str)*) – label to be shown in the colorbar. If one provided, the same is used for all subplots. Otherwise provide as many as subplots in array_sub.
- **title** (*str or list(str)*) – subplot title. If one provided, the same is used for all subplots. Otherwise provide as many as subplots in array_sub.
- **cat_name** (*dict, optional*) – Categories name for the colorbar labels. Keys are all the unique values in array_sub, values are their labels. The default is labels = unique values.
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- ****kwargs** – Arbitrary keyword arguments for hexbin matplotlib function

Return type

cartopy.mpl.geoaxes.GeoAxesSubplot

```
plot_warning_meteoswiss_style (var_name='Warn Levels', title='Categorical Warning Map',
                               cat_name=None, adapt_fontsize=True)
```

Map plots for categorical data defined in array(s) over input coordinates. The MeteoSwiss coloring scheme is used, therefore only 5 warn levels are allowed.

This method wraps around util.geo_scatter_from_array and uses all its args and kwargs.

Parameters

- **var_name** (*str or list(str)*) – label to be shown in the colorbar. If one provided, the same is used for all subplots. Otherwise provide as many as subplots in array_sub.
- **title** (*str or list(str)*) – subplot title. If one provided, the same is used for all subplots. Otherwise provide as many as subplots in array_sub.
- **cat_name** (*dict, optional*) – Categories name for the colorbar labels. Keys are all the unique values in array_sub, values are their labels. The default is labels = unique values.
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- ****kwargs** – Arbitrary keyword arguments for hexbin matplotlib function

Return type

cartopy.mpl.geoaxes.GeoAxesSubplot

2.2 climada_petals.entity package

2.2.1 climada_petals.entity.exposures package

climada_petals.entity.exposures.black_marble module

```
class climada_petals.entity.exposures.black_marble.BlackMarble (*args, meta=None,
                                                               nightlight_file=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

```
__init__ (*args, meta=None, nightlight_file=None, **kwargs)
```

Parameters

- **data** (*dict, iterable, DataFrame, GeoDataFrame, ndarray*) – data of the initial DataFrame, see `pandas.DataFrame()`. Used to initialize values for “region_id”, “category_id”, “cover”, “deductible”, “value”, “geometry”, “impf_[hazard type]”.
- **columns** (*Index or array, optional*) – Columns of the initial DataFrame, see `pandas.DataFrame()`. To be provided if *data* is an array
- **index** (*Index or array, optional*) – Columns of the initial DataFrame, see `pandas.DataFrame()`. can optionally be provided if *data* is an array or for defining a specific row index
- **dtype** (*dtype, optional*) – data type of the initial DataFrame, see `pandas.DataFrame()`. Can be used to assign specific data types to the columns in *data*
- **copy** (*bool, optional*) – Whether to make a copy of the input *data*, see `pandas.DataFrame()`. Default is False, i.e. by default *data* may be altered by the Exposures object.
- **geometry** (*array, optional*) – Geometry column, see `geopandas.GeoDataFrame()`. Must be provided if *lat* and *lon* are None and *data* has no “geometry” column.
- **crs** (*value, optional*) – Coordinate Reference System, see `geopandas.GeoDataFrame()`.
- **meta** (*dict, optional*) – Metadata dictionary. Default: {} (empty dictionary). May be used to provide any of *description*, *ref_year*, *value_unit* and *crs*
- **description** (*str, optional*) – Default: None
- **ref_year** (*int, optional*) – Reference Year. Defaults to the entry of the same name in *meta* or 2018.
- **value_unit** (*str, optional*) – Unit of the exposed value. Defaults to the entry of the same name in *meta* or ‘USD’.
- **value** (*array, optional*) – Exposed value column. Must be provided if *data* has no “value” column
- **lat** (*array, optional*) – Latitude column. Can be provided together with *lon*, alternative to *geometry*
- **lon** (*array, optional*) – Longitude column. Can be provided together with *lat*, alternative to *geometry*

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'}, 'ril': {'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,
crop=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

`__init__` (*args, meta=None, crop=None, **kwargs)

Parameters

- **data** (*dict, iterable, DataFrame, GeoDataFrame, ndarray*) – data of the initial `DataFrame`, see `pandas.DataFrame()`. Used to initialize values for “region_id”, “category_id”, “cover”, “deductible”, “value”, “geometry”, “impf_[hazard type]”.
- **columns** (*Index or array, optional*) – Columns of the initial `DataFrame`, see `pandas.DataFrame()`. To be provided if *data* is an array
- **index** (*Index or array, optional*) – Columns of the initial `DataFrame`, see `pandas.DataFrame()`. can optionally be provided if *data* is an array or for defining a specific row index
- **dtype** (*dtype, optional*) – data type of the initial `DataFrame`, see `pandas.DataFrame()`. Can be used to assign specific data types to the columns in *data*
- **copy** (*bool, optional*) – Whether to make a copy of the input *data*, see `pandas.DataFrame()`. Default is `False`, i.e. by default *data* may be altered by the `Exposures` object.
- **geometry** (*array, optional*) – Geometry column, see `geopandas.GeoDataFrame()`. **Must** be provided if *lat* and *lon* are `None` and *data* has no “geometry” column.
- **crs** (*value, optional*) – Coordinate Reference System, see `geopandas.GeoDataFrame()`.
- **meta** (*dict, optional*) – Metadata dictionary. Default: `{}` (empty dictionary). May be used to provide any of *description*, *ref_year*, *value_unit* and *crs*
- **description** (*str, optional*) – Default: `None`
- **ref_year** (*int, optional*) – Reference Year. Defaults to the entry of the same name in *meta* or 2018.
- **value_unit** (*str, optional*) – Unit of the exposed value. Defaults to the entry of the same name in *meta* or ‘USD’.
- **value** (*array, optional*) – Exposed value column. **Must** be provided if *data* has no “value” column
- **lat** (*array, optional*) – Latitude column. Can be provided together with *lon*, alternative to *geometry*
- **lon** (*array, optional*) – Longitude column. Can be provided together with *lat*, alternative to *geometry*

`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 crop yield 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*) – File Name STRing depending on VARiable and ISIMIP simulation 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),
                                    in-
                                    put_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 `CropProduction.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/clinada/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 `CropProduction.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 simulation 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 function `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,
re-
turn_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 initiated 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` (*data=None, index=None, columns=None, dtype=None, copy=False, geometry=None, crs=None, meta=None, description=None, ref_year=None, value_unit=None, value=None, lat=None, lon=None*)

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.osm_data_loader module

class `climada_petals.entity.exposures.osm_data_loader.OSMApiQuery` (*area, condition*)

Bases: `object`

Queries features directly via the overpass turbo API.

Parameters

- **area** (*tuple (ymin, xmin, ymax, xmax)*)
- **condition** (*str*) – must be of format `[“key”]` or `[“key”=“value”]`, etc.

Note

The area (bounding box) ordering in the overpass query language is different from the convention in shapely / geopandas. If you directly pass area as bbox, make sure the order is (ymin, xmin, ymax, xmax). If you use a classib bbox from shapely or geopandas, use the `from_bounding_box()` method, which reorders the inputs!

`__init__` (*area, condition*)

classmethod `from_bounding_box` (*bbox, condition*)

Parameters

- **bbox** (*tuple*) – bbox as given from the standard convention of a shapely / geopandas
- **bounding box as (xmin, ymin, xmax, ymax)**
- **condition** (*str*) – must be of format `[“key”]` or `[“key”=“value”]`, etc.

classmethod `from_polygon` (*polygon, condition*)

Parameters

- **polygon** (*shapely.geometry.polygon*)
- **condition** (*str*) – must be of format `[“key”]` or `[“key”=“value”]`, etc.

get_data_overpass (*closed_lines_are_polys=True*)

wrapper for all helper funcs to get & assemble data

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_allokey_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 (data=None, index=None,  
                                                           columns=None, dtype=None,  
                                                           copy=False, geometry=None,  
                                                           crs=None, meta=None,  
                                                           description=None, ref_year=None,  
                                                           value_unit=None, value=None,  
                                                           lat=None, lon=None)
```

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 additional 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

2.2.2 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 definition 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 supported in 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_jrc_region_sector` (*region, sector='residential'*)

Create a new `ImpfRiverFlood` object based on the specified world region and sector. Impact functions come from the following JRC publication:

Huizinga, J., De Moel, H. and Szewczyk, W., Global flood depth-damage functions: Methodology and the database with guidelines, EUR 28552 EN, Publications Office of the European Union, Luxembourg, 2017, ISBN 978-92-79-67781-6, doi:10.2760/16510, JRC105688.

Notes

The impact functions assess percentage losses at 0, 0.5, 1, 1.5, 2, 3, 4, 5, 6 meters of water. For North America, percentage losses higher than 0 are already registered at 0 meters of water, because it accounts for the presence of basements (see main publication). Since this could be problematic when computing impacts, as one would have losses also when there is no flood, a 0.05 meters of water point is added to all functions (not only North America, for consistency), and this corresponds to the 0 meters point in the JRC functions.

Parameters

- **region** (*str*) – world region for which the impact function was defined. Supported values:
- “Africa”, “Asia”, “Europe”, “North America”, “Oceania”, “South America”.
- **sector** (*str*) – sector for which the impact function was defined. Supported values: “residential”, “commercial”, “industrial”, “transport”, “infrastructure”, “agriculture”.

Returns

impf – New `ImpfRiverFlood` object with parameters for the specified world region and sector.

Return type

ImpfRiverFlood

Raises

ValueError –

set_RF_Impf_Africa (*args, **kwargs)

This function is deprecated, use ImpfRiverFlood.from_region_sector instead.

set_RF_Impf_Asia (*args, **kwargs)

This function is deprecated, use ImpfRiverFlood.from_region_sector instead.

set_RF_Impf_Europe (*args, **kwargs)

This function is deprecated, use ImpfRiverFlood.from_region_sector instead.

set_RF_Impf_NorthAmerica (*args, **kwargs)

This function is deprecated, use ImpfRiverFlood.from_region_sector instead.

set_RF_Impf_Oceania (*args, **kwargs)

This function is deprecated, use ImpfRiverFlood.from_region_sector instead.

set_RF_Impf_SouthAmerica (*args, **kwargs)

This function is deprecated, use ImpfRiverFlood.from_region_sector 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.

2.3 climada_petals.hazard package

2.3.1 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

2.3.2 River Flood from GloFAS River Discharge Data Module

Main Module

```
class climada_petals.hazard.rf_glofas.river_flood_computation.RiverFloodCachePaths (discharge,
                                                                    re-
                                                                    turn_period,
                                                                    re-
                                                                    turn_period_regrid,
                                                                    re-
                                                                    turn_period_regrid_protect)
```

Bases: *RiverFloodCachePaths*

Container for storing paths to caches for *RiverFloodInundation*

Depending on the state of the corresponding *RiverFloodInundation* instance, files might be present or not. Please check this explicitly before accessing them.

discharge

Path to the discharge data cache.

Type

`pathlib.Path`

return_period

Path to the return period data cache.

Type

`pathlib.Path`

return_period_regrid

Path to the regrided return period data cache.

Type

`pathlib.Path`

return_period_regrid_protect

Path to the regrided return period data cache, where the return period was restricted by the protection limits.

Type

`pathlib.Path`

classmethod `from_dir` (*cache_dir: Path*)

Set default paths from a cache directory

```

class climada_petals.hazard.rf_glofas.river_flood_computation.RiverFloodInundation (store_intermediates:
    bool
    =
    True,
    data_dir:
    Path
    | str
    =
    PosixPath('/home/docs/c
    flood-
    computation'),
    cache_dir:
    Path
    | str
    =
    PosixPath('/home/docs/c
    flood-
    computation/.cache'))

```

Bases: object

Class for computing river flood inundations

store_intermediates

If intermediate results are stored in the respective *cache_paths*

Type

bool

cache_paths

Paths pointing to potential intermediate results stored in a cache directory.

Type

RiverFloodCachePaths

flood_maps

Flood inundation on lat/lon grid for specific return periods.

Type

xarray.DataArray

gumbel_fits

Gumbel parameters resulting from extreme value analysis of historical discharge data.

Type

xarray.Dataset

flopros

Spatially explicit information on flood protection levels.

Type

geopandas.GeoDataFrame

regridder

Storage for re-using the XESMF regridder in case the computation is repeated on the same grid. This reduces the runtime of subsequent computations.

Type

xesmf.Regridder

```
__init__ (store_intermediates: bool = True, data_dir: Path | str =
    PosixPath('/home/docs/climada/data/river-flood-computation'), cache_dir: Path | str =
    PosixPath('/home/docs/climada/data/river-flood-computation/.cache'))
```

Initialize the instance

Parameters

- **store_intermediates** (*bool, optional*) – Whether the data of each computation step should be stored in the cache directory. This is recommended especially for larger data. Only set this to `False` if the data operated on is very small (e.g., for a small country or region). Defaults to `True`.
- **data_dir** (*Path or str, optional*) – The directory where flood maps, Gumbel distribution parameters and the FLOPROS database are located. Defaults to `<climada>/data/river-flood-computation`, where `<climada>` is the Climada data directory indicated by `local_data : system` in the `climada.conf`. This directory must exist.
- **cache_dir** (*Path or str, optional*) – The top-level cache directory where computation caches of this instance will be placed. Defaults to `<climada>/data/river-flood-computation/.cache` (see above for configuration). This directory (and all its parents) will be created.

```
clear_cache ()
```

Clear the cache of this instance

This will delete the temporary cache directory and create a new one by calling `_create_tempdir()`.

```
compute (discharge: DataArray | None = None, apply_protection: bool | str = 'both', resample_kws: Mapping[str, Any] | None = None, regrid_kws: Mapping[str, Any] | None = None)
```

Compute river flood inundation from downloaded river discharge

After downloading discharge data, this will execute the pipeline for computing river flood inundation. This pipeline has the following steps:

- Compute the equivalent return period, either with `return_period()`, or `return_period_resample()`.
- Regrid the return period data onto the grid of the flood hazard maps with `regrid()`.
- *Optional*: Apply the protection layer with `apply_protection()`.
- Compute the flood depth by interpolating flood hazard maps with `flood_depth()`.

Resampling, regridding, and the application of protection information are controlled via the parameters of this method.

Parameters

- **discharge** (*xr.DataArray or None (optional)*) – The discharge data to compute flood depths for. If `None`, the cached discharge will be used. Defaults to `None`.
- **apply_protection** (*bool or "both" (optional)*) – If the stored protection layer should be considered when computing the flood depth. If `"both"`, this method will return a dataset with two flood depth arrays. Defaults to `both`.
- **resample_kws** (*Mapping (str, Any) or None (optional)*) – Keyword arguments for `return_period_resample()`. If `None`, this method will call `return_period()`. Otherwise, it will call `return_period_resample()` and pass this parameter as keyword arguments. Defaults to `None`.
- **regrid_kws** (*Mapping (str, Any) or None (optional)*) – Keyword arguments for `regrid()`. Defaults to `None`.

Returns

Dataset containing the flood data with the same dimensions as the input discharge data. Depending on the choice of `apply_protection`, this will contain one or two DataArrays.

Return type

`xr.Dataset`

Raises

RuntimeError – If `discharge` is `None`, but no discharge data is cached.

`download_forecast` (*countries: str | List[str], forecast_date: str | datetime64 | datetime | Timestamp, lead_time_days: int = 10, preprocess: Callable | None = None, **download_glofas_discharge_kwargs*) → DataArray

Download GloFAS discharge ensemble forecasts

If `store_intermediates` is true, the returned data is also stored in `cache_paths`.

Parameters

- **countries** (*str or list of str*) – Names or codes of countries to download data for. The downloaded data will be a lat/lon grid covering all specified countries.
- **forecast_date** – The date at which the forecast was issued. Can be defined any way that is compatible with `pandas.Timestamp`, see <https://pandas.pydata.org/docs/reference/api/pandas.Timestamp.html>
- **lead_time_days** (*int, optional*) – How many days of lead time to include in the downloaded forecast. Maximum is 30. Defaults to 10, in which case the 10 days following the `forecast_date` are included in the download.
- **preprocess** – Callable for preprocessing data while loading it. See https://docs.xarray.dev/en/stable/generated/xarray.open_mfdataset.html
- **download_glofas_discharge_kwargs** – Additional arguments to `climada_petals.hazard.rf_glofas.transform_ops.download_glofas_discharge()`

Returns

forecast – Downloaded forecast as DataArray after preprocessing

Return type

`xr.DataArray`

See also

`climada_petals.hazard.rf_glofas.transform_ops.download_glofas_discharge()`

`download_reanalysis` (*countries: str | Iterable[str], year: int, preprocess: Callable | None = None, **download_glofas_discharge_kwargs*)

Download GloFAS discharge historical data

If `store_intermediates` is true, the returned data is also stored in `cache_paths`.

Parameters

- **countries** (*str or list of str*) – Names or codes of countries to download data for. The downloaded data will be a lat/lon grid covering all specified countries.
- **year** (*int*) – The year to download data for.
- **preprocess** – Callable for preprocessing data while loading it. See https://docs.xarray.dev/en/stable/generated/xarray.open_mfdataset.html

- **download_glofas_discharge_kwargs** – Additional arguments to `climada_petals.hazard.rf_glofas.transform_ops.download_glofas_discharge()`

Returns

reanalysis – Downloaded forecast as DataArray after preprocessing

Return type

`xr.DataArray`

 **See also**

`climada_petals.hazard.rf_glofas.transform_ops.download_glofas_discharge()`

return_period (*discharge: DataArray | None = None*) → DataArray

Compute the return period for a given discharge

If no discharge data is given as parameter, the discharge cache will be accessed.

If `store_intermediates` is true, the returned data is also stored in `cache_paths`.

Parameters

discharge (*xr.DataArray, optional*) – The discharge data to operate on. Defaults to `None`, which indicates that data should be loaded from the cache

Returns

r_period – Return period for each location of the input discharge.

Return type

`xr.DataArray`

 **See also**

`climada_petals.hazard.rf_glofas.transform_ops.return_period()`

return_period_resample (*num_bootstrap_samples: int, discharge: DataArray | None = None, fit_method: str = 'MM', num_workers: int = 1, memory_per_worker: str = '2G'*)

Compute the return period for a given discharge using bootstrap sampling.

For each input discharge value, this creates an ensemble of return periods by employing bootstrap sampling. The ensemble size is controlled with `num_bootstrap_samples`.

If `store_intermediates` is true, the returned data is also stored in `cache_paths`.

Parameters

- **num_bootstrap_samples** (*int*) – Number of bootstrap samples to compute for each discharge value.
- **discharge** (*xr.DataArray, optional*) – The discharge data to operate on. Defaults to `None`, which indicates that data should be loaded from the cache.
- **fit_method** (*str, optional*) – Method for fitting data to bootstrapped samples.
 - "MM": Method of Moments
 - "MLE": Maximum Likelihood Estimation
- **num_workers** (*int, optional*) – Number of parallel processes to use when computing the samples.

- **memory_per_worker** (*str, optional*) – Memory to allocate for each process.

Returns

r_period – Return period samples for each location of the input discharge.

Return type

xr.DataArray

 **See also**

`climada_petals.hazard.rf_glofas.transform_ops.return_period_resample()`

regrid (*r_period: DataArray | None = None, method: str = 'bilinear', reuse_regridder: bool = False*)

Regrid the return period data onto the flood hazard map grid.

This computes the regridding matrix for the given coordinates and then performs the actual regridding. The matrix is stored in `regridder`. If another regridding is performed on the same grid (but possibly different data), the regridder can be reused to save time. To control that, set `reuse_regridder=True`.

If `store_intermediates` is true, the returned data is also stored in `cache_paths`.

Parameters

- **r_period** (*xr.DataArray, optional*) – The return period data to regrid. Defaults to `None`, which indicates that data should be loaded from the cache.
- **method** (*str, optional*) – Interpolation method of the return period data. Defaults to "bilinear". See https://xesmf.readthedocs.io/en/stable/notebooks/Compare_algorithms.html
- **reuse_regridder** (*bool, optional*) – Reuse the regridder stored if one is stored. Defaults to `False`, which means that a new regridder is always built when calling this function. If `True`, and no regridder is stored, it will be built nonetheless.

Returns

return_period_regrid – The regridded return period data.

Return type

xr.DataArray

 **See also**

`climada_petals.hazard.rf_glofas.transform_ops.regrid()`

apply_protection (*return_period_regrid: DataArray | None = None*)

Limit the return period data by applying FLOPROS protection levels.

This sets each return period value where the local FLOPROS protection level is not exceeded to NaN and returns the result. Protection levels are read from `flopros`.

If `store_intermediates` is true, the returned data is also stored in `cache_paths`.

Parameters

return_period_regrid (*xr.DataArray, optional*) – The return period data to regrid. Defaults to `None`, which indicates that data should be loaded from the cache.

Returns

return_period_regrid_protect – The regridded return period where each value that does not reach the protection limit is set to NaN.

Return type

xr.DataArray

See also`climada_petals.hazard.rf_glofas.transform_ops.apply_flopros()`**flood_depth** (*return_period_regrid: DataArray | None = None*)

Compute the flood depth from regrided return period data.

Interpolate the flood hazard maps stored in `:py:attr`flood_maps`` in the return period dimension at every location to compute the flood footprint.**Note**Even if `store_intermediates` is true, the returned data is **not** stored automatically! Use `climada_petals.hazard.rf_glofas.transform_ops.save_file()` to store the data yourself.**Parameters****return_period_regrid** (*xr.DataArray, optional*) – The regrided return period data to use for computing the flood footprint. Defaults to `None` which indicates that data should be loaded from the cache. If `RiverFloodCachePaths.return_period_regrid_protect` exists, that data is used. Otherwise, the “unprotected” data `RiverFloodCachePaths.return_period_regrid` is loaded.**Returns****inundation** – The flood inundation at every location of the flood hazard maps grid.**Return type**

xr.DataArray

Transformation Operations`climada_petals.hazard.rf_glofas.transform_ops.sel_lon_lat_slice` (*target: DataArray, source: DataArray*) → DataArray

Select a lon/lat slice from ‘target’ using coordinates of ‘source’

`climada_petals.hazard.rf_glofas.transform_ops.rp_comp` (*sample: ndarray, loc: ndarray, scale: ndarray, max_rp: float | None = inf*)

Compute the return period from a right-handed Gumbel distribution

All parameters can be arrays, in which case numpy broadcasting rules apply.

The return period of a sample x from an extreme value distribution is defined as $(1 - \text{cdf}(x))^{-1}$, where `cdf` is the cumulative distribution function of said distribution.**Parameters**

- **sample** (*array*) – Samples for which to compute the return period
- **loc** (*array*) – Loc parameter of the Gumbel distribution
- **scale** (*array*) – Scale parameter of the distribution
- **max_rp** (*float, optional*) – The maximum value of return periods. This avoids returning infinite values. Defaults to `np.inf` (no maximum).

Returns

The return period(s) for the input parameters

Return type

np.ndarray

`climada_petals.hazard.rf_glofas.transform_ops.reindex` (*target: DataArray, source: DataArray, tolerance: float | None = None, fill_value: float = nan, assert_no_fill_value: bool = False*) → DataArray

Reindex target to source with nearest neighbor lookup

Parameters

- **target** (*xr.DataArray*) – Array to be reindexed.
- **source** (*xr.DataArray*) – Array whose coordinates are used for reindexing.
- **tolerance** (*float (optional)*) – Maximum distance between coordinates. If it is superseded, the `fill_value` is inserted instead of the nearest neighbor value. Defaults to NaN
- **fill_value** (*float (optional)*) – The fill value to use if coordinates are changed by a distance of more than `tolerance`.
- **assert_no_fill_value** (*bool (optional)*) – Throw an error if fill values are found in the data after reindexing. This will also throw an error if the fill value is present in the `target` before reindexing (because the check afterwards would else not make sense)

Returns

target – Target reindexed like ‘source’ with nearest neighbor lookup for the data.

Return type

xr.DataArray

Raises

- **ValueError** – If tolerance is exceeded when reindexing, in case `assert_no_fill_value` is True.
- **ValueError** – If `target` already contains the `fill_value` before reindexing, in case `assert_no_fill_value` is True.

`climada_petals.hazard.rf_glofas.transform_ops.merge_flood_maps` (*flood_maps: Mapping[str, DataArray]*) → DataArray

Merge the flood maps GeoTIFFs into one NetCDF file

Adds a “zero” flood map (all zeros)

Parameters

flood_maps (*dict(str, xarray.DataArray)*) – The mapping of GeoTIFF file paths to respective DataArray. Each flood map is identified through the folder containing it. The folders are expected to follow the naming scheme `floodMapGL_rpXXXy`, where XXX indicates the return period of the respective map.

`climada_petals.hazard.rf_glofas.transform_ops.fit_gumbel_r` (*input_data: DataArray, time_dim: str = 'year', fit_method: str = 'MM', min_samples: int = 2*)

Fit a right-handed Gumbel distribution to the data

Parameters

- **input_data** (*xr.DataArray*) – The input time series to compute the distributions for. It must contain the dimension specified as `time_dim`.

- **time_dim** (*str*) – The dimension indicating time. Defaults to `year`.
- **fit_method** (*str*) – The method used for computing the distribution. Either `MLE` (Maximum Likelihood Estimation) or `MM` (Method of Moments).
- **min_samples** (*int*) – The number of finite samples along the time dimension required for a successful fit. If there are fewer samples, the fit result will be `NaN`.

Returns

A dataset on the same grid as the input data with variables

- `loc`: The loc parameter of the fitted distribution (mode)
- `scale`: The scale parameter of the fitted distribution
- `samples`: The number of samples used to fit the distribution at this coordinate

Return type

`xr.Dataset`

```
climada_petals.hazard.rf_glofas.transform_ops.download_glofas_discharge(product: str,
                                date_from: str,
                                date_to: str | None,
                                num_proc: int = 1,
                                download_path: str |
                                Path =
                                PosixPath('/home/docs/clinada/data/cds_glofas_discharge/'),
                                countries: List[str] |
                                str | None = None,
                                preprocess: Callable
                                | None = None,
                                open_mfdataset_kw:
                                Mapping[str, Any] |
                                None = None,
                                **request_kwargs)
                                → DataArray
```

Download the GloFAS data and return the resulting dataset

Several parameters are passed directly to `climada_petals.hazard.rf_glofas.cds_glofas_downloader.glofas_request()`. See this functions documentation for further information.

Parameters

- **product** (*str*) – The string identifier of the product to download. See `climada_petals.hazard.rf_glofas.cds_glofas_downloader.glofas_request()` for supported products.
- **date_from** (*str*) – Earliest date to download. Specification depends on the `product` chosen.
- **date_to** (*str or None*) – Latest date to download. If `None`, only download the `date_from`. Specification depends on the `product` chosen.
- **num_proc** (*int*) – Number of parallel processes to use for downloading. Defaults to 1.
- **download_path** (*str or pathlib.Path*) – Directory to store the downloaded data. The directory (and all required parent directories!) will be created if it does not yet exist. Defaults to `~/clinada/data/glofas-discharge/`.
- **countries** (*str or list of str, optional*) – Countries to download data for. Uses the maximum extension of all countries for selecting the latitude/longitude range of data to download.

- **preprocess** (*str, optional*) – String expression for preprocessing the data before merging it into one dataset. Must be valid Python code. The downloaded data is passed as variable `x`.
- **open_mfdataset_kw** (*dict, optional*) – Optional keyword arguments for the `xarray.open_mfdataset` function.
- **request_kwargs** – Keyword arguments for the Copernicus data store request.

```
climada_petals.hazard.rf_glofas.transform_ops.max_from_isel (array: DataArray, dim: str,
                                                           selections: List[Iterable | slice]) →
                                                           DataArray
```

Compute the maximum over several selections of an array dimension

```
climada_petals.hazard.rf_glofas.transform_ops.return_period (discharge: DataArray, gev_loc:
                                                             DataArray, gev_scale: DataArray,
                                                             max_return_period: float =
                                                             10000.0) → DataArray
```

Compute the return period for a discharge from a Gumbel EV distribution fit

Coordinates of the three datasets must match up to a tolerance of 1e-3 degrees. If they do not, an error is thrown.

Parameters

- **discharge** (*xr.DataArray*) – The discharge values to compute the return period for
- **gev_loc** (*xr.DataArray*) – The loc parameters for the Gumbel EV distribution
- **gev_scale** (*xr.DataArray*) – The scale parameters for the Gumbel EV distribution

Returns

The equivalent return periods for the input discharge and Gumbel EV distributions

Return type

`xr.DataArray`

➔ See also

```
climada_petals.hazard.rf_glofas.transform_ops.rp(),           climada_petals.hazard.
rf_glofas.transform_ops.return_period_resample()
```

```
climada_petals.hazard.rf_glofas.transform_ops.return_period_resample (discharge: DataArray,
                                                                        gev_loc: DataArray,
                                                                        gev_scale: DataArray,
                                                                        gev_samples:
                                                                        DataArray,
                                                                        bootstrap_samples: int,
                                                                        max_return_period:
                                                                        float = 10000.0,
                                                                        fit_method: str = 'MLE')
                                                                        → DataArray
```

Compute resampled return periods for a discharge from a Gumbel EV distribution fit

This function uses bootstrap resampling to incorporate the uncertainty in the EV distribution fit. Bootstrap resampling takes the fitted distribution, draws `N` samples from it (where `N` is the number of samples originally used to fit the distribution), and fits a new distribution onto these samples. This “bootstrapped” distribution is then used to compute the return period. Repeating this process yields an ensemble of distributions that captures the uncertainty in the original distribution fit.

Coordinates of the three datasets must match up to a tolerance of 1e-3 degrees. If they do not, an error is thrown.

Parameters

- **discharge** (*xr.DataArray*) – The discharge values to compute the return period for
- **gev_loc** (*xr.DataArray*) – The loc parameters for the Gumbel EV distribution
- **gev_scale** (*xr.DataArray*) – The scale parameters for the Gumbel EV distribution
- **gev_samples** (*xr.DataArray*) – The samples used to fit the Gumbel EV distribution at every point
- **bootstrap_samples** (*int*) – The number of bootstrap samples to compute. Increasing this will improve the representation of uncertainty, but strongly increase computational costs later on.
- **fit_method** (*str*) – Method for fitting the Gumbel EV during resampling. Available methods are the Method of Moments `MM` or the Maximum Likelihood Estimation `MLE` (default).

Returns

The equivalent return periods for the input discharge and Gumbel EV distributions. The data array will have an additional dimension `sample`, representing the bootstrap samples for every point.

Return type

`xr.DataArray`

➔ See also

`climada_petals.hazard.rf_glofas.transform_ops.rp()`, `climada_petals.hazard.rf_glofas.transform_ops.return_period()`

`climada_petals.hazard.rf_glofas.transform_ops.interpolate_space` (*return_period: DataArray, flood_maps: DataArray, method: str = 'linear'*) → `DataArray`

Interpolate the return period in space onto the flood maps grid

`climada_petals.hazard.rf_glofas.transform_ops.regrid` (*return_period: DataArray, flood_maps: DataArray, method: str = 'bilinear', regridder: xesmf.Regridder | None = None, return_regridder: bool = False*) → `DataArray | Tuple[DataArray, xesmf.Regridder]`

Regrid the return period onto the flood maps grid

`climada_petals.hazard.rf_glofas.transform_ops.apply_flopros` (*flopros_data: GeoDataFrame, return_period: DataArray | Dataset, layer: str = 'MerL_Riv'*) → `DataArray | Dataset`

Restrict the given return periods using FLOPROS data

The FLOPROS database describes the regional protection to river flooding based on a return period. Users can choose from different database layers. For each coordinate in `return_period`, the value from the respective FLOPROS database layer is selected. Any `return_period` lower than or equal to the FLOPROS protection value is discarded and set to `NaN`.

Parameters

- **flopros_data** (*PassthroughContainer*) – The FLOPROS data bundled into a `dantro.containers.PassthroughContainer`

- **return_period** (*xr.DataArray* or *xr.Dataset*) – The return periods to be restricted by the FLOPROS data
- **layer** (*str*) – The FLOPROS database layer to evaluate

Returns

The `return_period` data with all values below the local FLOPROS protection threshold set to NaN.

Return type

`xr.DataArray` or `xr.Dataset`

```
climada_petals.hazard.rf_glofas.transform_ops.flood_depth (return_period: Dataset | DataArray,
                                                         flood_maps: DataArray) → Dataset |
                                                         DataArray
```

Compute the flood depth from a return period and flood maps.

At each lat/lon coordinate, take the return period(s) and use them to interpolate the flood maps at this position in the return period dimension. Take the interpolated values and return them as flood hazard footprints. Works with arbitrarily high dimensions in the `return_period` array. Interpolation is linear.

Parameters

- **return_period** (*xr.DataArray* or *xr.Dataset*) – The return periods for which to compute the flood depth. If this is a dataset, the function will compute the flood depth for each variable.
- **flood_maps** (*xr.DataArray*) – Maps that indicate flood depth at latitude/longitude/return period coordinates.

Returns

The flood depths for the given return periods.

Return type

`xr.DataArray` or `xr.Dataset`

```
climada_petals.hazard.rf_glofas.transform_ops.save_file (data: Dataset | DataArray, output_path:
                                                         Path | str, encoding: Mapping[str, Any] |
                                                         None = None, engine: str | None =
                                                         'netcdf4', **encoding_defaults)
```

Save xarray data as a file with default compression

Calls `data.to_netcdf`. See https://docs.xarray.dev/en/stable/generated/xarray.Dataset.to_netcdf.html for the documentation of the underlying method.

Parameters

- **data** (*xr.Dataset* or *xr.Dataarray*) – The data to be stored in the file
- **output_path** (*pathlib.Path* or *str*) – The file path to store the data into. If it does not contain a suffix, `.nc` is automatically appended. The enclosing folder must already exist.
- **encoding** (*dict (optional)*) – Encoding settings for every data variable. Will update the default settings.
- **engine** (*str (optional)*) – The engine used for writing the file. Defaults to "netcdf4".
- **encoding_defaults** – Encoding settings shared by all data variables. This will update the default encoding settings, which are `dict(dtype="float32", zlib=False, complevel=4)`.

Helper Functions

These are the functions exposed by the module.

`climada_petals.hazard.rf_glofas.rf_glofas.dask_client` (*n_workers*, *threads_per_worker*, *memory_limit*, *args, **kwargs)

Create a context with a `dask.distributed.Client`.

This is a lightweight wrapper and intended to expose only the most important parameters to end users.

Parameters

- **n_workers** (*int*) – Number of parallel processes to launch.
- **threads_per_worker** (*int*) – Compute threads launched by each worker.
- **memory_limit** (*str*) – Memory limit for each process. Example: 4 GB can be expressed as 4000M or 4G.
- **args, kwargs** – Additional (keyword) arguments passed to the `dask.distributed.Client` constructor.

Example

```
>>> with dask_client(n_workers=2, threads_per_worker=2, memory_limit="4G"):
...     xr.open_dataset("data.nc", chunks="auto").median()
```

`climada_petals.hazard.rf_glofas.rf_glofas.hazard_series_from_dataset` (*data: Dataset*, *intensity: str*, *event_dim: str*) → Series | Hazard

Create a series of Hazard objects from a multi-dimensional dataset

The input flood data is usually multi-dimensional. For example, you might have downloaded ensemble data over an extended period of time. Therefore, this function returns a `pandas.Series`. Each entry of the series is a `Hazard` object whose events have the same coordinates in this multi-dimensional space except the one given by `event_dim`. For example, if your data space has the dimensions `time`, `lead_time` and `number`, and you choose `event_dim="number"`, then the index of the series will be a `MultiIndex` from `time` and `lead_time`, and a single hazard object will contain all events along the `number` axis for a given `MultiIndex`.

Parameters

- **data** (*xarray.Dataset*) – Data to load a hazard series from.
- **intensity** (*str*) – Name of the dataset variable to read as hazard intensity.
- **event_dim** (*str*) – Name of the dimension to be used as event dimension in the hazards. All other dimension names except the dimensions for longitude and latitude will make up the hierarchy of the `MultiIndex` of the resulting series.

Returns

Series of `Hazard` objects with events along `event_dim` and with a `MultiIndex` of the remaining dimensions.

Return type

`pandas.Series`

Tip

This function must transpose the underlying data in the dataset to conveniently build `Hazard` objects. To ensure that this is an efficient operation, avoid plugging the return value of `compute()` directly into this function,

especially for **large data**. Instead, save the data first using `save_file()`, then re-open the data with xarray and call this function on it.

Examples

Execute the default pipeline and retrieve the Hazard series

```
>>> import xarray as xr
>>> dset = xr.open_dataset("flood.nc")
>>> sorted(list(dset.dims.keys()))
["date", "latitude", "longitude", "number", "sample"]
```

```
>>> from climada_petals.hazard.rf_glofas import hazard_series_from_dataset
>>> with xr.open_dataset("flood.nc") as dset:
>>>     hazard_series_from_dataset(dset, "flood_depth_flopros", "number")
date      sample
2022-08-10  0      <climada.hazard.base.Hazard ...
              1      <climada.hazard.base.Hazard ...
2022-08-11  0      <climada.hazard.base.Hazard ...
              1      <climada.hazard.base.Hazard ...
Length: 4, dtype: object
```

```
climada_petals.hazard.rf_glofas.setup.download_flopros_database (output_dir: str | Path =
                                                                PosixPath('/home/docs/climada/data/river-
                                                                flood-computation'))
```

Download the FLOPROS database and place it into the output directory.

Download the supplementary material of P. Scussolini et al.: “FLOPROS: an evolving global database of flood protection standards”, extract the zipfile, and retrieve the shapefile within. Discard the temporary data afterwards.

```
climada_petals.hazard.rf_glofas.setup.download_flood_hazard_maps (output_dir: str | Path)
```

Download the JRC flood hazard maps and unzip them

This stores the downloaded zip files as temporary files which are discarded after unzipping.

```
climada_petals.hazard.rf_glofas.setup.setup_flood_hazard_maps (flood_maps_dir: Path,
                                                                output_dir=PosixPath('/home/docs/climada/data/river-
                                                                flood-computation'))
```

Download the flood hazard maps and merge them into a single NetCDF file

Maps will be downloaded into `flood_maps_dir` if it does not exist. Then, the single maps are re-written as NetCDF files, if these do not exist. Finally, all maps are merged into a single dataset and written to the `output_dir`. Because NetCDF files are more flexibly read and written, this procedure is more efficient than directly merging the GeoTIFF files into a single dataset.

Parameters

- **flood_maps_dir** (*Path*) – Storage directory of the flood maps as GeoTIFF files. Will be created if it does not exist, in which case the files are automatically downloaded.
- **output_dir** (*Path*) – Directory to store the flood maps dataset.

```
climada_petals.hazard.rf_glofas.setup.setup_gumbel_fit (output_dir=PosixPath('/home/docs/climada/data/river-
                                                                flood-computation'), num_downloads: int =
                                                                1, parallel: bool = False)
```

Download historical discharge data and compute the Gumbel distribution fits.

Data is downloaded from the Copernicus Climate Data Store (CDS).

Parameters

- **output_dir** – The directory to place the resulting file
- **num_downloads** (*int*) – Number of parallel downloads from the CDS. Defaults to 1.
- **parallel** (*bool*) – Whether to preprocess data in parallel. Defaults to `False`.

```
climada_petals.hazard.rf_glofas.setup.download_gumbel_fit (output_dir=PosixPath('/home/docs/climada/data/river-flood-computation'))
```

Download the pre-computed Gumbel parameters from the ETH research collection.

Download dataset of <https://doi.org/10.3929/ethz-b-000641667>

```
climada_petals.hazard.rf_glofas.setup.setup_all (output_dir: str | Path =
PosixPath('/home/docs/climada/data/river-flood-computation'))
```

Set up the data for river flood computations.

This performs two tasks:

1. Downloading the JRC river flood hazard maps and merging them into a single NetCDF dataset.
2. Downloading the FLOPROS flood protection database.
3. Downloading the Gumbel distribution parameters fitted to GloFAS river discharge reanalysis data from 1979 to 2015.

Parameters

output_dir (*Path or str, optional*) – The directory to store the datasets into.

CDS Glofas Downloader

```
climada_petals.hazard.rf_glofas.cds_glofas_downloader.glofas_request (product: str, date_from:
str, date_to: str | None,
output_dir: Path | str,
num_proc: int = 1,
use_cache: bool = True,
request_kw:
Mapping[str, str] | None
= None, client_kw:
Mapping[str, Any] |
None = None) →
List[Path]
```

Request download of GloFAS data products from the Copernicus Data Store (CDS)

Uses the Copernicus Data Store API (`cdsapi`) Python module. The interpretation of the `date` parameters and the grouping of the downloaded data depends on the type of `product` requested.

Available products:

- **historical**: Historical reanalysis discharge data. `date_from` and `date_to` are interpreted as integer years. Data for each year is placed into a single file.
- **forecast**: Forecast discharge data. `date_from` and `date_to` are interpreted as ISO date format strings. Data for each day is placed into a single file.

Notes

Downloading data from the CDS requires authentication via a user key which is granted to each user upon registration. Do the following **before calling this function**:

- Create an account at the Copernicus Data Store website: <https://cds.climate.copernicus.eu/>
- Follow the instructions to install the CDS API key: <https://cds.climate.copernicus.eu/api-how-to#install-the-cds-api-key>

Parameters

- **product** (*str*) – The identifier for the CMS product to download. See below for available options.
- **date_from** (*str*) – First date to download data for. Interpretation varies based on `product`.
- **date_to** (*str or None*) – Last date to download data for. Interpretation varies based on `product`. If `None`, or the same date as `date_from`, only download data for `date_from`
- **output_dir** (*Path*) – Output directory for the downloaded data
- **num_proc** (*int*) – Number of processes used for parallel requests
- **use_cache** (*bool (optional)*) – Skip downloading if the target file exists and the accompanying request file contains the same request
- **request_kw** (*dict(str: str)*) – Dictionary to update the default request for the given product
- **client_kw** (*dict (optional)*) – Dictionary with keyword arguments for the `cdsapi.Client` used for downloading

Returns

Paths of the downloaded files

Return type

list of `Path`

The default configuration for each product will be updated with the `request_kw` from `climada_petals.hazard.rf_glofas.cds_glofas_downloader.glofas_request()`:

```
climada_petals.hazard.rf_glofas.cds_glofas_downloader.DEFAULT_REQUESTS = {'forecast':
{'day': '01', 'format': 'grib', 'hydrological_model': 'lisflood', 'leadtime_hour':
['24', '48', '72', '96', '120', '144', '168', '192', '216', '240', '264', '288', '312',
'336', '360', '384', '408', '432', '456', '480', '504', '528', '552', '576', '600',
'624', '648', '672', '696', '720'], 'month': '08', 'product_type':
'ensemble_perturbed_forecasts', 'system_version': 'version_3_1', 'variable':
'river_discharge_in_the_last_24_hours', 'year': '2022'}, 'historical': {'format':
'grib', 'hday': ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12',
'13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26',
'27', '28', '29', '30', '31'], 'hmonth': ['january', 'february', 'march', 'april',
'may', 'june', 'july', 'august', 'september', 'october', 'november', 'december'],
'hydrological_model': 'lisflood', 'hyear': '1979', 'product_type': 'consolidated',
'system_version': 'version_3_1', 'variable': 'river_discharge_in_the_last_24_hours'}}
```

Default request keyword arguments to be updated by the user requests

2.3.3 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

2.3.4 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 be downloaded 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 resolution (e.g. at the equator, $1^\circ \sim 111$ 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 loaded gdf
- **res** (*float*) – resolution in units of the input_gdf crs of the final grid cells which are created. With 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 probabilities 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 occurrence 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.

2.3.5 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 resolution 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 Sector (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, $0.0 < \text{percentile} < 100.0$
- **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 simulation 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 time 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 extent below min_number_cells

Parameters

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

Return type

Hazard

2.3.6 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. 'ewemb' (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 'hist-soc'
- **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 simulation 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

2.3.7 climada_petals.hazard.river_flood module

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

Bases: Hazard

It contains flood events retrieved by: - PIK/ISIMIP:

<https://files.isimip.org/cama-flood/results/>

- **Aqueduct project:**

<https://www.wri.org/data/aqueduct-floods-hazard-maps>

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_cent

flooded area in every centroid for every event

Type

2d array(n_years x n_centroids)

fla_ev_cent

flooded area in every centroid for every event

Type

2d array(n_events x n_centroids)

__init__ (*args, **kwargs)

Empty constructor

classmethod from_aqueduct_tif (*scenario: str, target_year: str, gcm: str, return_periods: int | Iterable[int] = None, countries: str | Iterable[str] | None = None, boundaries: Iterable[float] = None, dwd_dir: str = PosixPath('/home/docs/clinada/data/RiverFlood/Aqueduct')*)

It downloads and extracts riverine flood events pulled by the Aqueduct project

scenario

[str] scenario to use. Possible values are historical, 45 and 85. The latter two clearly refer to RCP4.5 and RCP8.5.

target_year

[str] future target year. Possible values are 1980, 2030, 2050 and 2080.

gcm

[str]

the Global Circulation Model to use. Possible values are

WATCH, NorESM1-M, GFDL-ESM2M, HadGEM2-ES, IPSL-CM5A-LR and MIROC-ESM-CHEM.

WATCH is used only under historic, all others are used in the two RCPs.

return_periods

[int or list of int] events' return periods. Possible values are 2, 5, 10, 25, 50, 100, 250, 500, 1000. By default, all are considered.

countries

[str or list of str] countries ISO3 codes

boundaries

[tuple of floats]

geographical boundaries in the order:

minimum longitude, minimum latitude, maximum longitude, maximum latitude

classmethod `from_isimip_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_isimip_nc (**args, **kwargs*)

This function is deprecated, use RiverFlood.from_isimip_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 threshold value is excluded. (This function is only needed for very specific applications)

Raises

NameError –

set_flooded_area (*save_cent*=False)

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

Raises

MemoryError –

set_flood_volume (*save_cent*=False)

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

Raises

MemoryError –

2.3.8 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, 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: centroids at 360 arc-seconds resolution within tracks' bounds.
- **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***2.3.9 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_hdf5(HAZ_DEMO_H5)
```

```
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.

2.3.10 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*)

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

2.3.11 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_proba: 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 clustering 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 clustering 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 determine `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

2.4 climada_petals.util package

2.4.1 climada_petals.util.config module

2.4.2 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

HAZARD TUTORIALS

3.1 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.

3.1.1 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.

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

```
from climada.hazard import TropCyclone
hazard = TropCyclone.from_hdf5(EMULATOR_DATA_DIR.joinpath("hazard_360as_miroc_rcp26.
↳hdf5"))
```

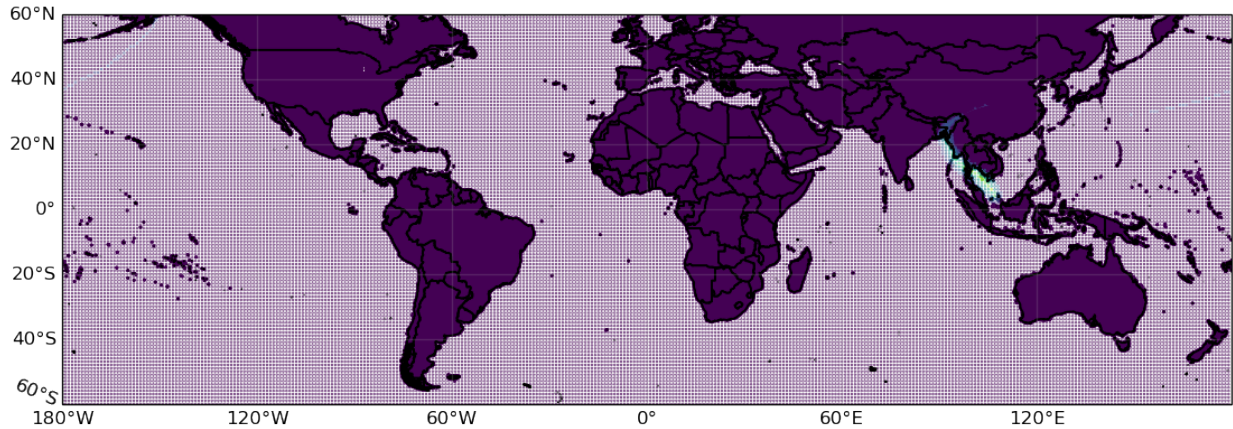
```
2021-03-24 17:46:31,484 - climada.hazard.base - INFO - Reading /home/tovogt/.climada/
↳demo/data/emulator/hazard_360as_miroc_rcp26.hdf5
```

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:

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

```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x7f74c8bb0a60>
```



3.1.2 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.

```
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.
```

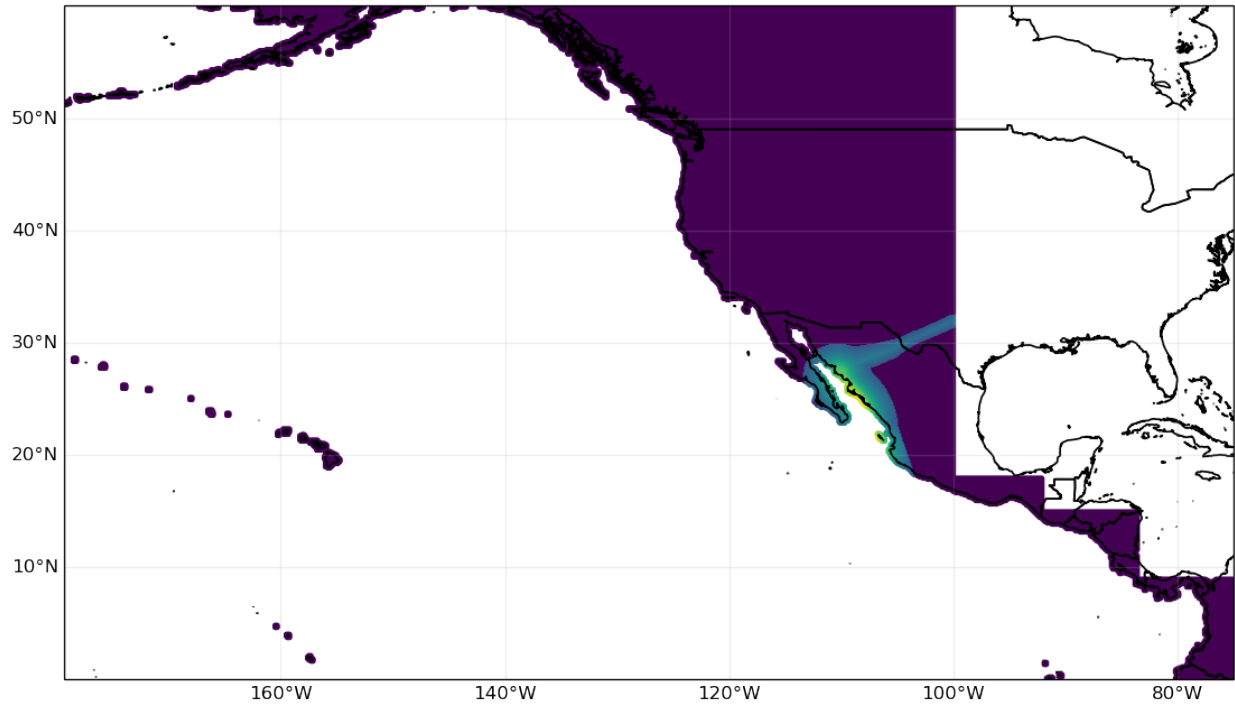
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f74985581f0>
```



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

```
import shapely
hazard.centroids.gdf['region_id'] = shapely.vectorized.contains(
    region.shape, hazard.centroids.lon, hazard.centroids.lat) & hazard.centroids.gdf.
↳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);
```

```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x7f7498568ee0>
```



3.1.3 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):

```
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 ...
```

	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

(continues on next page)

(continued from previous page)

```
7399 45284 28484 2100 10 9 17.8 -101.5 59.966329
7400 45298 28498 2100 10 13 24.2 -107.5 31.401268
```

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

3.1.4 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*:

```
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 :

```
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]
```

3.1.5 Bias-corrected statistics

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

```
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):

```
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:

```
observed = TropCyclone.from_hdf5(EMULATOR_DATA_DIR.joinpath("hazard_360as_ibtracs_
↳1950-2019.hdf5"))
observed.centroids.gdf['region_id'] = shapely.vectorized.contains(
    region.shape, observed.centroids.lon, observed.centroids.lat) & observed.
↳centroids.gdf.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.

```
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.

3.1.6 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.

```
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):

```
em.stats
```

	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		NaN
147	NaN	NaN	NaN	NaN		NaN
148	NaN	NaN	NaN	NaN		NaN
149	NaN	NaN	NaN	NaN		NaN
150	NaN	NaN	NaN	NaN		NaN

```
[151 rows x 9 columns]
```

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

```
# 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.

```
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

3.1.7 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:

```
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`:

```
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:

```
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:

```
em.stats_pred
```

```

      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

```

(continues on next page)

(continued from previous page)

```
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.

```
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

3.1.8 Create sample Hazard object from draws

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

```
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):

```
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)]
```

```
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),
```

(continues on next page)

(continued from previous page)

```
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:

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

3.1.9 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 atmospheric (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 Moni-](#)

toring website, hosted by Columbia University. For the air pressure input, we extracted the CMIP5 monthly mean atmospheric (Amon) `psl` field of MIROC5 RCP 2.6 simulations covering the years 1861-2299.

3.2 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()`

3.2.1 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 occurrence 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.

```
# Amending the Landslide catalog by loading and re-saving (only necessary first time!)
import geopandas as gpd
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'
ls_gdf_all = gpd.read_file(PATH_COOLR)
ls_gdf_all.to_file(PATH_COOLR)
```

Now we can start the actual task..

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, $1^\circ \sim 111$ 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.

```
from climada_petals.hazard.landslide import Landslide

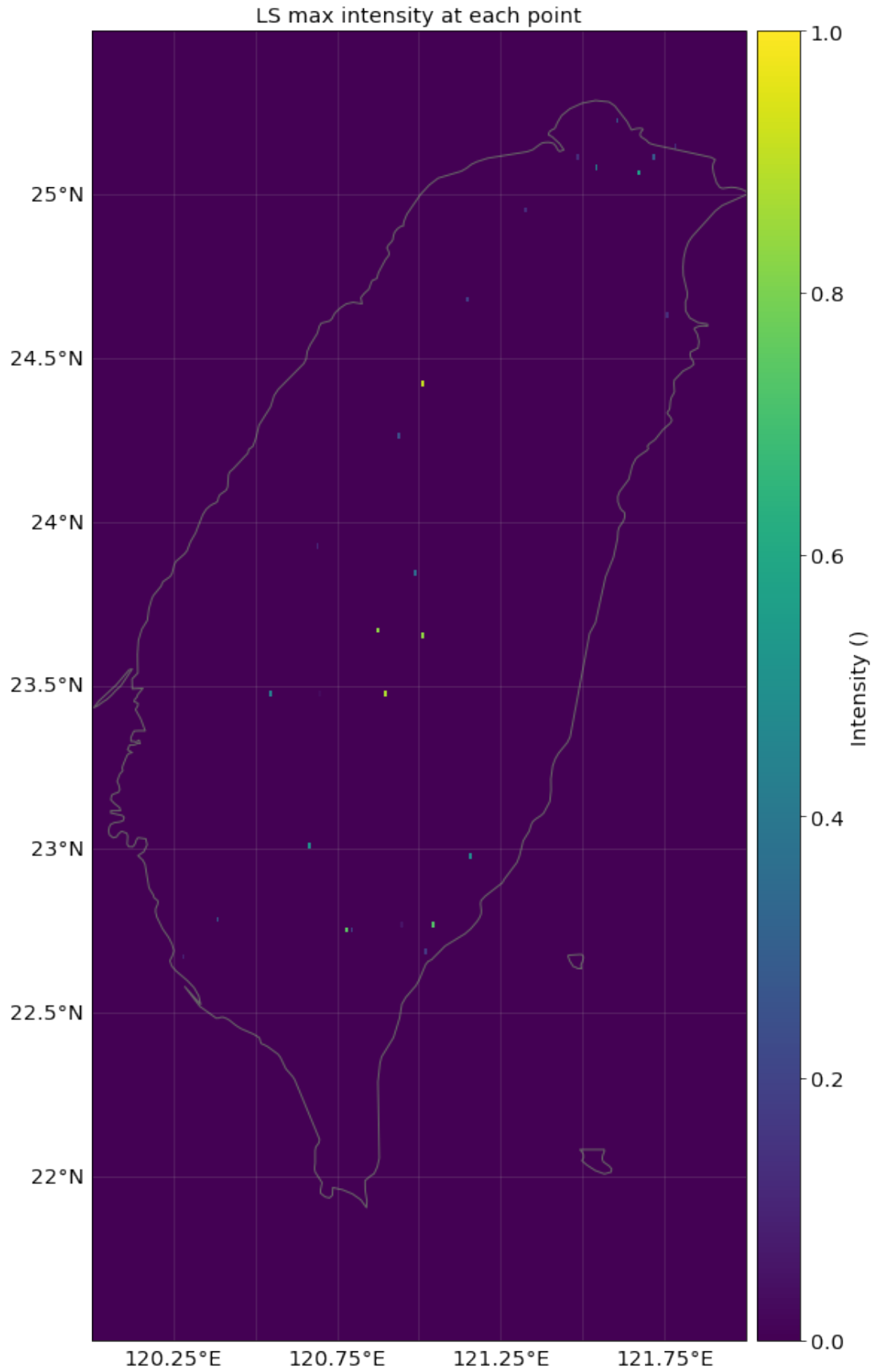
bbox_taiwan = (120.0, 21.5, 122.0, 25.5) # bbox as (minx, miny, maxx, maxy)
# Example for Taiwan
haz_ls_Taiwan_hist = Landslide.from_hist(bbox=bbox_taiwan, input_gdf=PATH_COOLR,
    ↪res=0.004)
# Visual inspection of the hazard
haz_ls_Taiwan_hist.plot_intensity(0);
```

```
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_
    ↪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)
```

```
<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:
```



3.2.2 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.

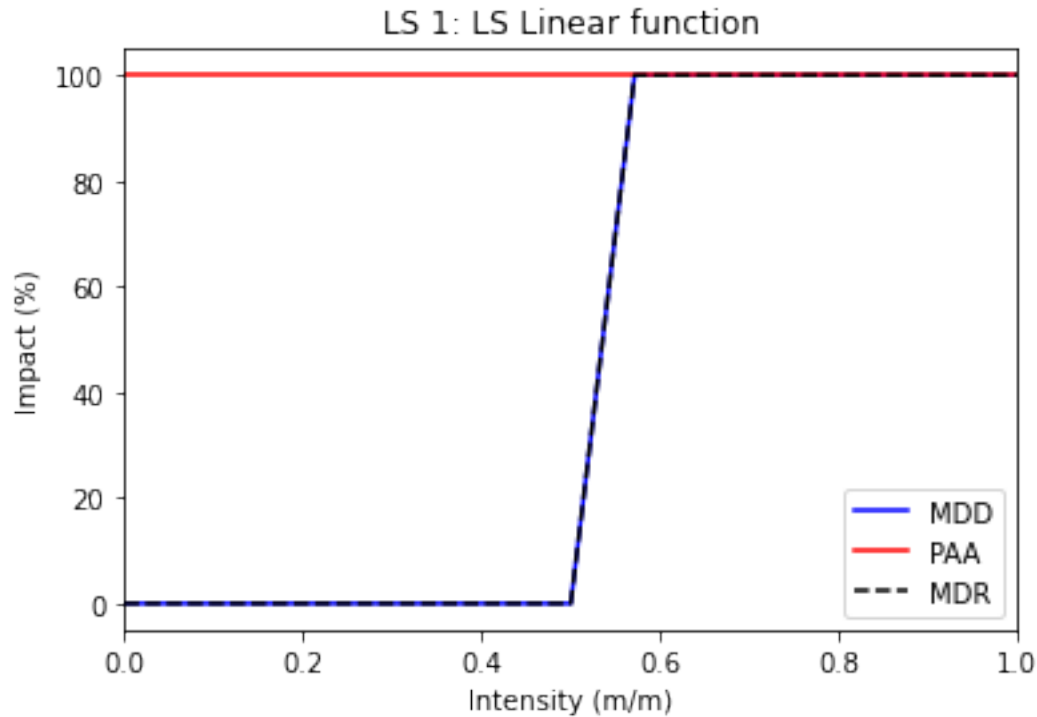
Impact functions relate the hazard intensity to a percentage of damage in the exposure. For a detailed description on impact functions, check out the respective tutorial.

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.

```
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
```

```
# 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.

```
# 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
```

(continues on next page)

(continued from previous page)

```

↪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(

```

```

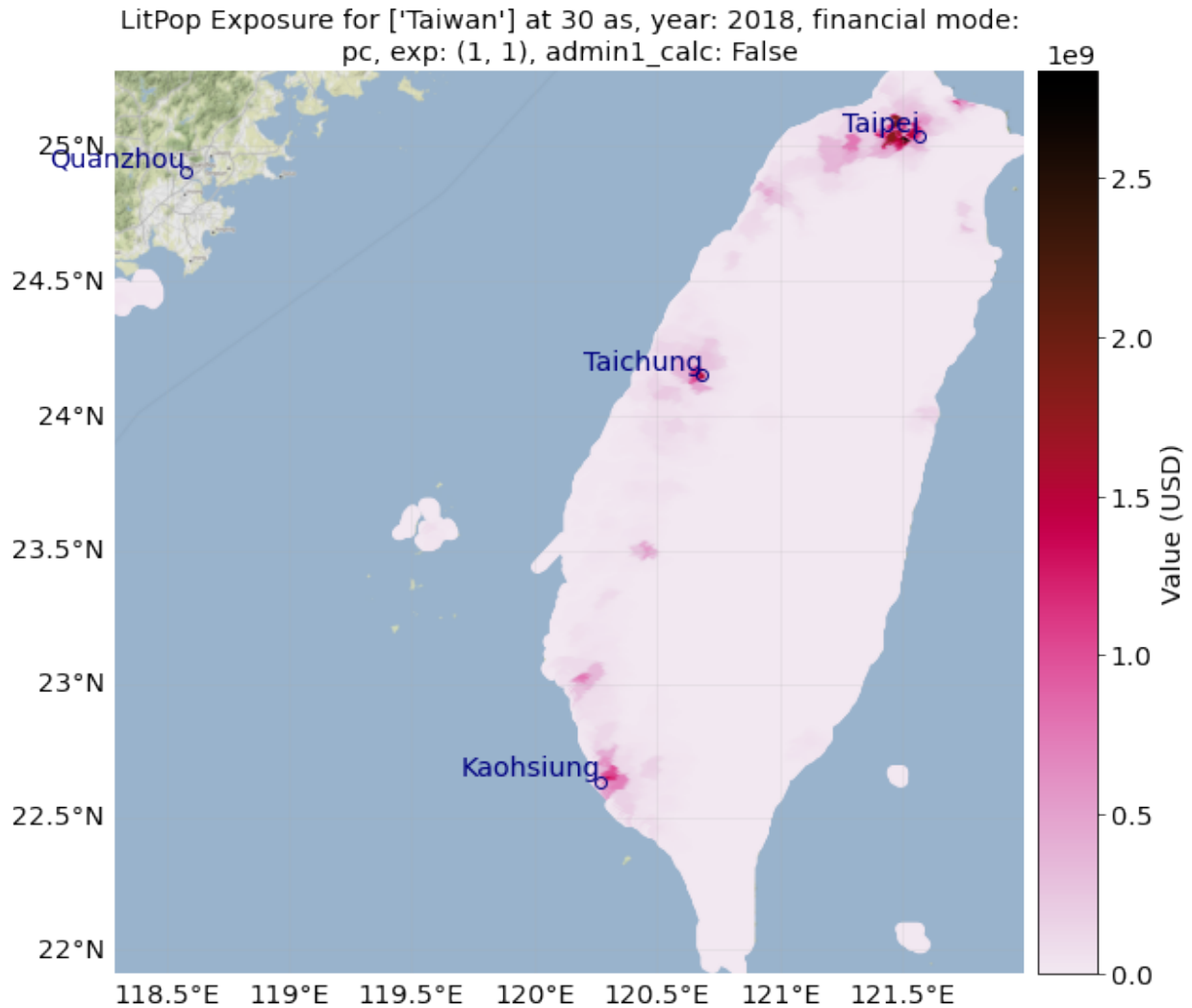
2022-04-11 15:11:59,465 - climada.entity.exposures.base - INFO - Setting latitude and
↪longitude attributes.

```

```

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

```



```
# 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 for `impact.at_event` in turn, no frequency correction is made. Apply the probabilistic method (explained below) for such calculations.

```
# 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_
↳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.

```

```

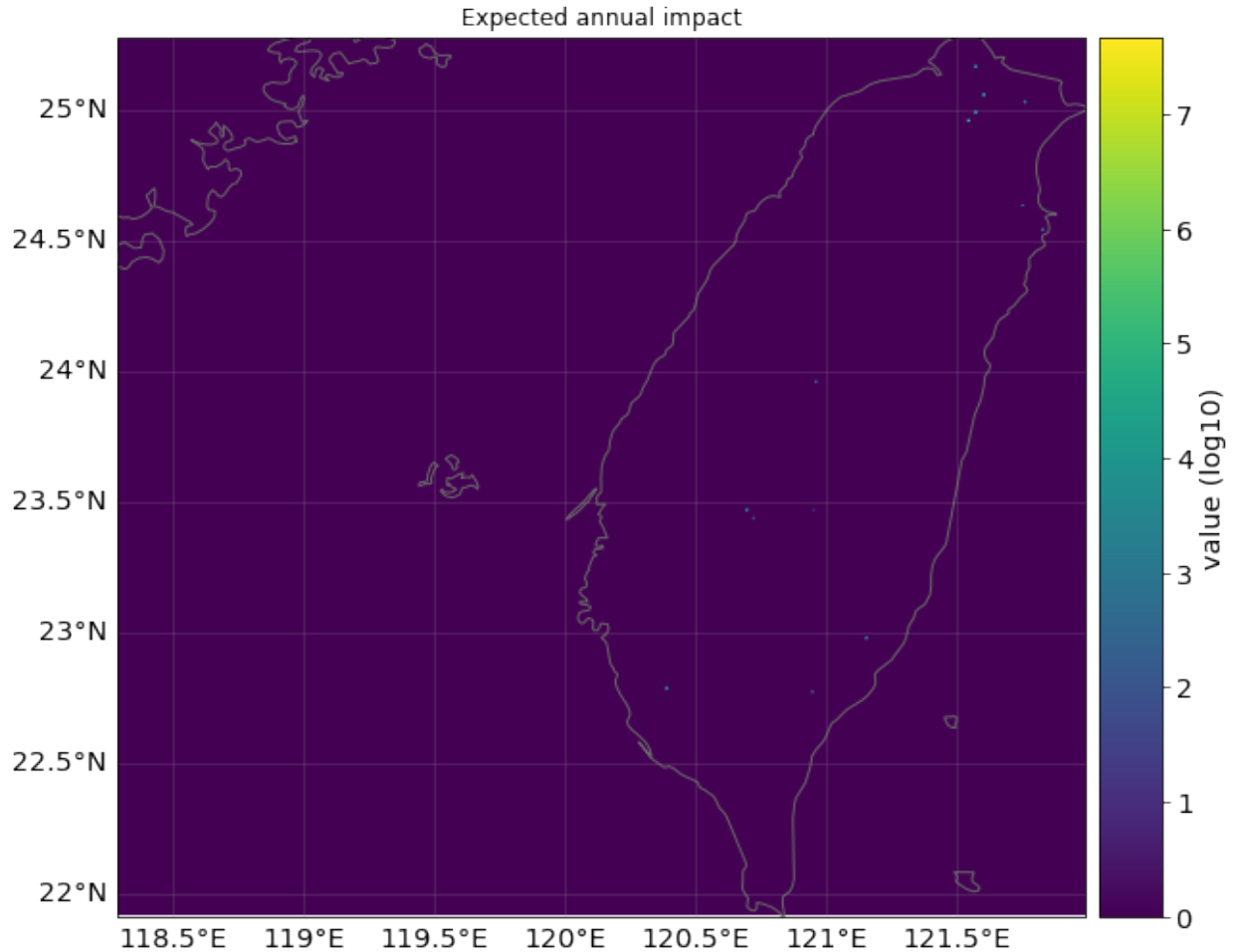
/Users/evelynm/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/
↳ crs.py:1256: UserWarning: You will likely lose important projection information
↳ when converting to a PROJ string from another format. See: https://proj.org/faq.html
↳ #what-is-the-best-format-for-describing-coordinate-reference-systems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

```



3.2.3 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.

```
# 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.

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

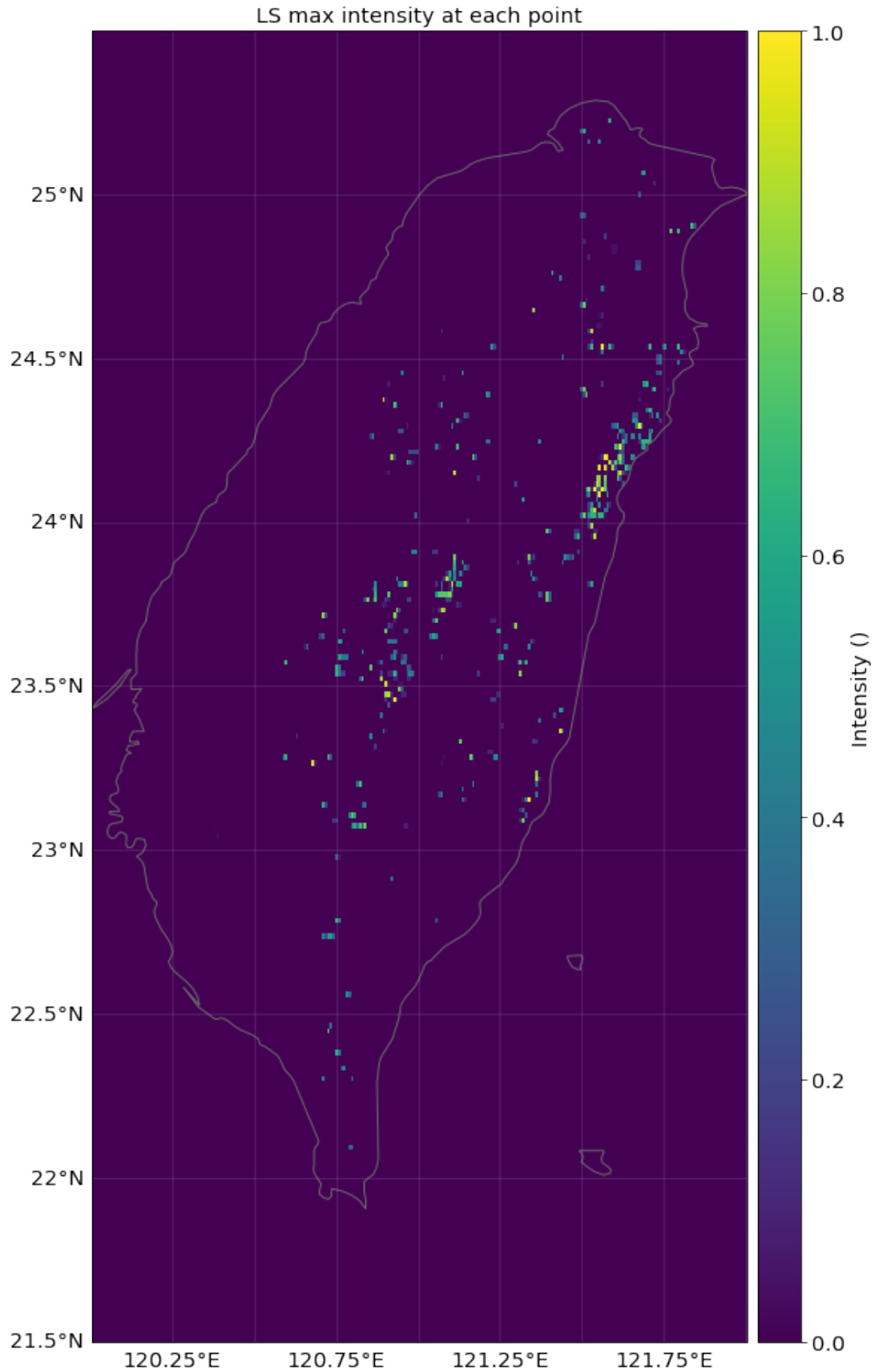
# The check-plots produce intensity (image 1) and fraction (image 2) plots
# a correction factor of 1 million is applied since probs are reportet as x10e6 in_
↳the original data:
haz_ls_taiwan_prob = Landslide.from_prob(bbox=bbox_taiwan, path_sourcefile=PATH_
↳LSPROB,
                                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.centri - 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)
```

```
<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:
```



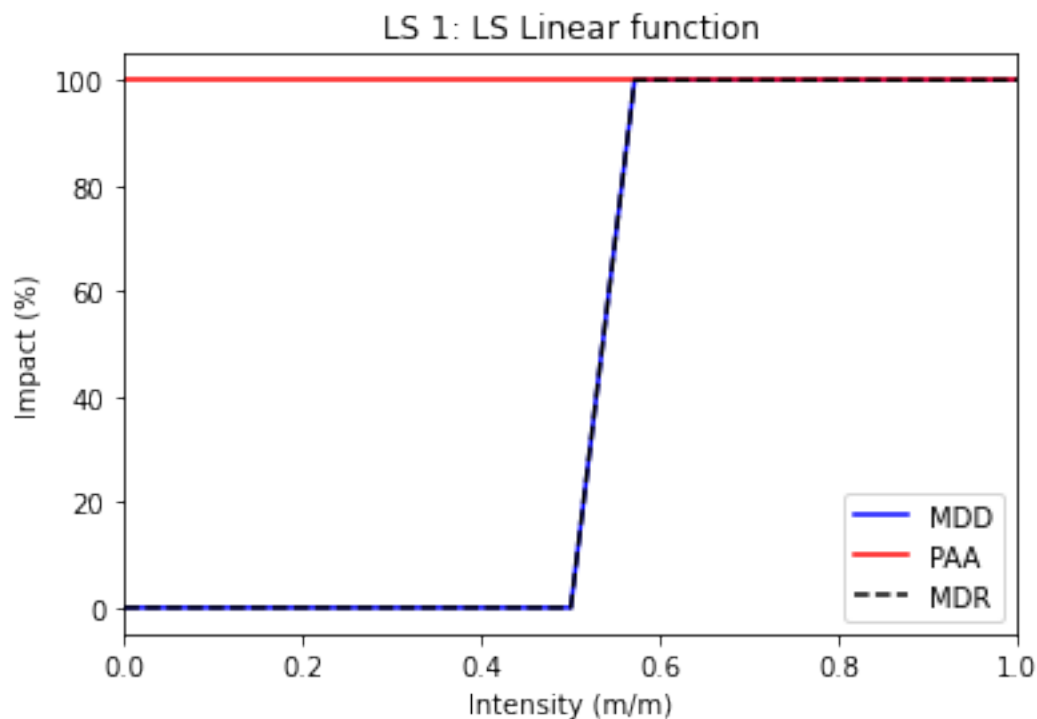
With the hazard set loaded, it is now possible to calculate the expected damage for the simulated period:

```
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.

```
# 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.
```



```
# 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.
```

```
# Set Entity
ent_LS_prob = Entity()
ent_LS_prob.exposures = exp_LS_prob
ent_LS_prob.impact_funcs = impf_set_LS_prob
```

```
# Set impact for probabilistic simulation
imp_LS_Taiwan_prob = Impact()
```

(continues on next page)

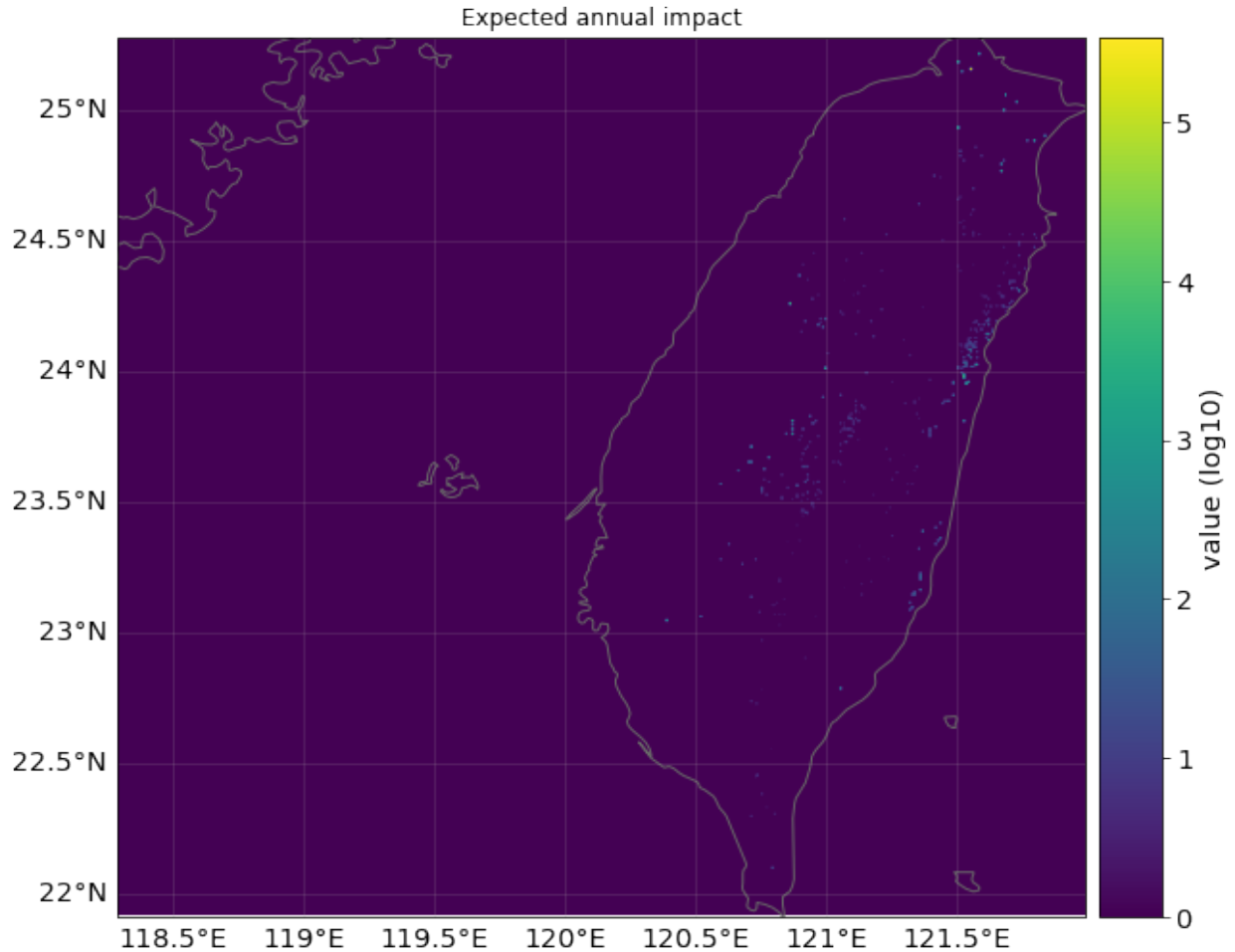
(continued from previous page)

```
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
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:
```

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



```
imp_LS_Taiwan_prob.aai_agg
```

```
1508057.2027470088
```

3.3 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:

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

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

Name	Data Type	Description
fla_ev_centri	2d array(n_events x n_centroids)	flooded area in every centroid for every event
fla_ann_centri	2d array(n_years x n_centroids)	flooded area in every centroid for every year
fv_ann_centri	2d array(n_years x n_centroids)	flood volume in every centroid for every year

3.3.1 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 (fiddph) and flooded fraction (fldfr) 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 further 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.

3.3.2 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:

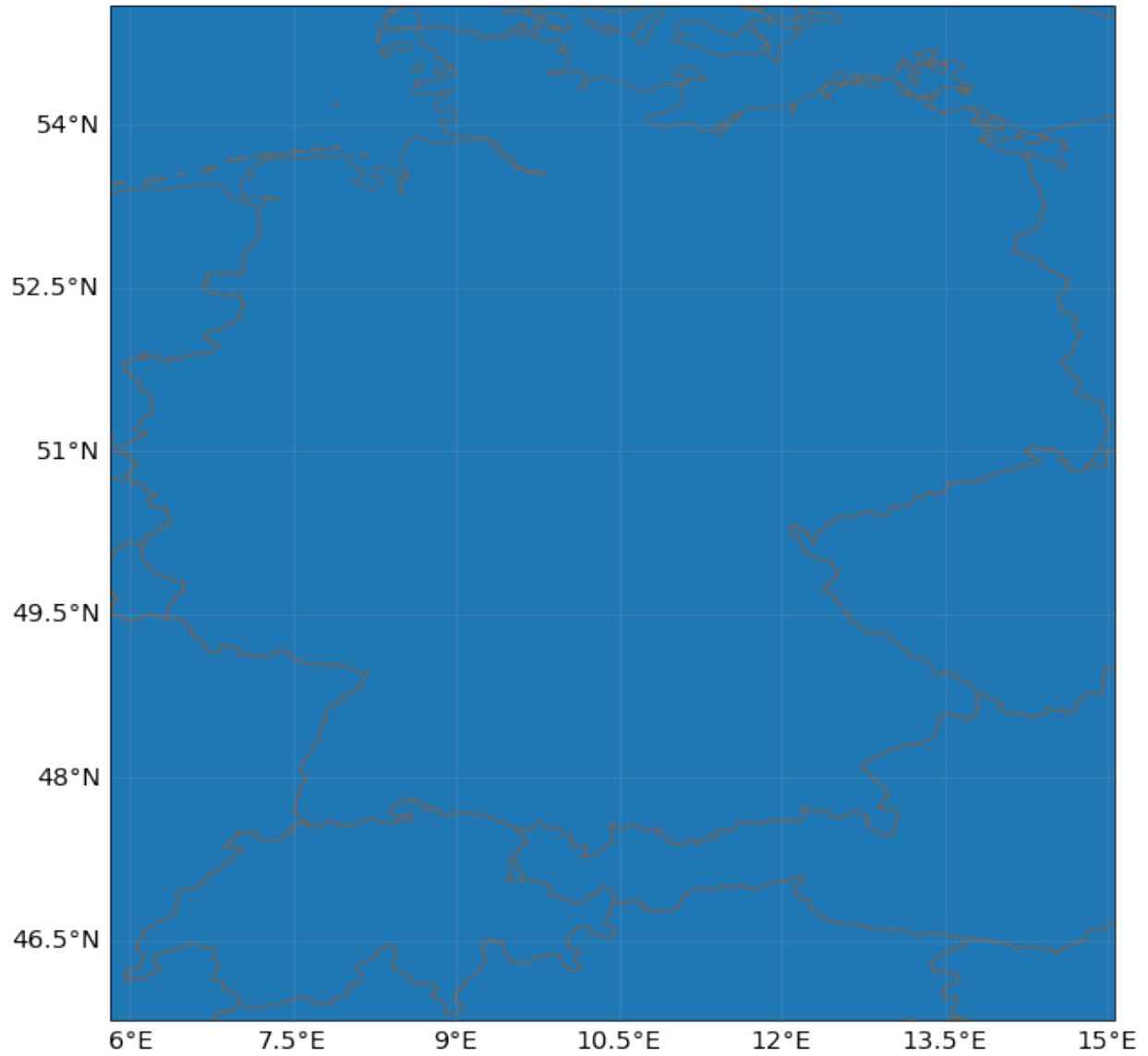
```
import numpy as np
import matplotlib.pyplot as plt
from climada_petals.hazard.river_flood import RiverFlood
from climada.hazard.centroids import Centroids
from climada_petals.util.constants import HAZ_DEMO_FLDDPH, HAZ_DEMO_FLDFRC

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
```

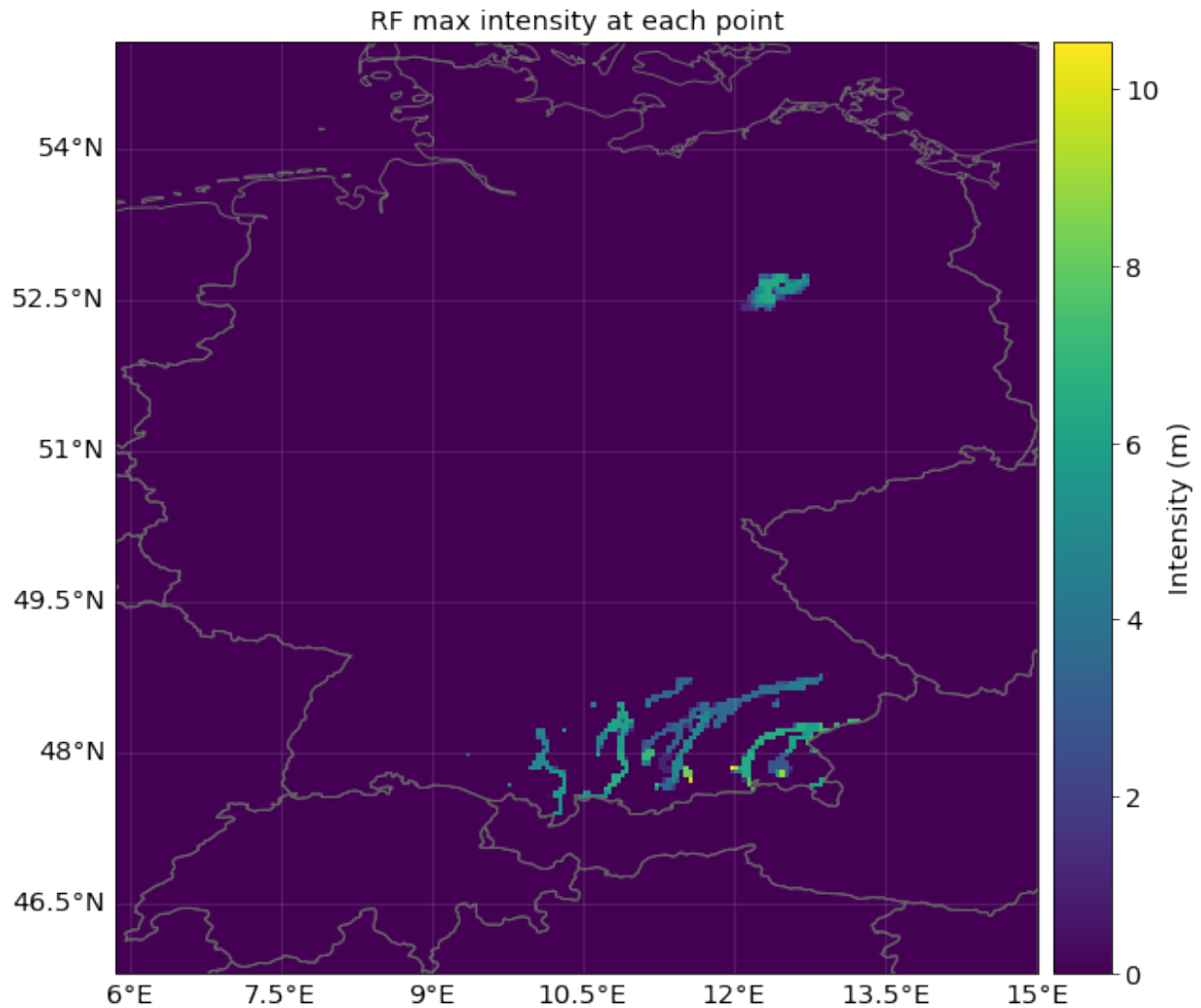
```
['2000']
```

```
# 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: 0.041666666666666662
```

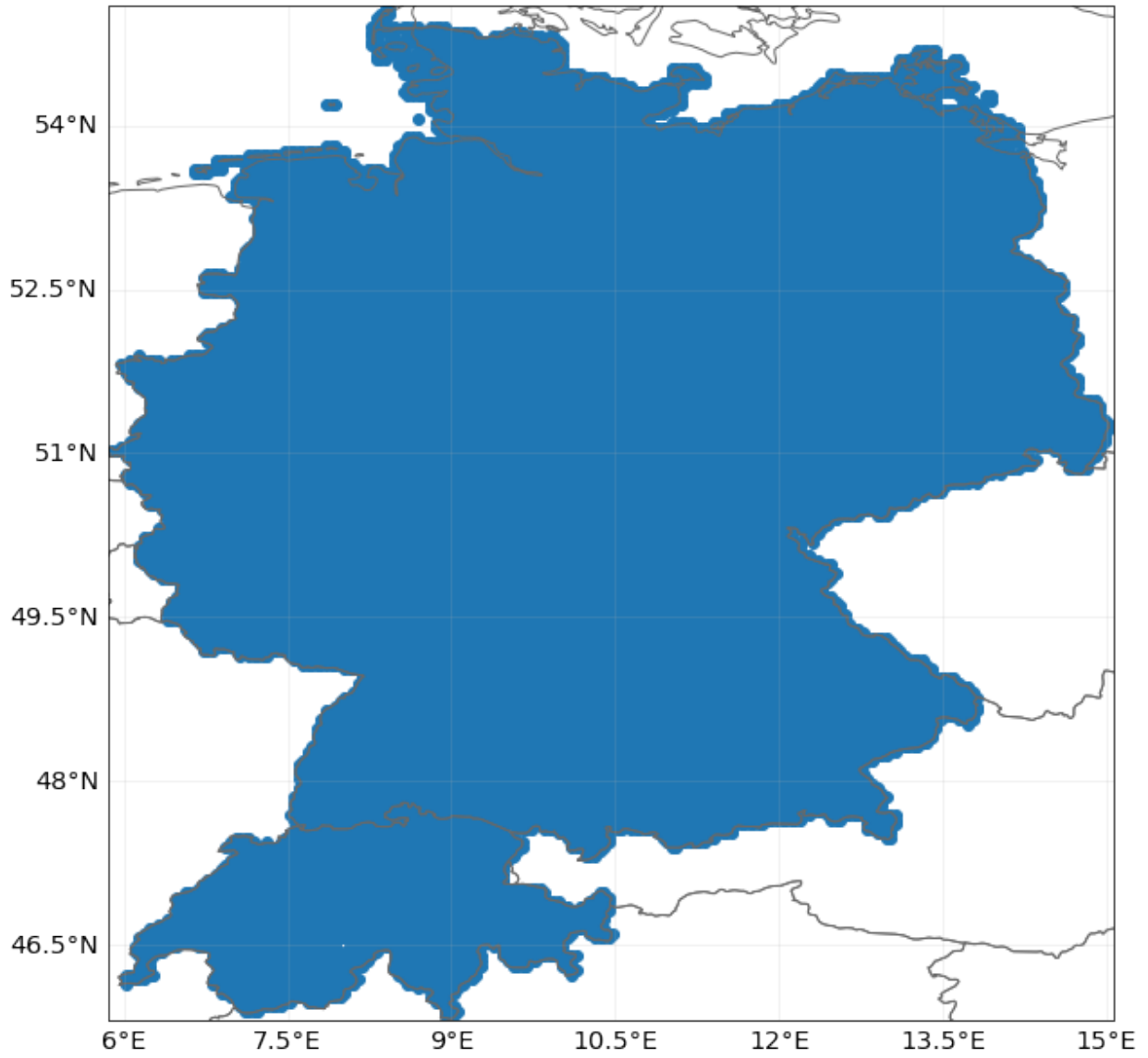


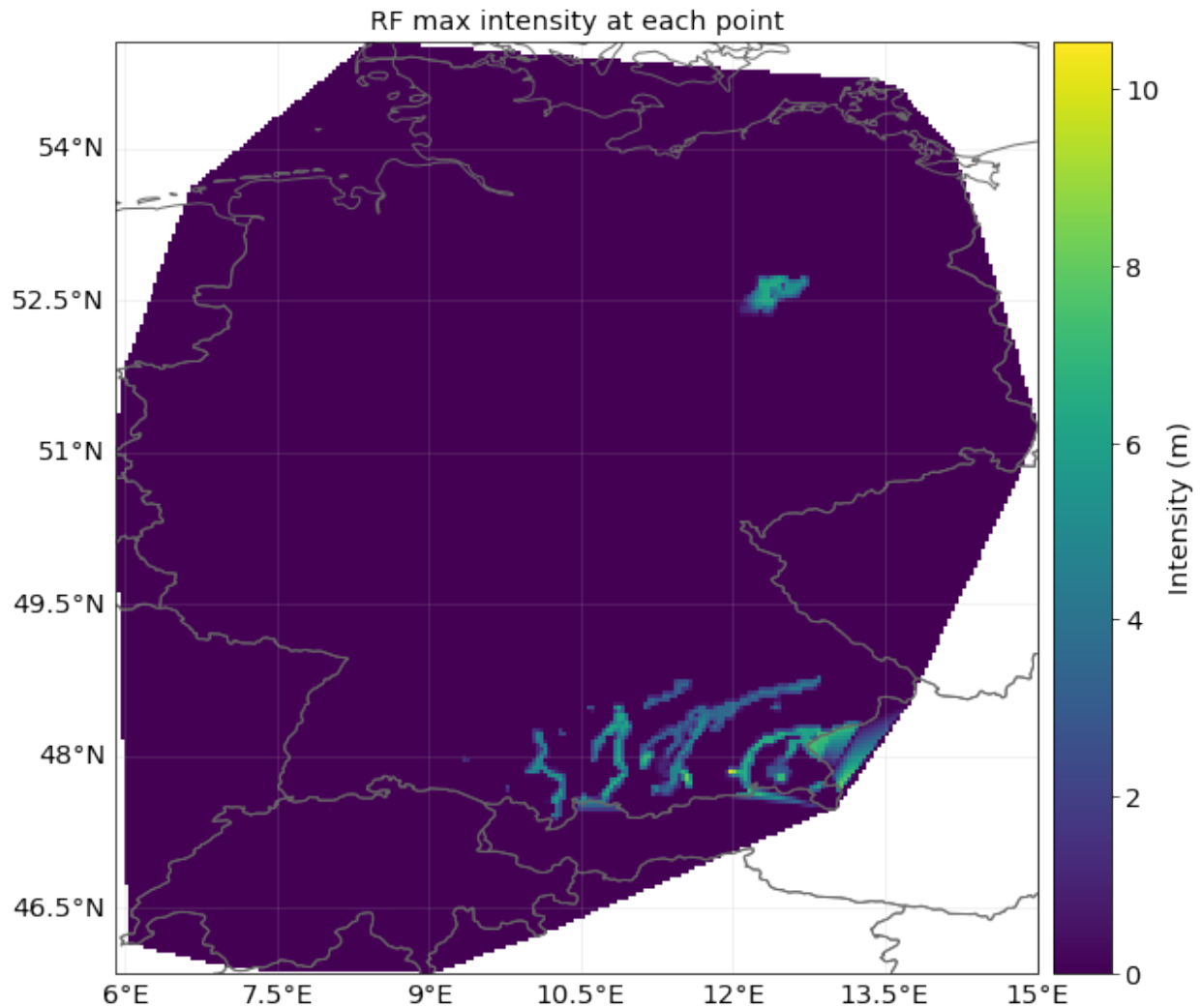
```
# plotting intensity (Flood depth in m)  
rf.plot_intensity(event=0, smooth = False);
```



Setting flood with ISIMIP NatIDGrid:

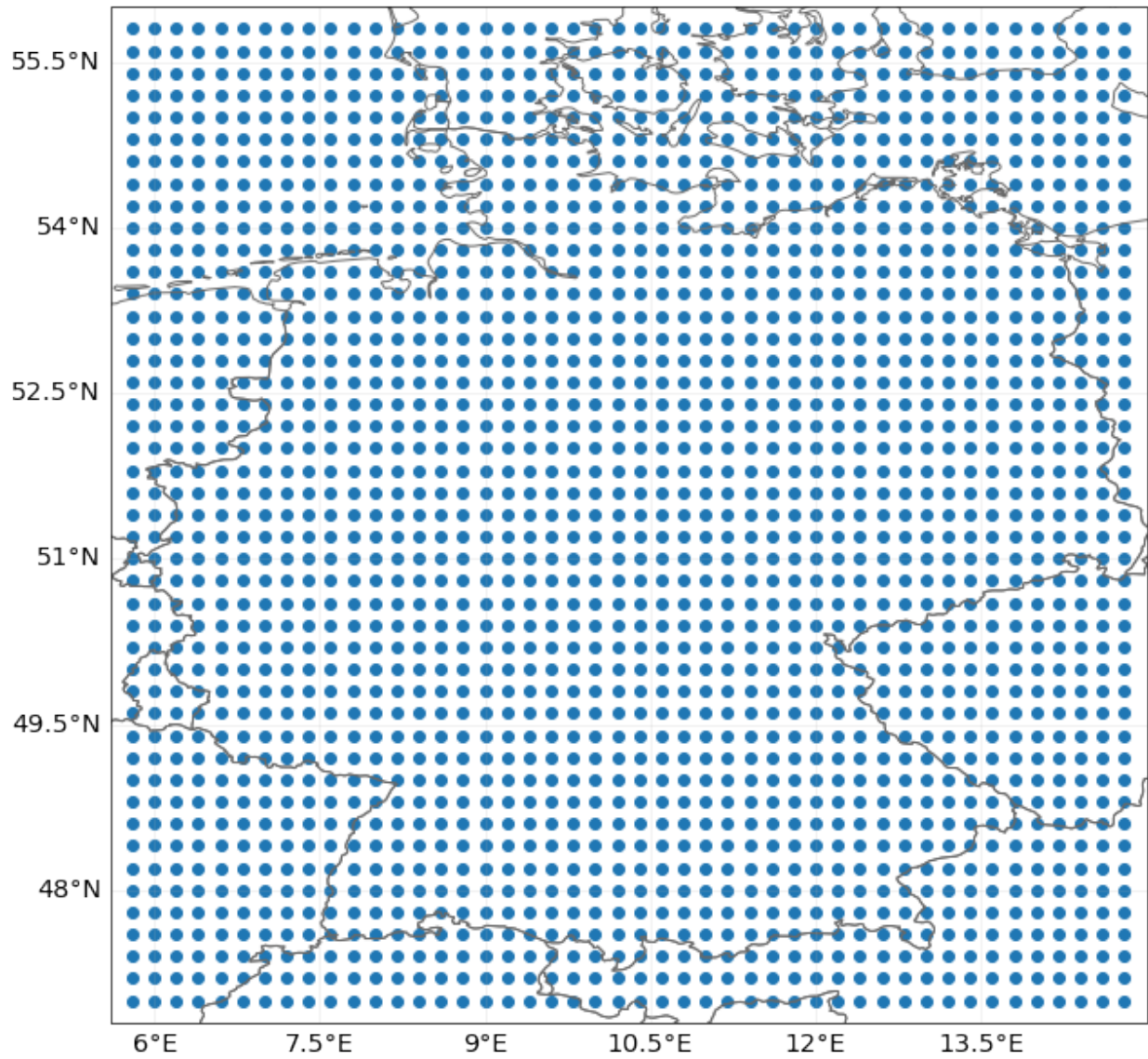
```
# generating RiverFlood hazard from netCDF file, using the ISIMIP NatIDGrid
↳(according to ISIMIP standards) with a resolution of 150as (aprox 5km)
# setting centroids for a region
rf_isi = RiverFlood.from_nc(countries = ['DEU','CHE'], years=years, dph_path=HAZ_DEMO_
↳FLDDPH, frc_path=HAZ_DEMO_FLDFRC,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

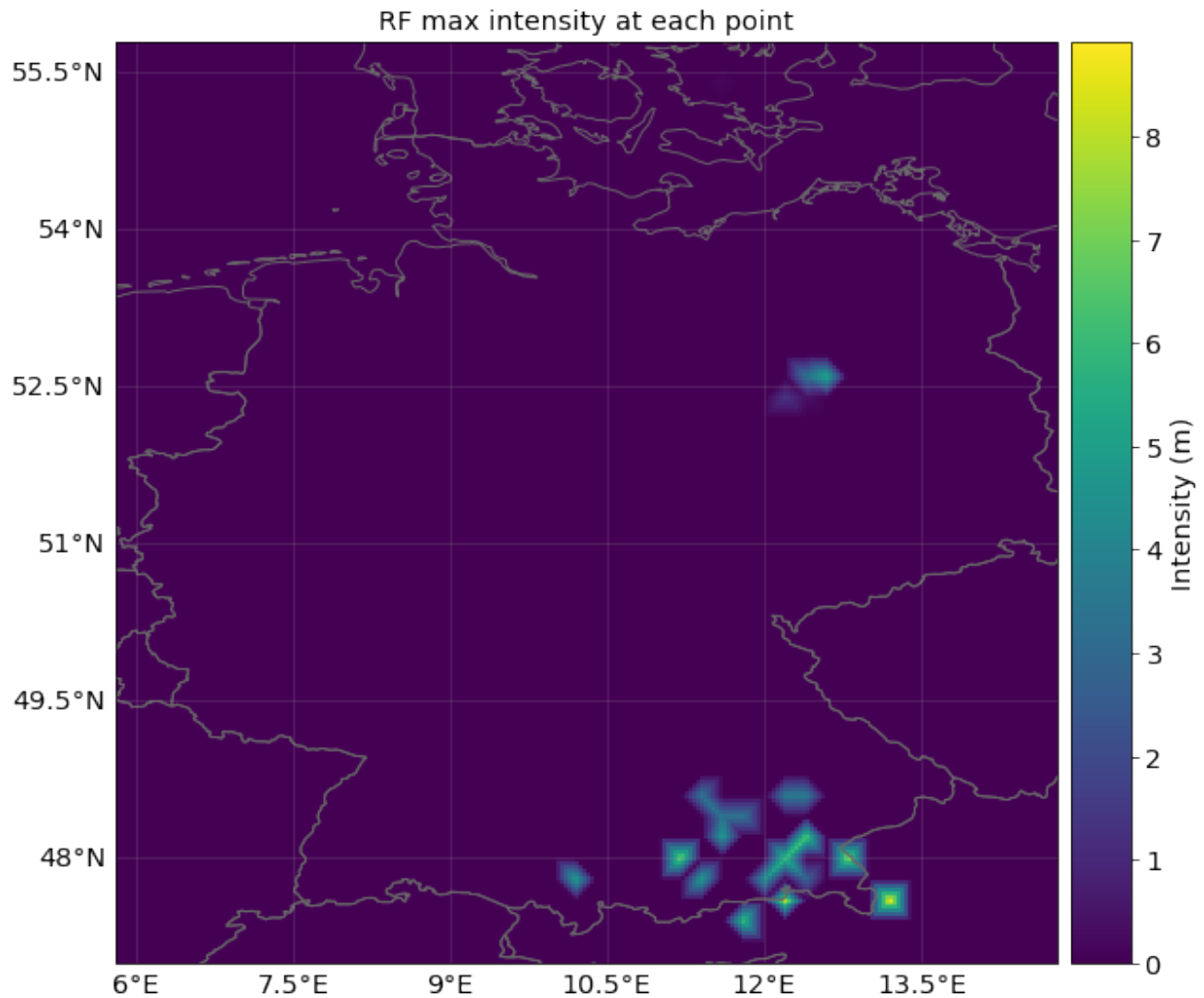




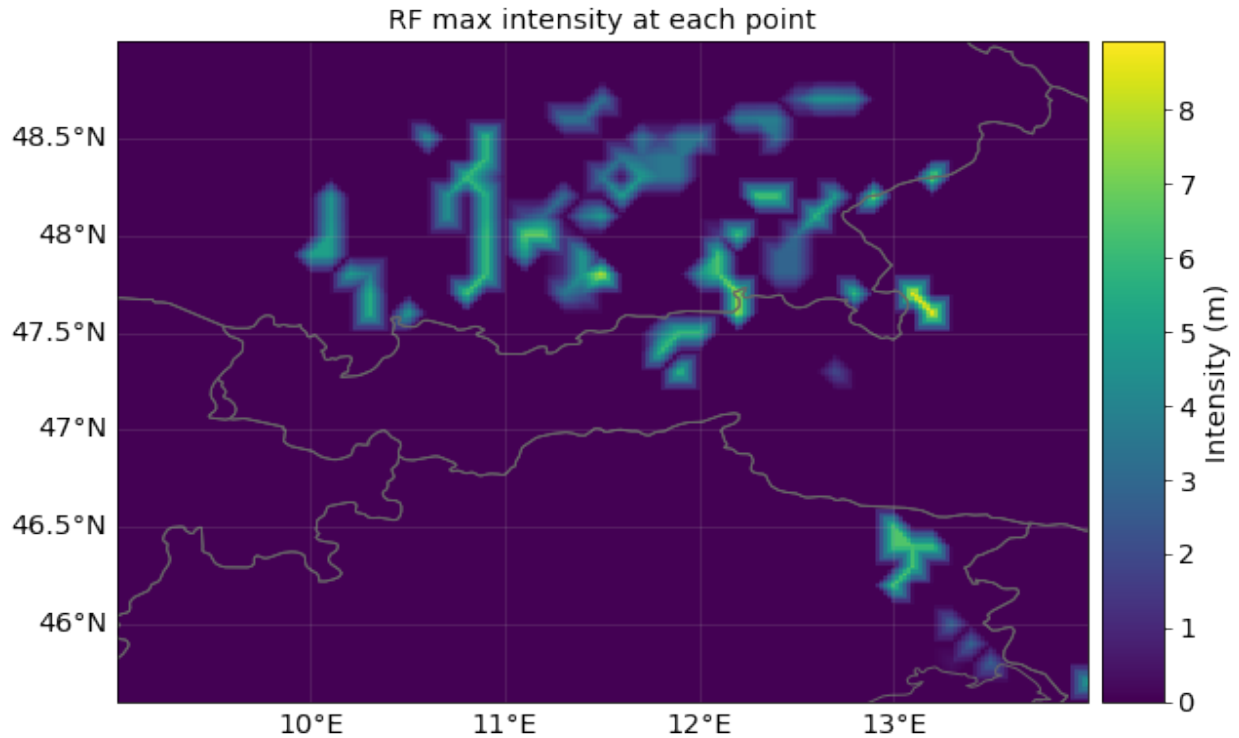
Setting flood with random points as coordinates:

```
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_FLDIFRC,
                             centroids=rand_centroids, ISINatIDGrid=False)
rf_rand.centroids.plot()
rf_rand.plot_intensity(event = 0);
```





```
# setting random points 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);
```



3.3.3 Calculating Flooded Area

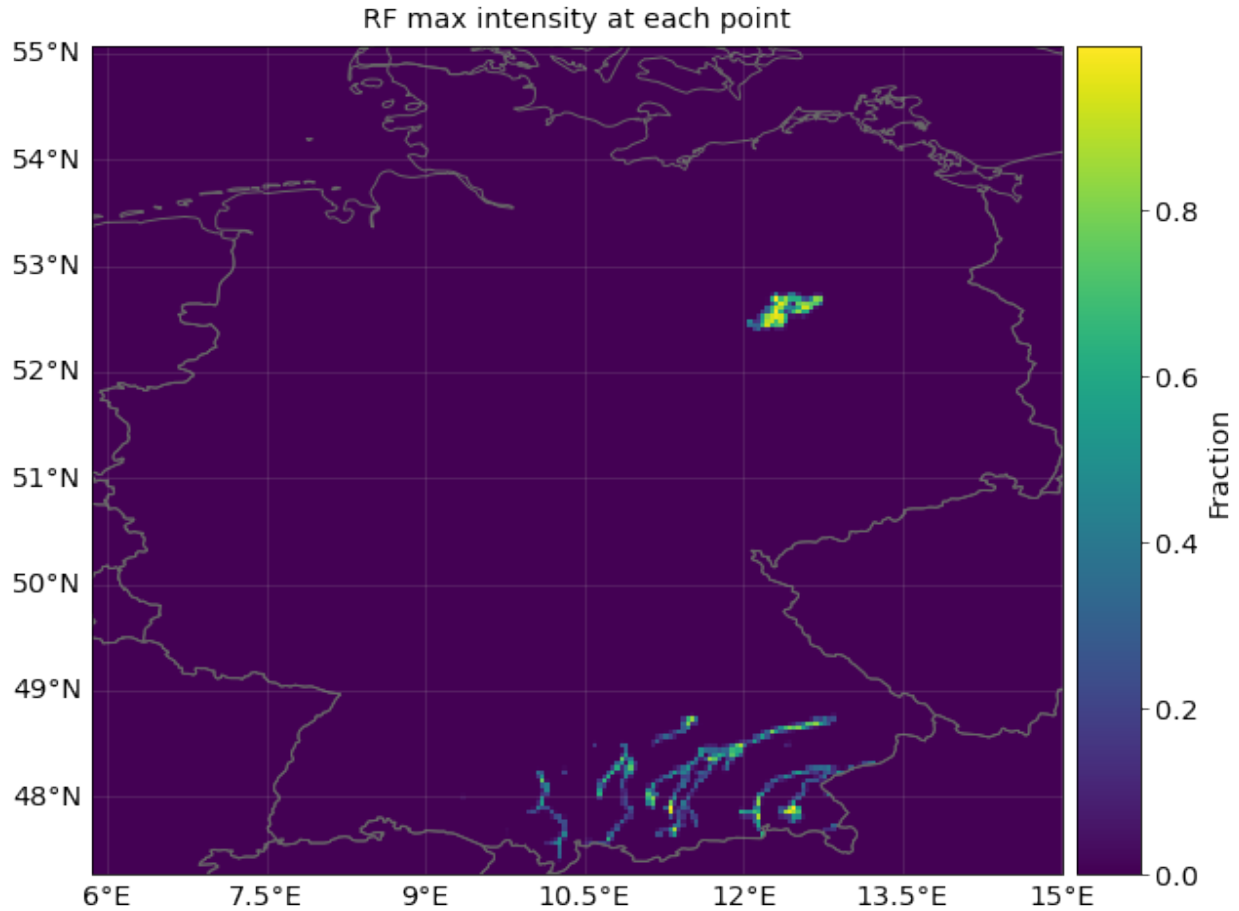
The fraction indicates the flooded part of a grid cell. It is possible to calculate the flooded area for each grid cell and for the whole area under consideration

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

```
#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
```



```
#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:

```
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,
```

(continues on next page)

(continued from previous page)

9444812.29294032, 628258.03278596, 1907510.51222833,
 3731174.16397245, 9034472.37433945, 13017186.7293233 ,
 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,

(continues on next page)

(continued from previous page)

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,
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,

(continues on next page)

(continued from previous page)

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,
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,

(continues on next page)

(continued from previous page)

```

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,
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])

```

3.3.4 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.

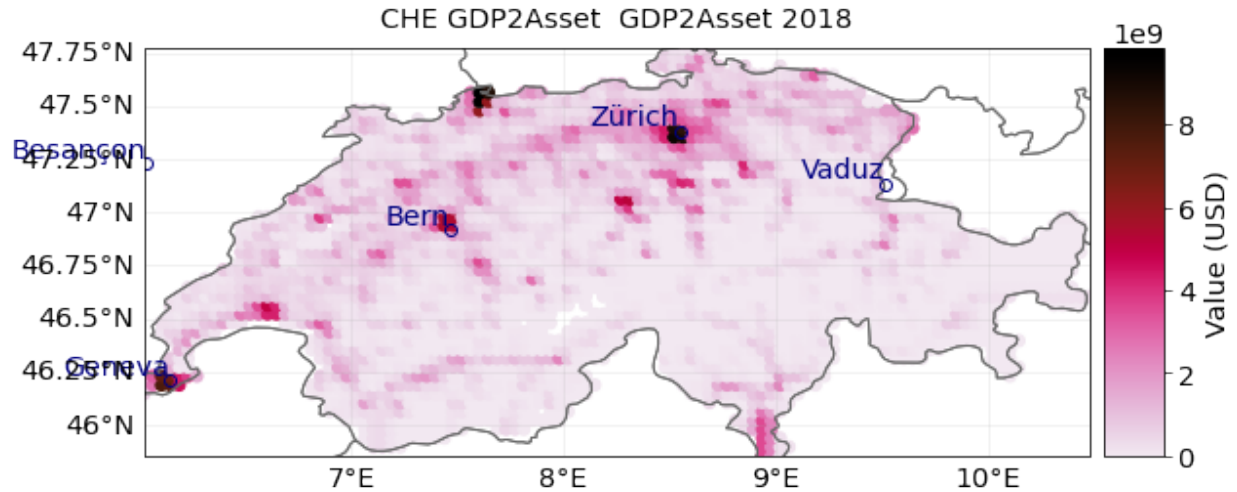
Murakami, D. and Yamagata, Y. (2019) ‘Estimation of gridded population and GDP scenarios with spatially explicit statistical downscaling’, Sustainability (Switzerland). Multidisciplinary Digital Publishing Institute, 11(7), p. 2106. doi: 10.3390/su11072106.

```
# 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
```

	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]

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



3.3.5 Setting JRC damage functions

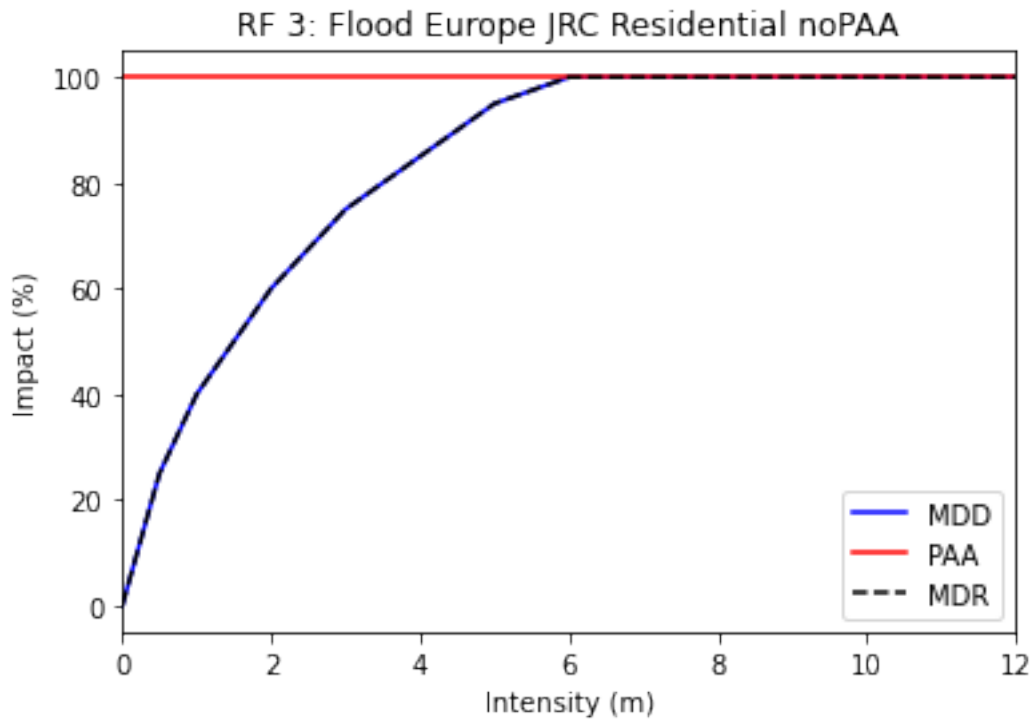
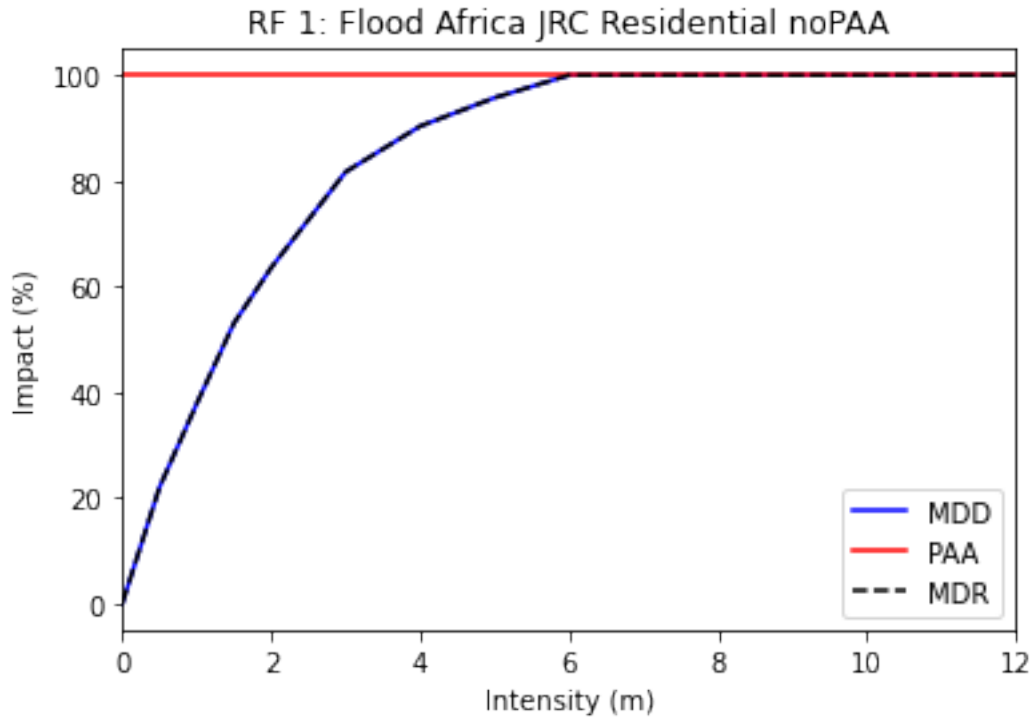
In CLIMADA we currently calculate damage by translating flood-depth into a damage factors. Damage assessments implemented in CLIMADA base on the residential damage functions basing on an empirical estimate published in the JRC report. Individual damage functions are available for six continents:

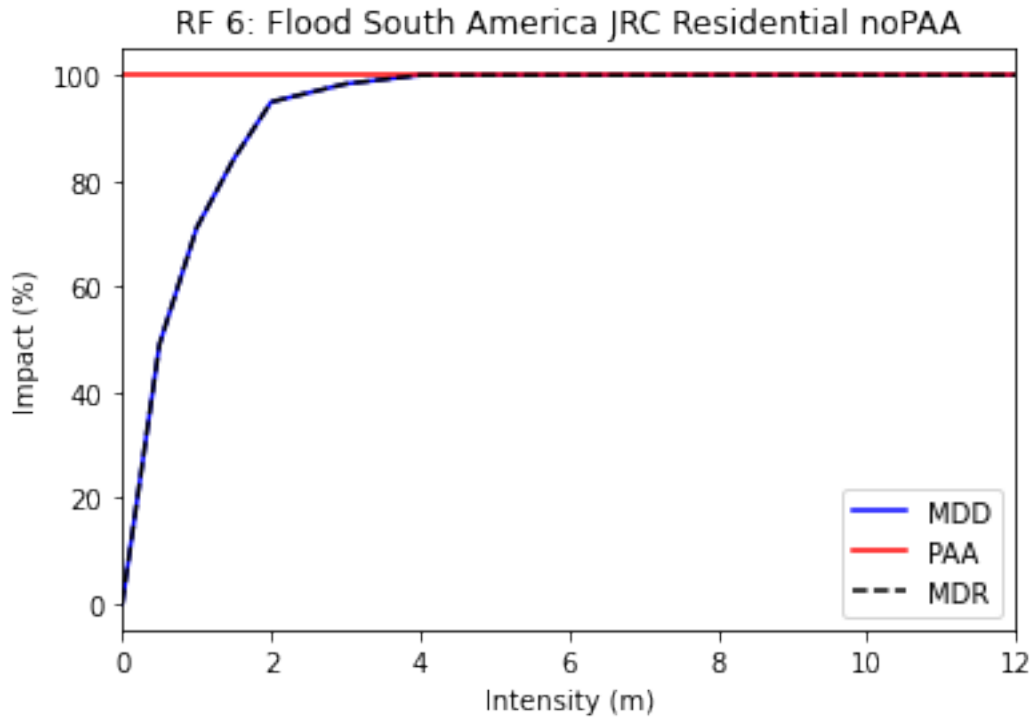
RF1: Africa RF2: Asia RF3: Europe RF4: North America RF5: Oceania RF6: South America

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.

```
# 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()
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 Geo-DataFrame:

```
gdpa.gdf
```

	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.

3.3.6 Deriving flood impact with LitPop exposure

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

```

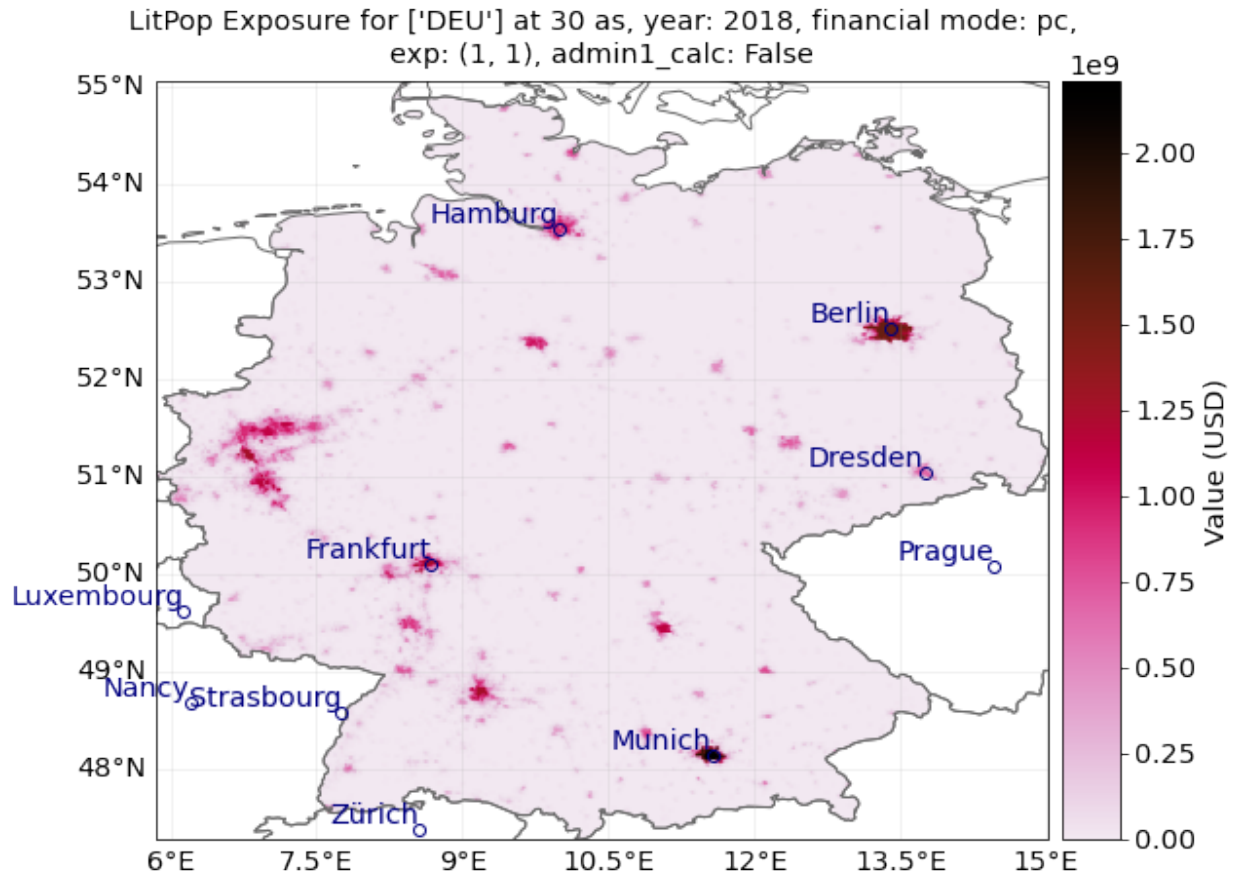
      value      geometry  latitude  longitude  \
0      83527.518952 POINT (6.72083 53.61250)  53.612500  6.720833
1      129830.381955 POINT (6.72917 53.61250)  53.612500  6.729167
2      175137.241455 POINT (6.73750 53.61250)  53.612500  6.737500
3      213973.213450 POINT (6.74583 53.61250)  53.612500  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]
```

```
# 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);
```

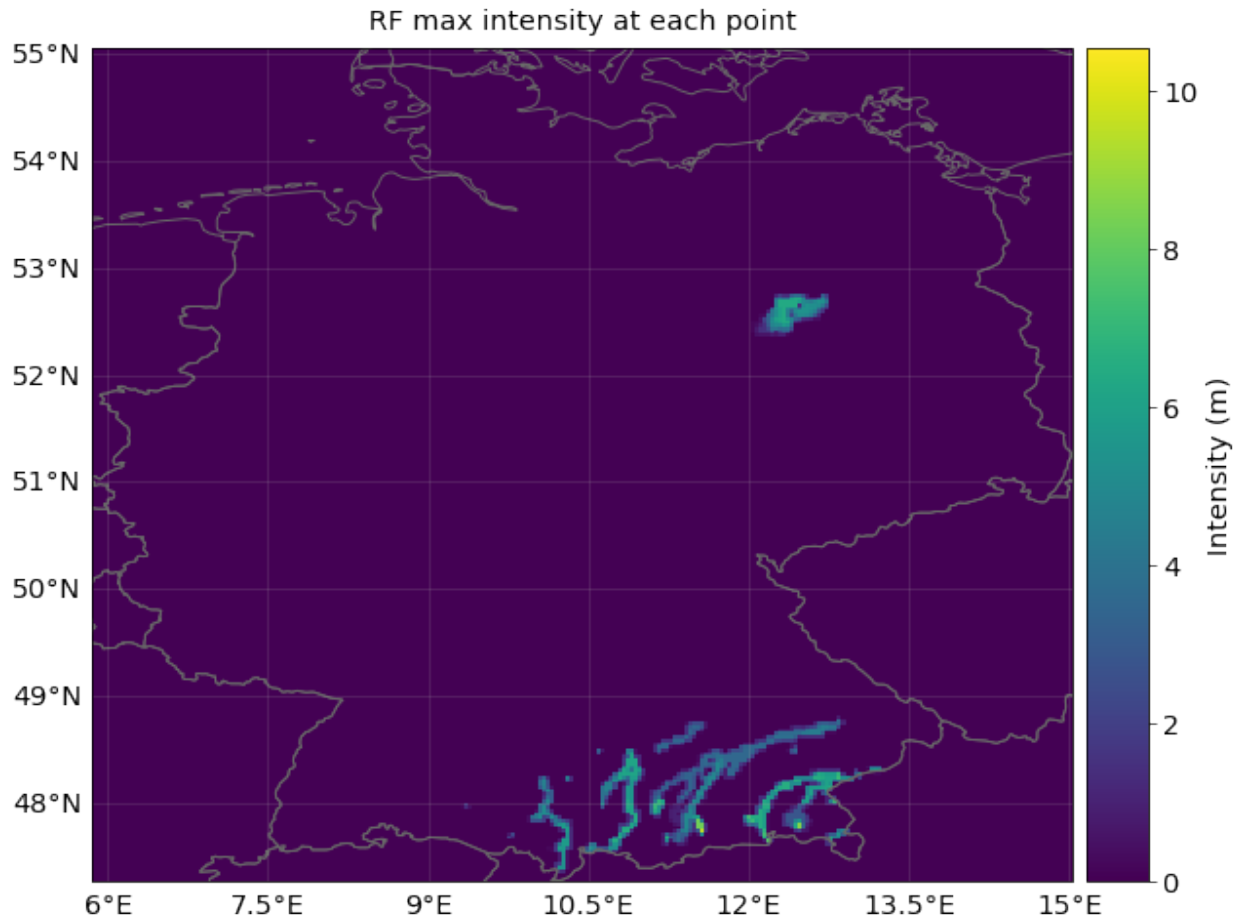


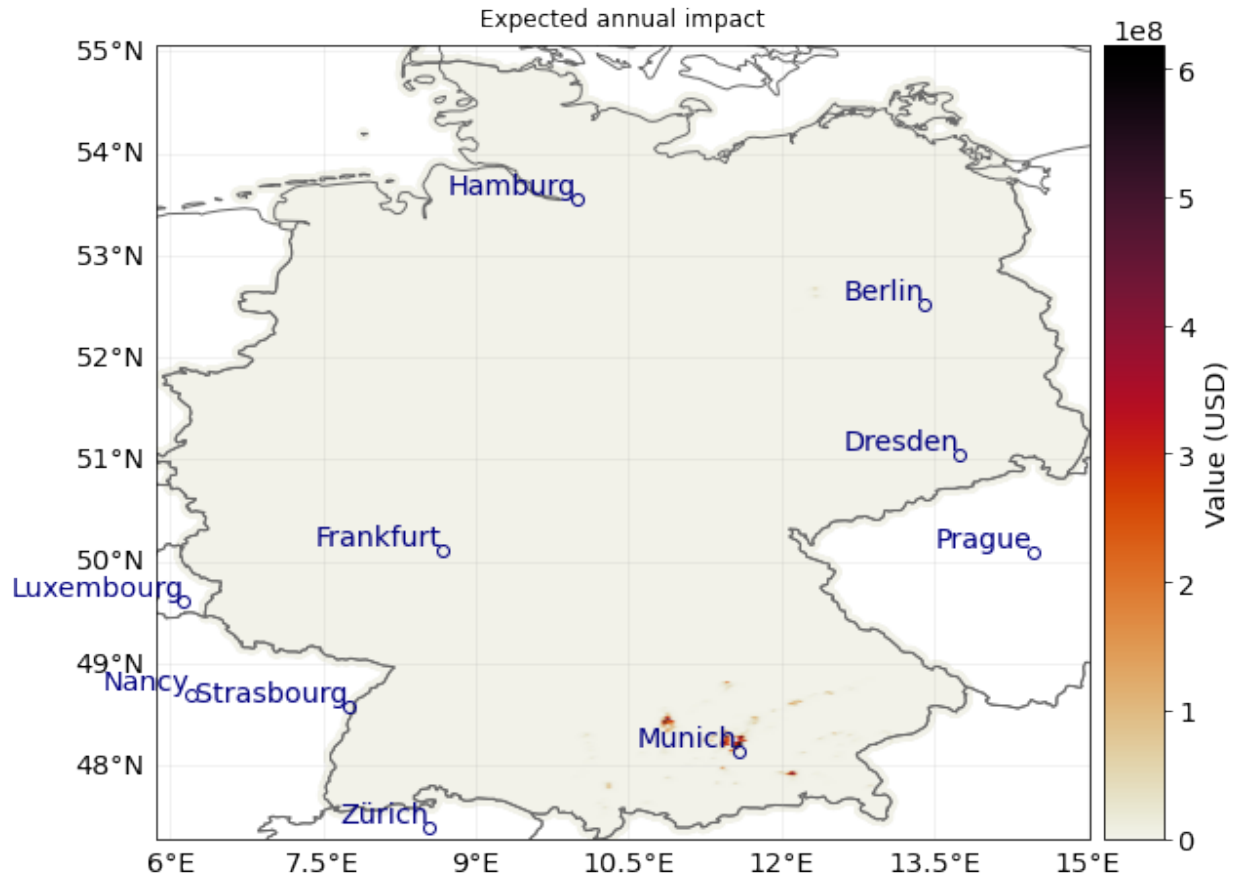
```

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();

```





3.4 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:

```
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()
```

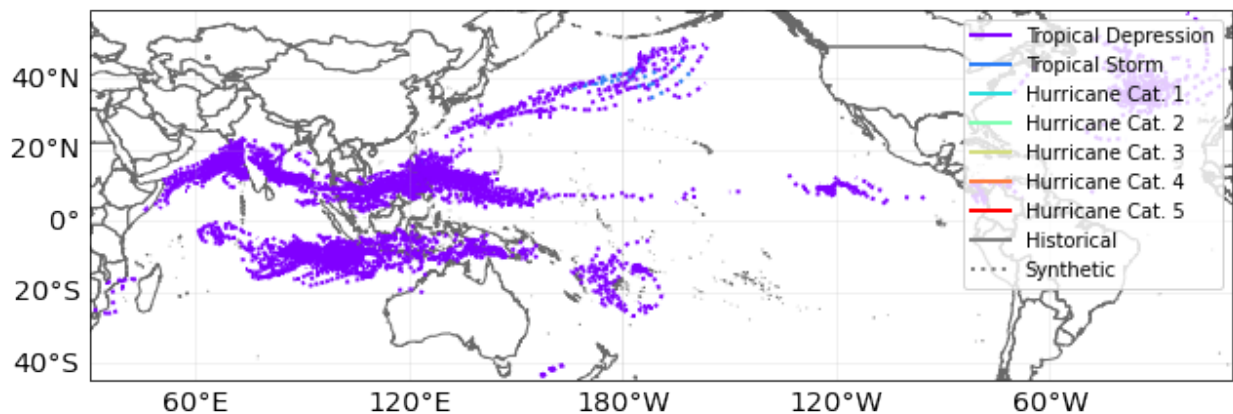
```
Processing ECMWF forecast. Date: 2021-11-16 17:49:27.152261
2021-11-16 17:49:28,816 - climada_petals.hazard.tc_tracks_forecast - INFO - Fetching_
↳BUFR tracks:
```

```
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:

```
forecast.plot()
```

```
<GeoAxesSubplot:>
```



The `TCTForecast` 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 `TCTForecast.fetch_buf_r_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 `TCTForecast.fetch_buf_r_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).

3.4.1 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.

3.4.2 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.

3.5 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".

3.5.1 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:

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

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

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.

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

import numpy as np
import matplotlib.pyplot as plt

l_T0 = np.array([3.1, 6.1, 11.0])
```

(continues on next page)

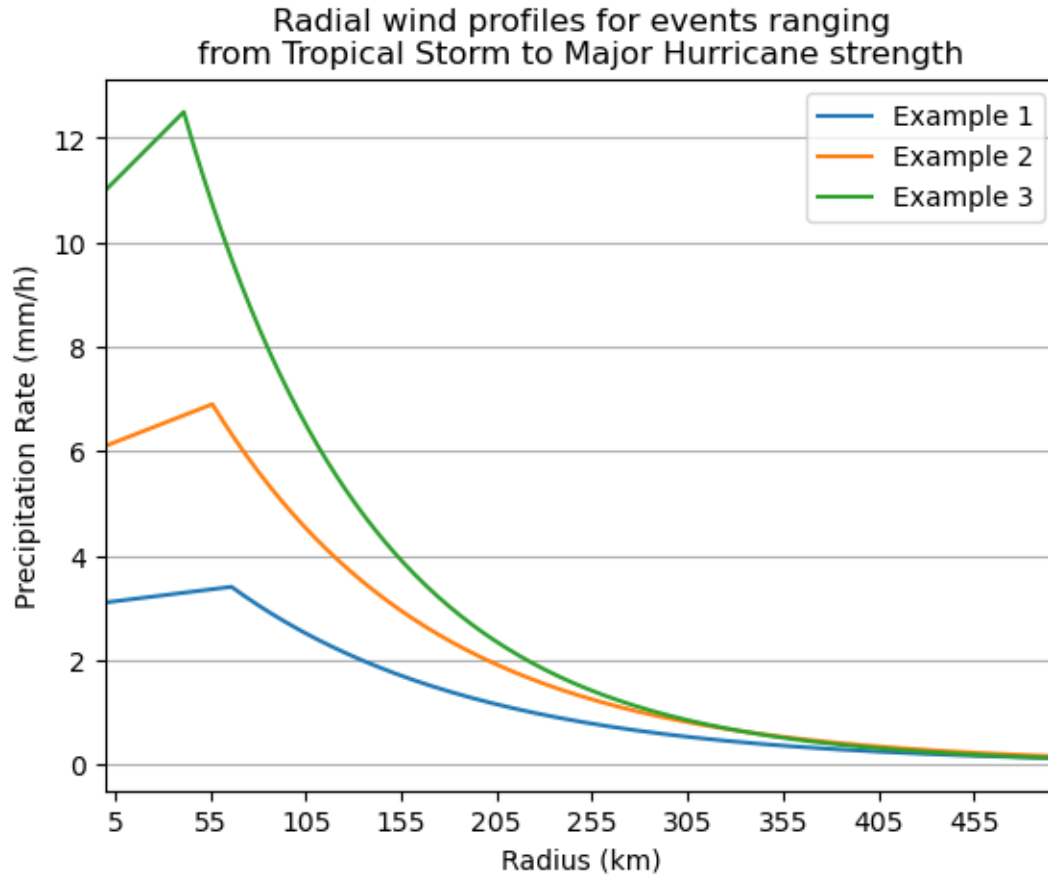
(continued from previous page)

```
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()
```



3.5.2 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-Clapeyron 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.

3.5.3 Examples

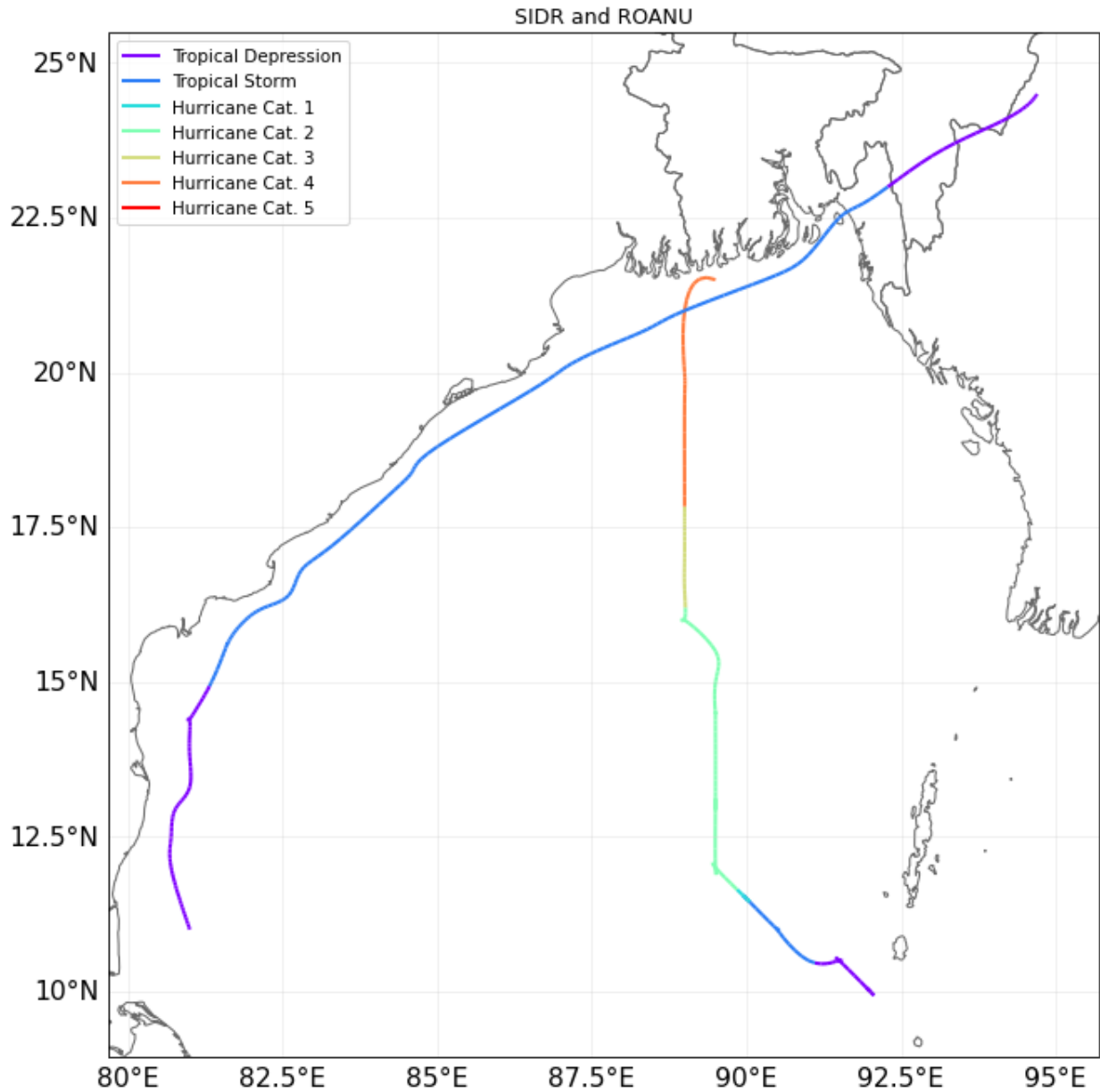
We compute the rain fields of Sidr 2007 and Roanu 2016 over Bangladesh as follows:

```
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.
```



```

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_on_land()
cent_bang = cent_bang.select(sel_cen=cent_bang.on_land)

```

Rain field output with R-CLIPER

```

from climada_petals.hazard import TCRain

tr_bang = TCRain.from_tracks(tracks, cent_bang, model="R-CLIPER")

```

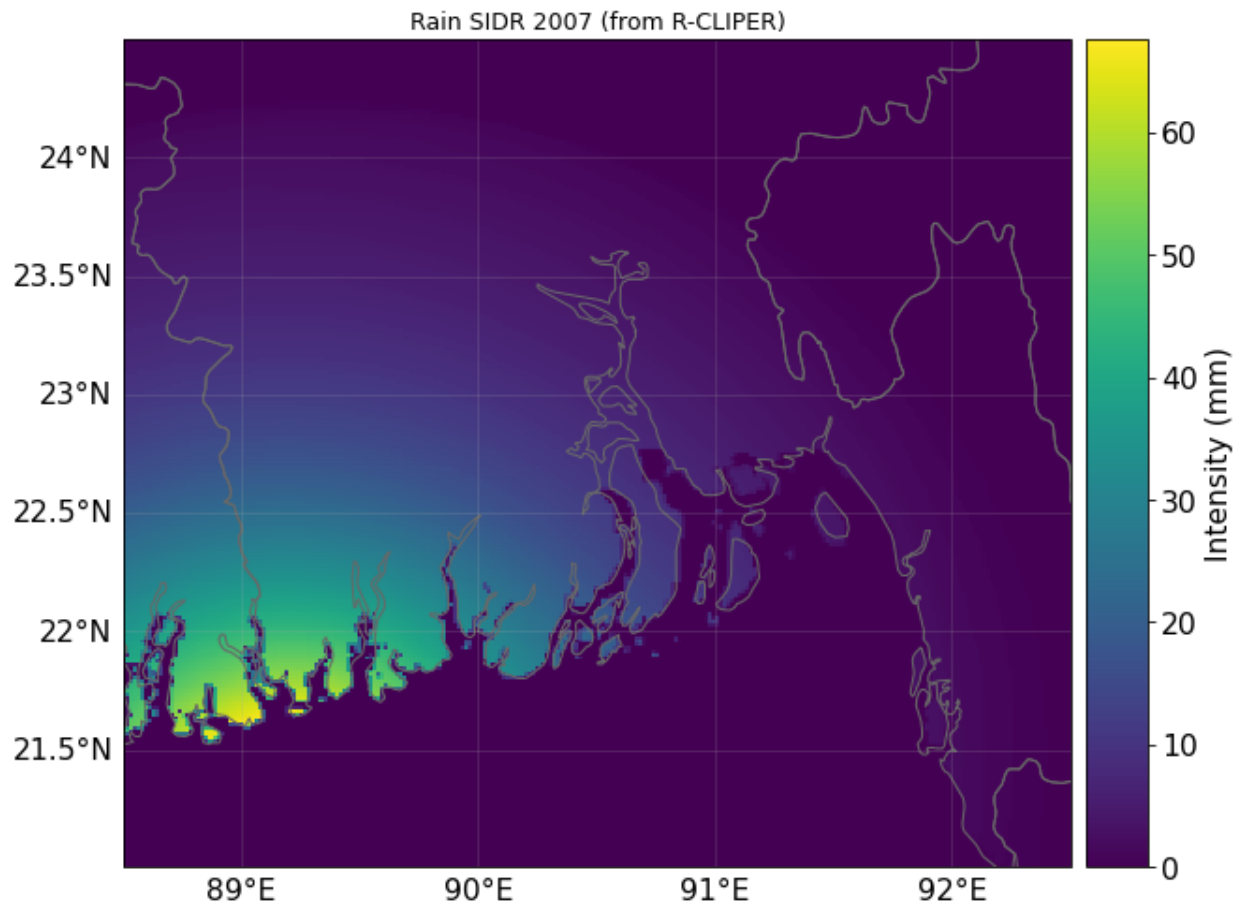
(continues on next page)

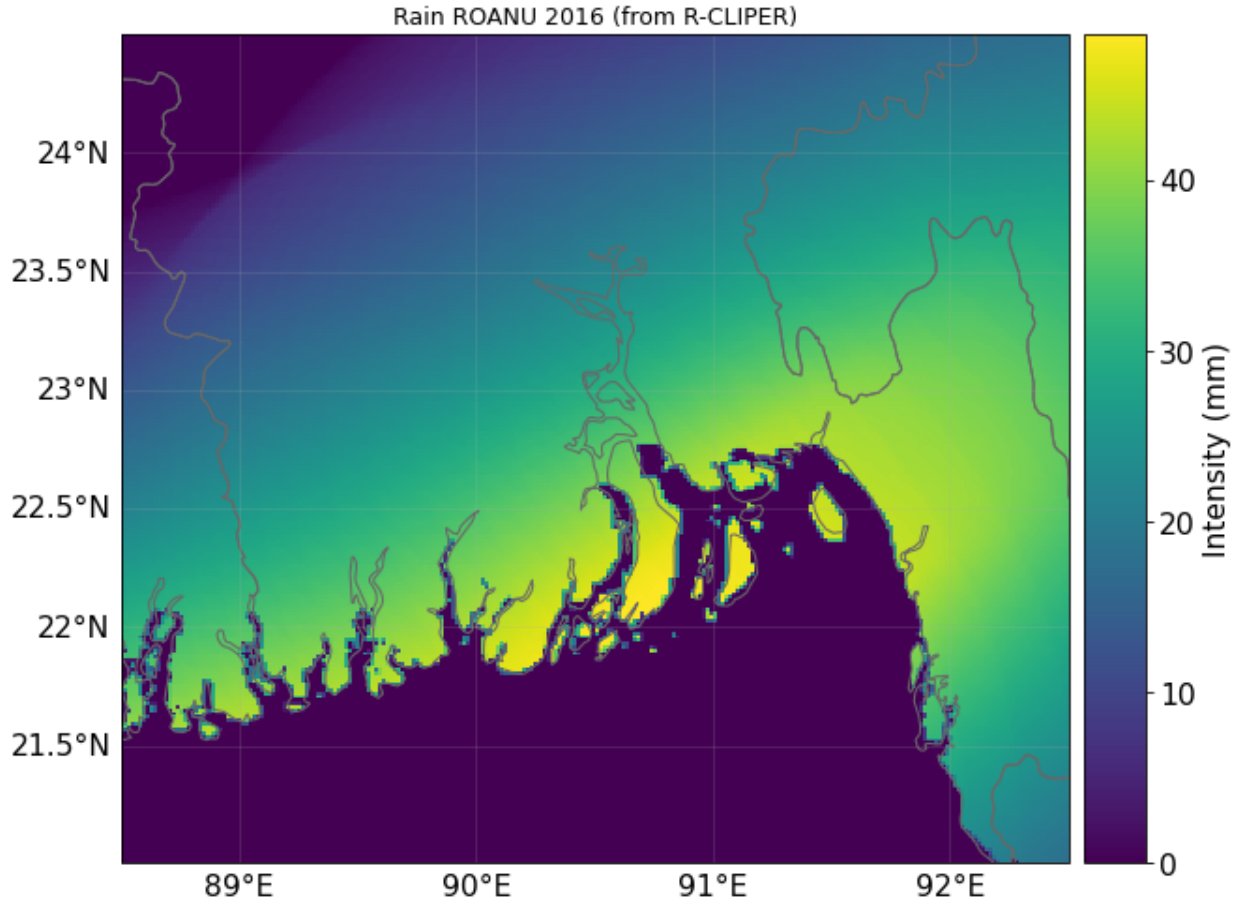
(continued from previous page)

```
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%
```



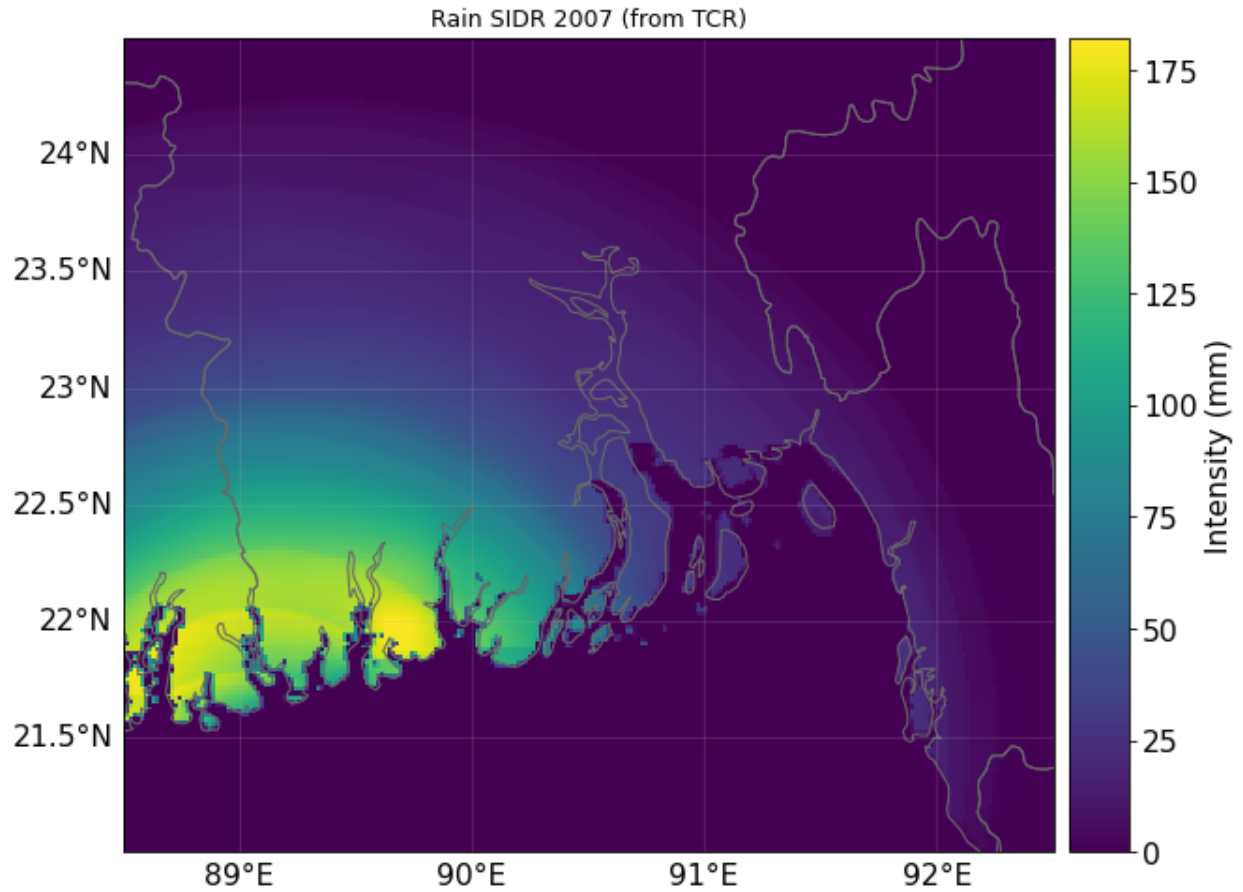


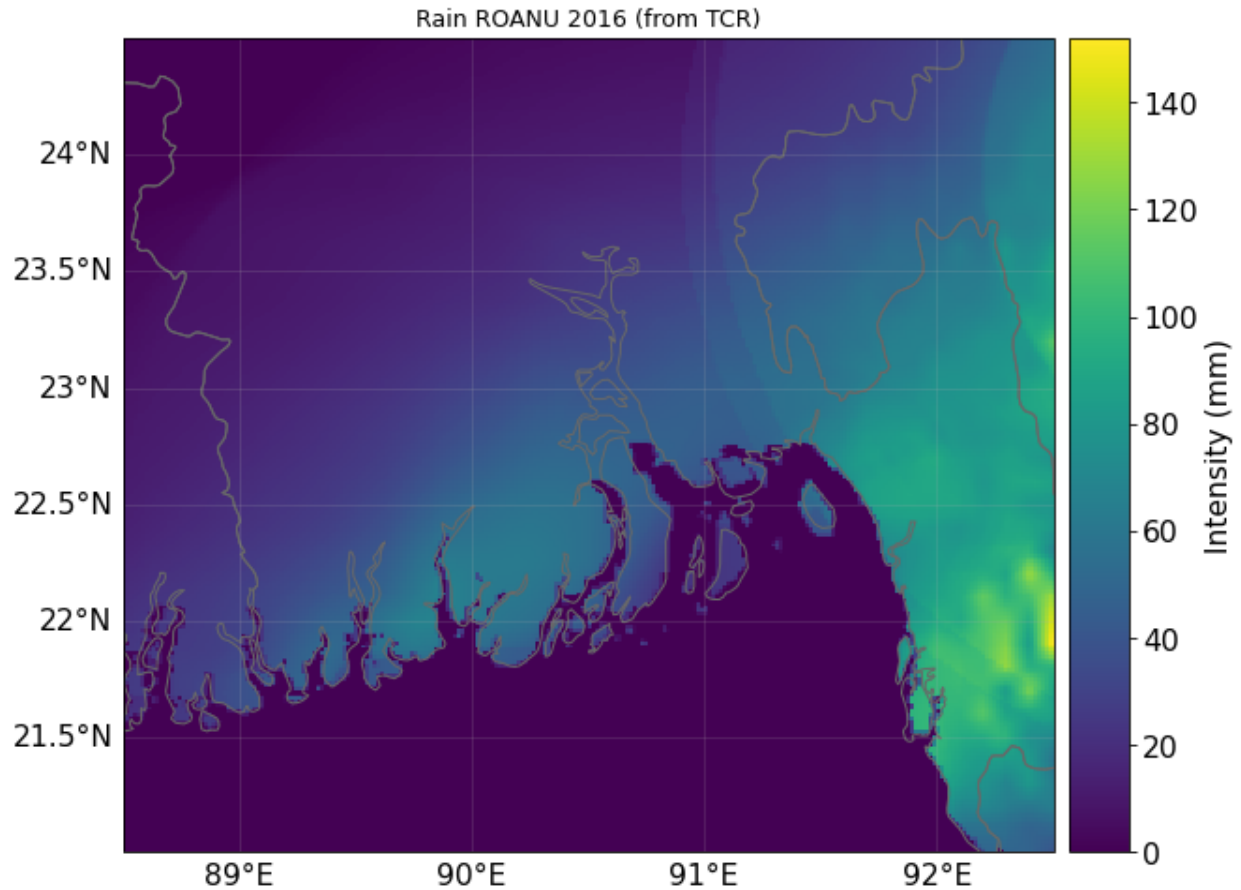
Rain field output with TCR

```
tr_bang = TCRain.from_tracks(tracks, 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%
```





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:

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

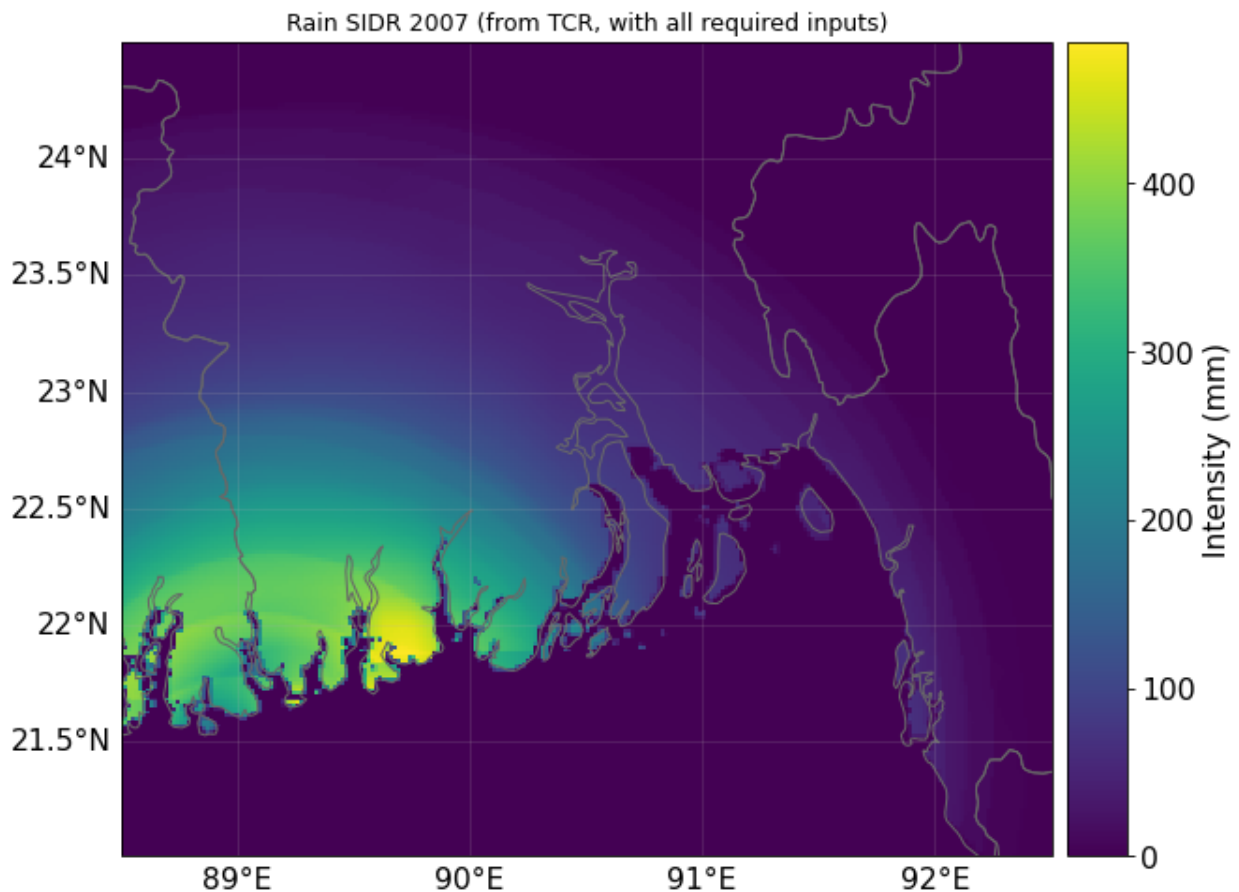
tracks = TCTracks.from_hdf5(tcrain_examples)
tracks.equal_timestep(0.5)

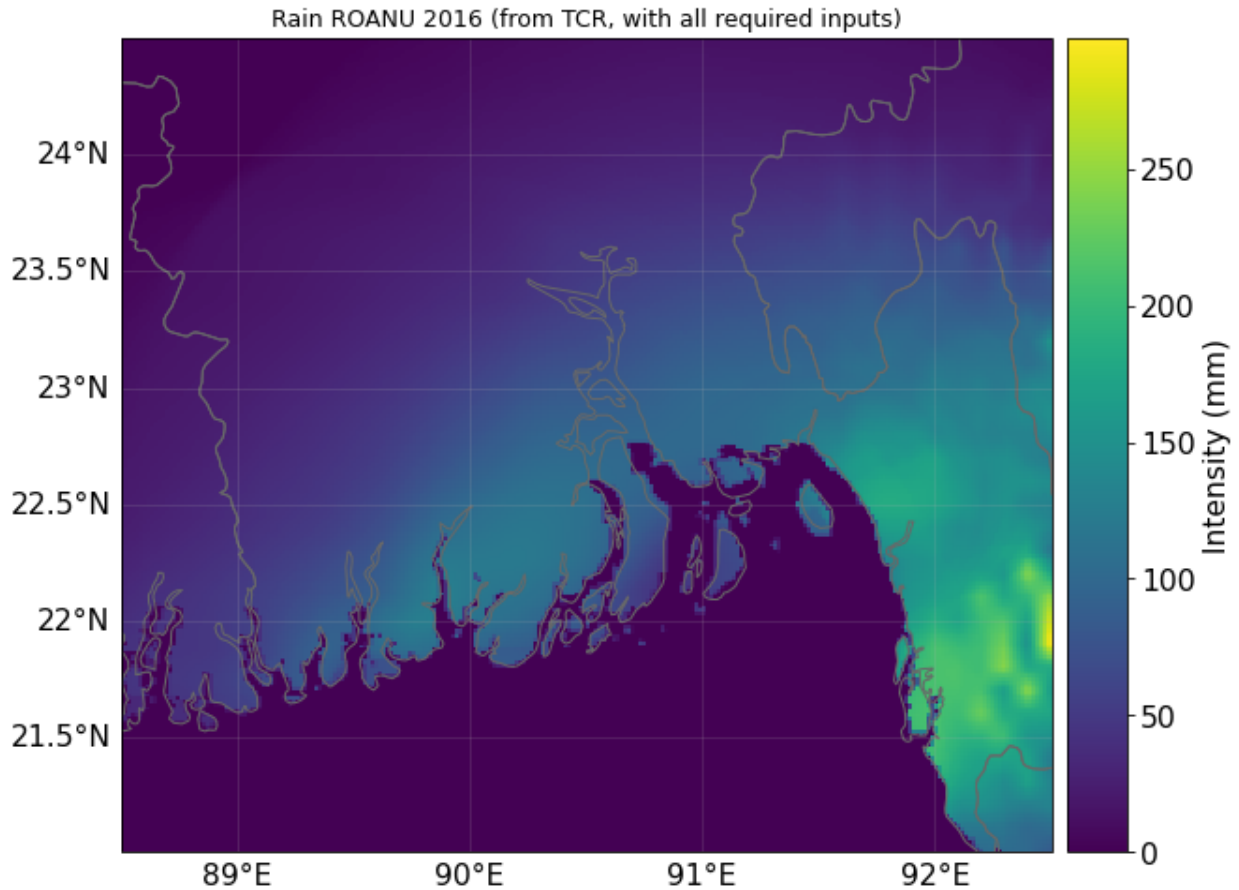
tr_bang = TCRain.from_tracks(tracks, cent_bang, model="TCR")
ax = tr_bang.plot_intensity(1)
ax.set_title('Rain SIDR 2007 (from TCR, with all required inputs)')
ax = tr_bang.plot_intensity(2)
ax.set_title('Rain ROANU 2016 (from TCR, with all required inputs)');
```

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





3.6 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`.

3.6.1 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.

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

# conversion factors
mph2ms = 0.44704;
f2m = 0.3048;

# the points read from the SLOSH graph
v0 = 60*mph2ms;
v1 = 140*mph2ms;
```

(continues on next page)

(continued from previous page)

```

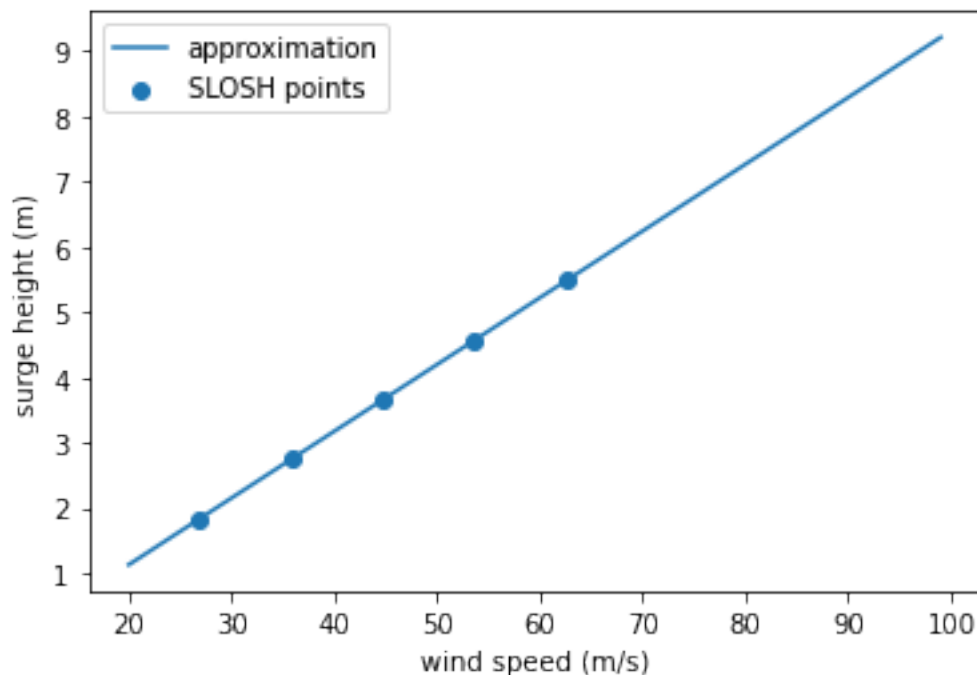
s0 = 6*f2m;
s1 = 18*f2m;

# 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.

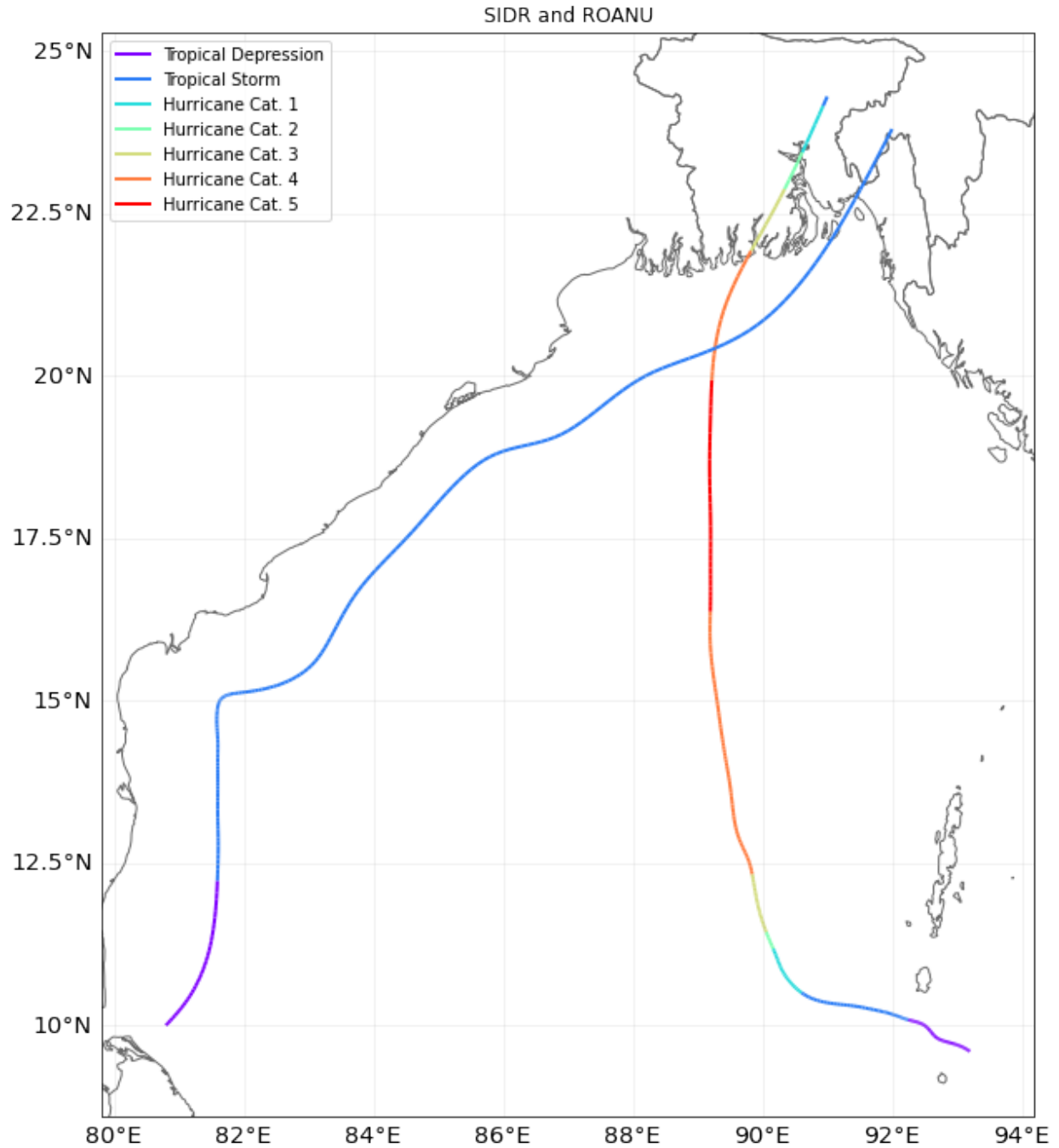
3.6.2 Example

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

```
%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.
```



```
# 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)

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%
```

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

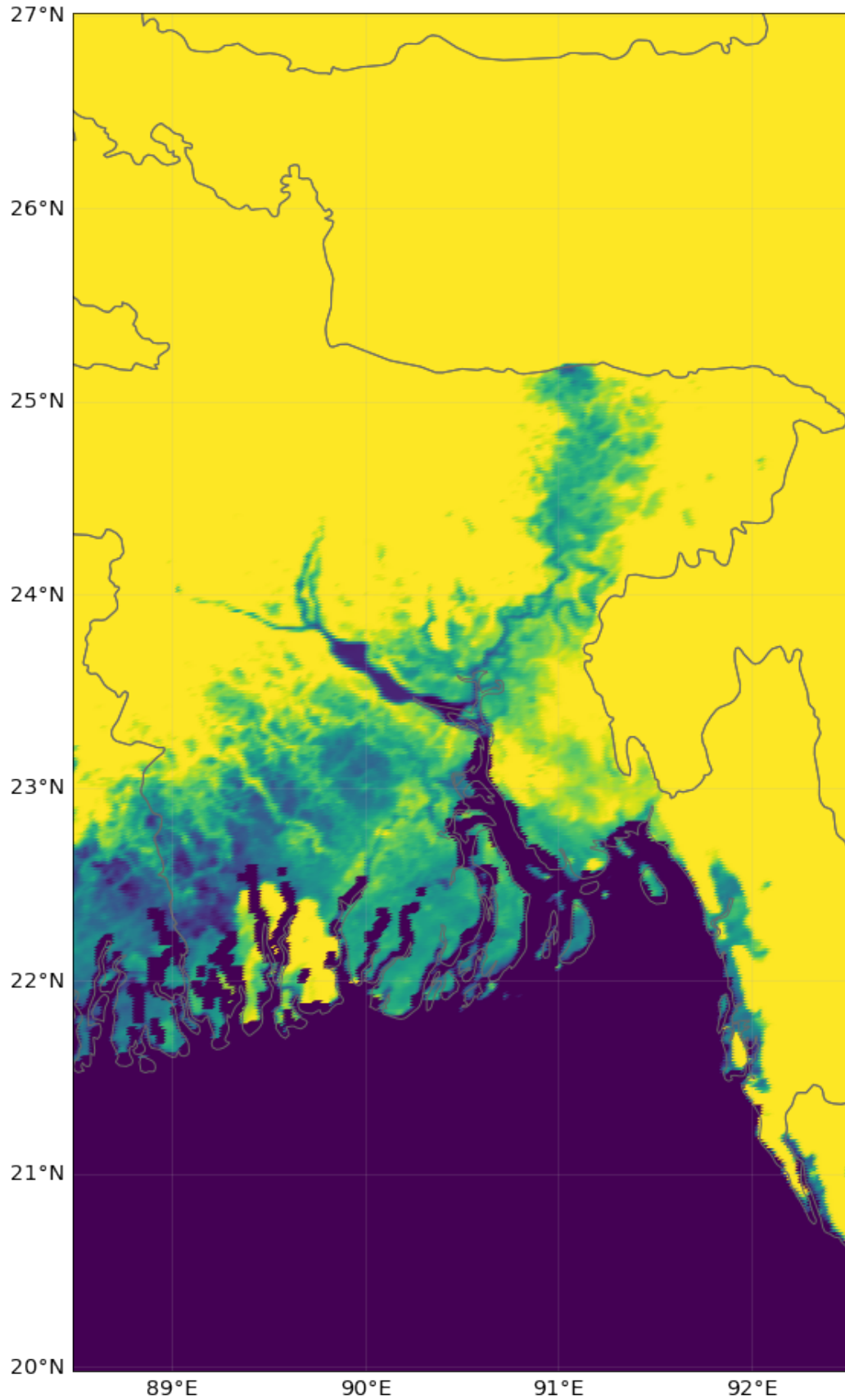
# If you have the global SRTM15+V2.0 elevation data set, you can replace the following
# sample DEM data set by your SRTM15+V2.0.tiff:
topo_path = DEMO_DIR.joinpath('SRTM15+V2.0_sample.tiff')
ts_bang = TCSurgeBathtub.from_tc_winds(tc_bang, topo_path)
```

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

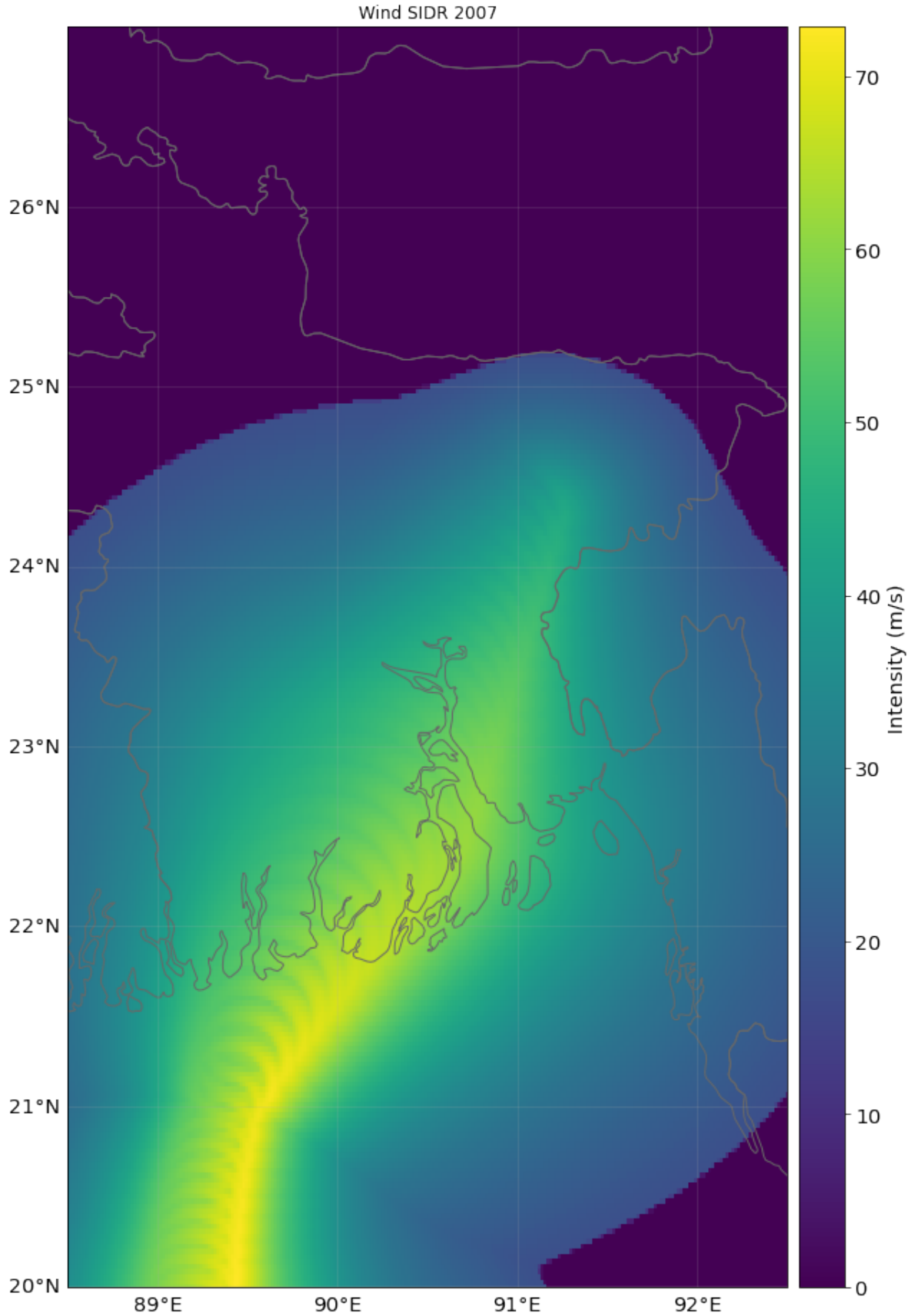
```
# plot elevation of the raster
elevation = ts_bang.centroids.get_elevation(topo_path)
ts_bang.centroids.plot(c=elevation, vmin=0, vmax=10)
```

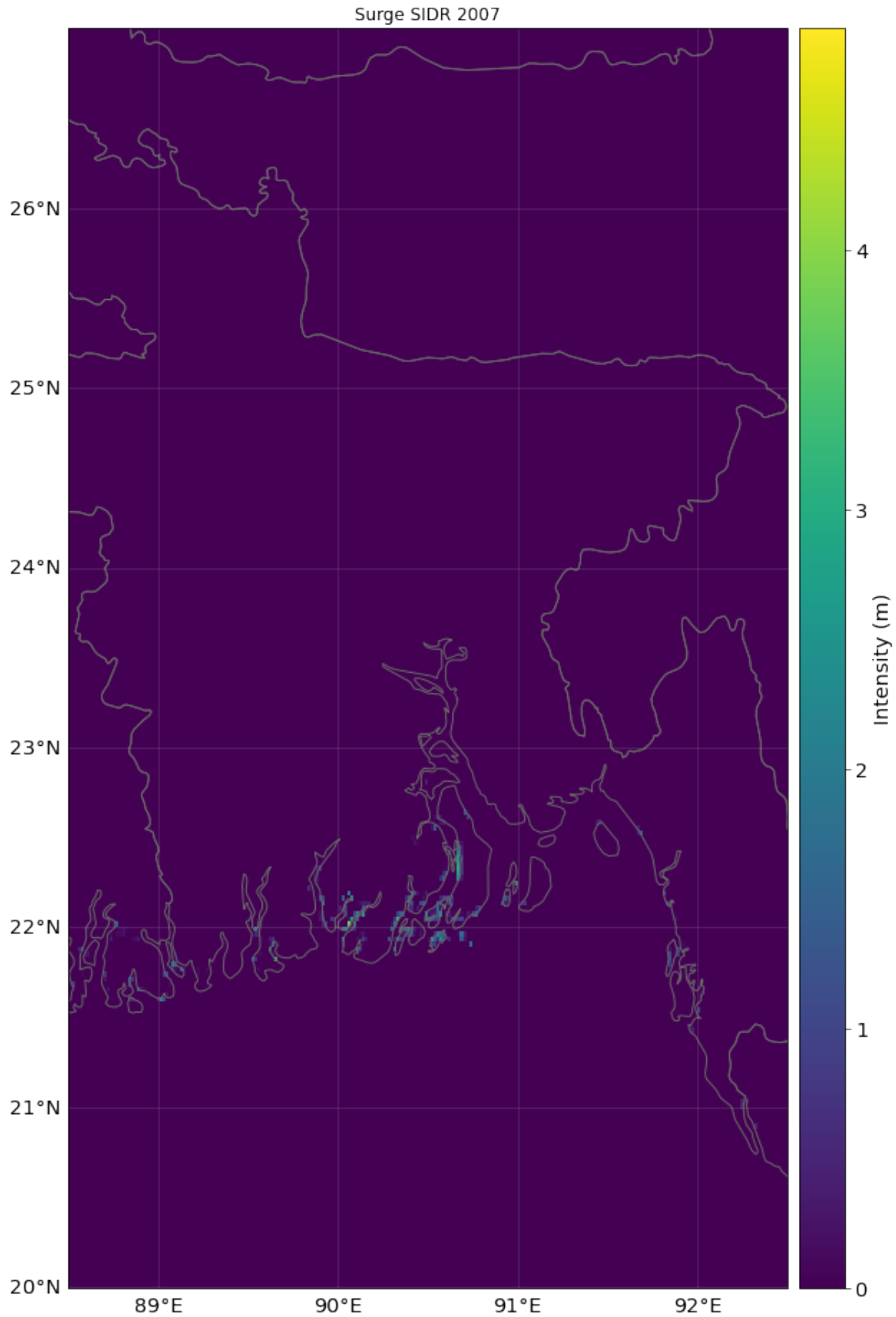
```
2021-07-09 15:24:41,544 - climada.util.coordinates - INFO - Sampling from /home/
↳tovogt/.climada/demo/data/SRTM15+V2.0_sample.tiff
```

```
<GeoAxesSubplot:>
```

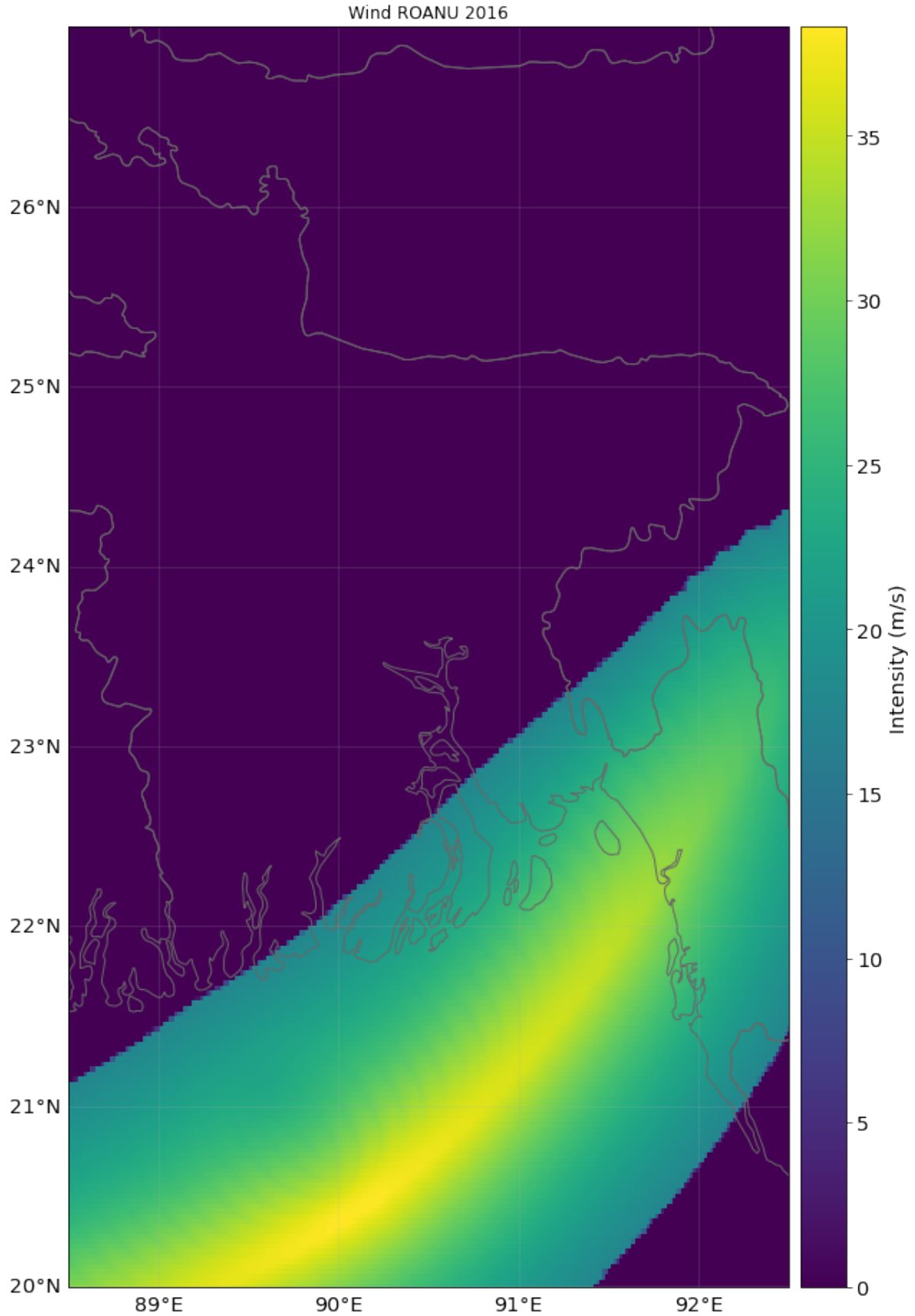


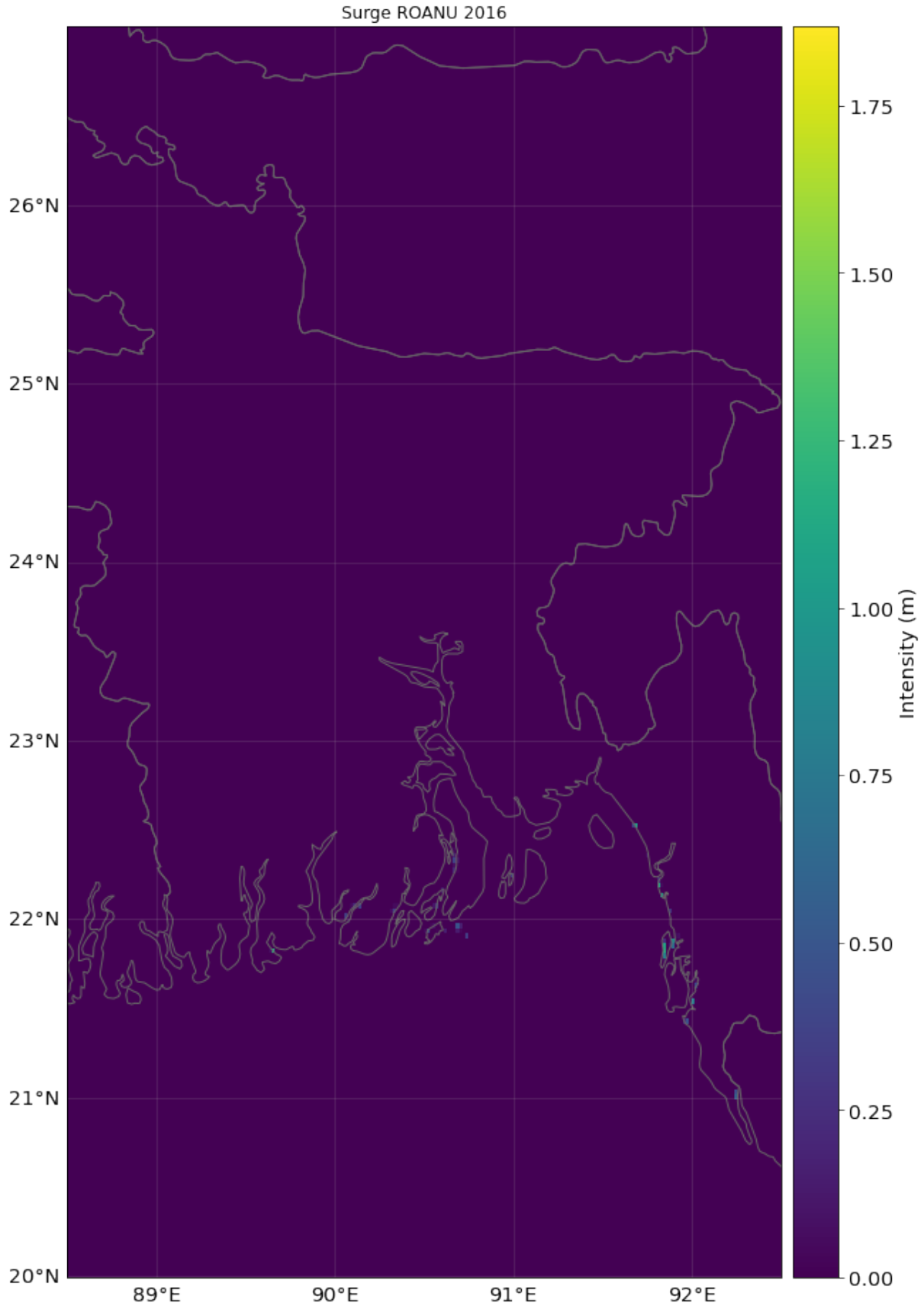
```
# plot wind and surge SIDR
ax = tc_bang.plot_intensity(1)
ax.set_title('Wind SIDR 2007')
ax = ts_bang.plot_intensity(1)
ax.set_title('Surge SIDR 2007');
```





```
# 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');
```





3.7 Surge from tropical cyclones modeled with GeoClaw

Major damages from tropical cyclones (TC) are caused by coastal flooding from storm surge. Those damages are badly represented when using only windfields as a predictor. CLIMADA's `TCSurgeGeoClaw` class employs the geophysical flow solver `GeoClaw` to predict TC surge damages at a resolution of up to 9 arc-seconds (approximately 270 meters, depending on latitude). For more information about `GeoClaw`, please refer to [the official web page](#) and to the following publications:

- Berger, M. J., George, D. L., LeVeque, R. J. and Mandli, K. M.: *The GeoClaw software for depth-averaged flows with adaptive refinement*, *Advances in Water Resources* 34 (2011), pp. 1195-1206. DOI
- Mandli, K. T. & Dawson, C. N.: *Adaptive Mesh Refinement for Storm Surge*. *Ocean Modelling* 75, 36–50 (2014). DOI

3.7.1 Performance warning

Even though `GeoClaw` uses an efficient adaptive mesh refinement to model wave dynamics in the 2d depth-averaged shallow water equations, modeling storm surge is the computationally most challenging task included in CLIMADA. Computing the inundated areas from a single tropical cyclone might take between minutes and hours depending on the resolution, the storm and the host machine's resources.

3.7.2 Input data

Three kinds of input data are required by `TCSurgeGeoClaw`: storm track, sea level and topographic data. CLIMADA's `TCTracks` class handles several kinds of TC track datasets, including best track data from the IBTrACS dataset of historical tropical cyclones. Sea level and topographic data have to be provided by the user in raster file format.

Since the quality of any storm surge model is dependent on good topographic data in coastal areas, it is highly recommended to choose the best regionally available **digital elevation model (DEM)** on a case-by-case basis instead of global DEMs such as SRTM. Note that, in this context, topography also includes bathymetry or underwater topography (sea floor). Bathymetry of sufficient quality is available as `SRTM15+V2.3` or `GEBCO Gridded Bathymetry`. High-quality coastal DEM data is provided for the US coast by the `CoNED project`. A global dataset with focus on coastal areas is `CoastalDEM`, but users should always consider regionally available sources on a case-by-case basis for better results. Note that, if several DEM datasets are combined, all of them need to be translated to the same reference geoid (so-called vertical datum), such as the geoid model EGM96.

The most severe damages from TC storm surge are caused by a combination of high seasonal tides and waves driven by TC winds. That's why `TCSurgeGeoClaw` requires **gridded sea surface height data** as an input. An at least monthly temporal resolution is required in order to take into account seasonal variability. Starting from 1993, monthly sea surface anomaly data from satellite altimetry is provided by the `Copernicus project`. The variable "Absolute dynamic topography" contains sea surface anomaly relative to the reference geoid "GOCO05s" (see Mulet et al. 2021) with monthly resolution. Make sure to reproject the data so that it is compatible with the reference geoid/vertical datum used in the DEM input (e.g. the geoid model EGM96).

3.7.3 Setup

Note that this feature only works on Linux and Mac since Windows is not supported by `GeoClaw`. Due to the high computational demand, this functionality should be run on an HPC cluster with decent amounts of memory and processors (at least 32 GB and 8 cores) available. Only for testing purposes, it makes sense to run this functionality on a smaller machine at lower resolution, e.g. when running this tutorial notebook.

It is required to run the method `TCSurgeGeoClaw.from_tc_tracks` (or the function `climada_petals.hazard.tc_surge_geoclaw.setup_clawpack`) with a working internet connection at least once to trigger the download and installation of the flow solver (`GeoClaw`).

Moreover, before you can run several instances of this method in parallel, e.g. on an HPC cluster, make sure to run a single instance of this method since all instances of this method will be accessing the same version of the solver, and

compilation might be triggered in all instances at the same time. The same applies if you want to recompile using the option `recompile=True`.

By default, the flow solver (GeoClaw) is configured to use multiple OpenMP threads with their number equal to the number of physical CPU cores in the machine. You can change this behavior by setting the environment variable `OMP_NUM_THREADS` to the desired number of threads. Note, however, that changes to `OMP_NUM_THREADS` will only be effective if you set `recompile=True` at least once.

3.7.4 Notes about this tutorial

This tutorial notebook computes coastal flooding caused by a single storm (Katrina, 2005) in a comparably small geographical region (New Orleans, Louisiana) at a comparably coarse spatial resolution (300 arc-seconds, or approximately 10 km). With this configuration, it should be possible to execute this tutorial on a personal laptop computer in a few minutes. However, in practice, it is advisable to choose a higher resolution (e.g. 30 arc-seconds), and run the flow solver on an HPC cluster node (e.g. with 16 CPU cores and 64 GB RAM).

```
# Choose project-specific sea level and topographic datasets

from climada.util.api_client import Client
client = Client()

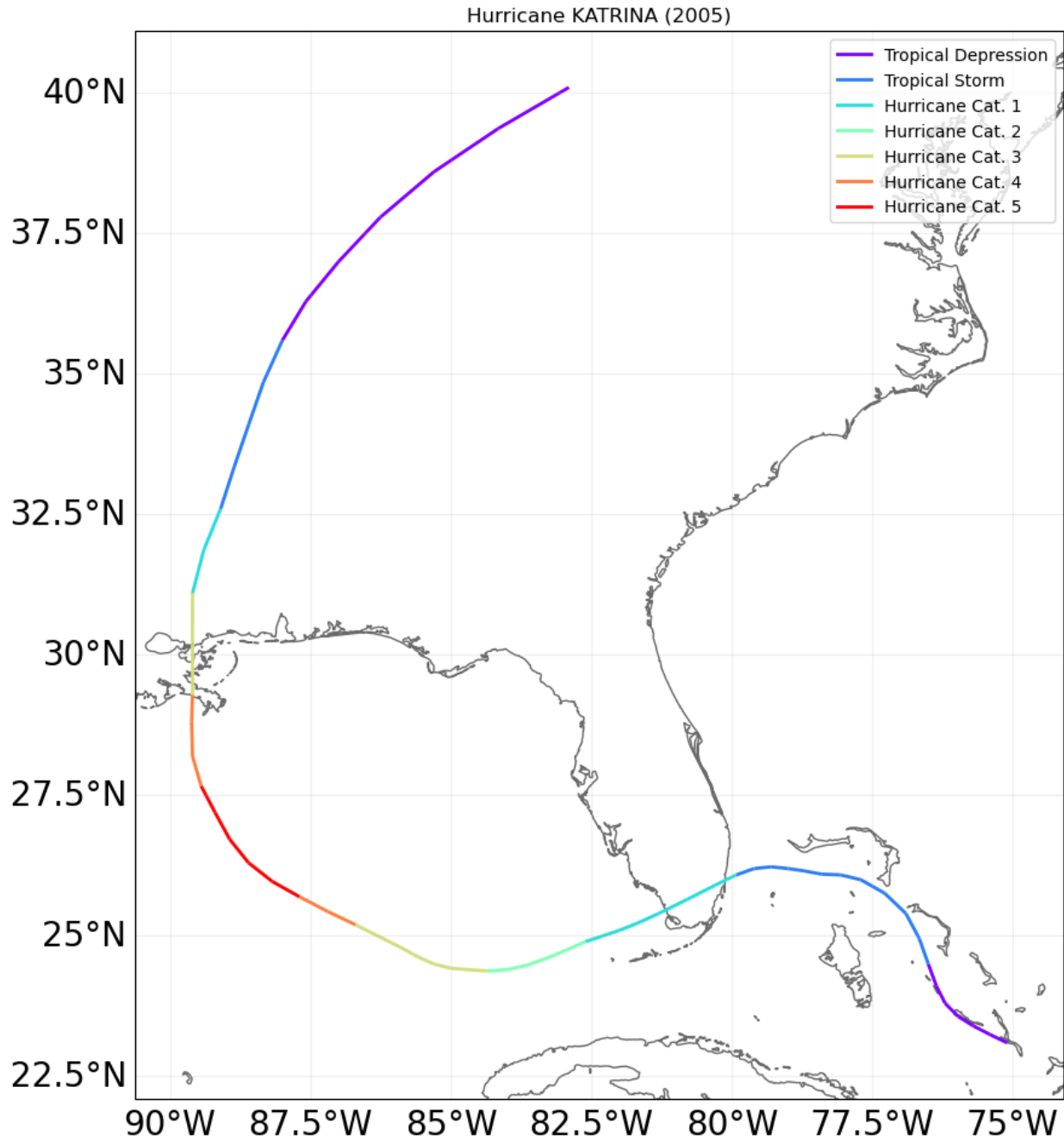
# sample of Copernicus satellite altimetry for year 2005 and relative to EGM96 geoid
_, [ALTIMETRY_PATH] = client.download_dataset(
    client.get_dataset_info(name='tutorial_altimetry_gulf', status='package-data')
)

# sample from SRTM15+V2.3 combined bathymetry and topography at 300 arc-seconds
↳resolution
_, [TOPO_PATH] = client.download_dataset(
    client.get_dataset_info(name='tutorial_bathymetry_gulf', status='package-data')
)
```

```
# storm track data is taken from IBTrACS
from climada.hazard.tc_tracks import TCTracks
tracks = TCTracks.from_ibtracs_netcdf(storm_id="2005236N23285")
tracks.plot().set_title('Hurricane KATRINA (2005)');
```

```
/home/tovogt/.local/share/miniconda3/envs/climada_env_3.9/lib/python3.9/site-packages/
↳dask/dataframe/_pyarrow_compat.py:17: FutureWarning: Minimal version of pyarrow
↳will soon be increased to 14.0.1. You are using 11.0.0. Please consider upgrading.
warnings.warn(
```

```
2024-01-31 09:37:42,039 - climada.hazard.tc_tracks - WARNING - The cached IBTrACS
↳data set dates from 2023-05-12 20:44:48 (older than 180 days). Very likely, a more
↳recent version is available. Consider manually removing the file /home/tovogt/.
↳climada/data/IBTrACS.ALL.v04r00.nc and re-running this function, which will
↳download the most recent version of the IBTrACS data set from the official URL.
2024-01-31 09:37:42,934 - climada.hazard.tc_tracks - INFO - Progress: 100%
```



3.7.5 Running the GeoClaw surge model from CLIMADA

For each run of GeoClaw, a working directory is created in `data/geoclaw/runs` in the CLIMADA data directory. Before the GeoClaw solver is run, the storm is analysed to predict which coastal areas are expected to be affected by the storm's surge. Furthermore, if a single storm involves multiple landfall events, each landfall event is modeled in a separate GeoClaw run. (Note that this example is only for a single landfall event to avoid long runtimes.)

Usually, it is not needed to inspect the internal working directories, but it is helpful for debugging purposes. The geographical areas of landfall are plotted and saved to a file `event_areas.pdf` in the working directory (also plotted inline below). Furthermore, plots of the DEM data used for each GeoClaw run are generated and saved to files `dems.pdf` in subdirectories of the working directory (also plotted inline below).

```

from climada.hazard import Centroids
from climada_petals.hazard.tc_surge_geoclaw import TCSurgeGeoClaw, sea_level_from_nc

# bounds : box around New Orleans, Louisiana:
bounds = (-95, 25, -85, 35)

# Note that the resolution of the centroids does not affect the internal resolution_
↳of the flow solver.
# In this example, we take a resolution of 150 arc-seconds for the centroids, but the_
↳internal
# resolution of the flow solver will only depend on the resolution of the topography.
centroids = Centroids.from_pnt_bounds(bounds, res=150 / (60 * 60))

# sea_level_fun : function that extracts sea level from a file for a given
#                 temporal period and spatial bounds
sea_level_fun = sea_level_from_nc(ALTIMETRY_PATH)

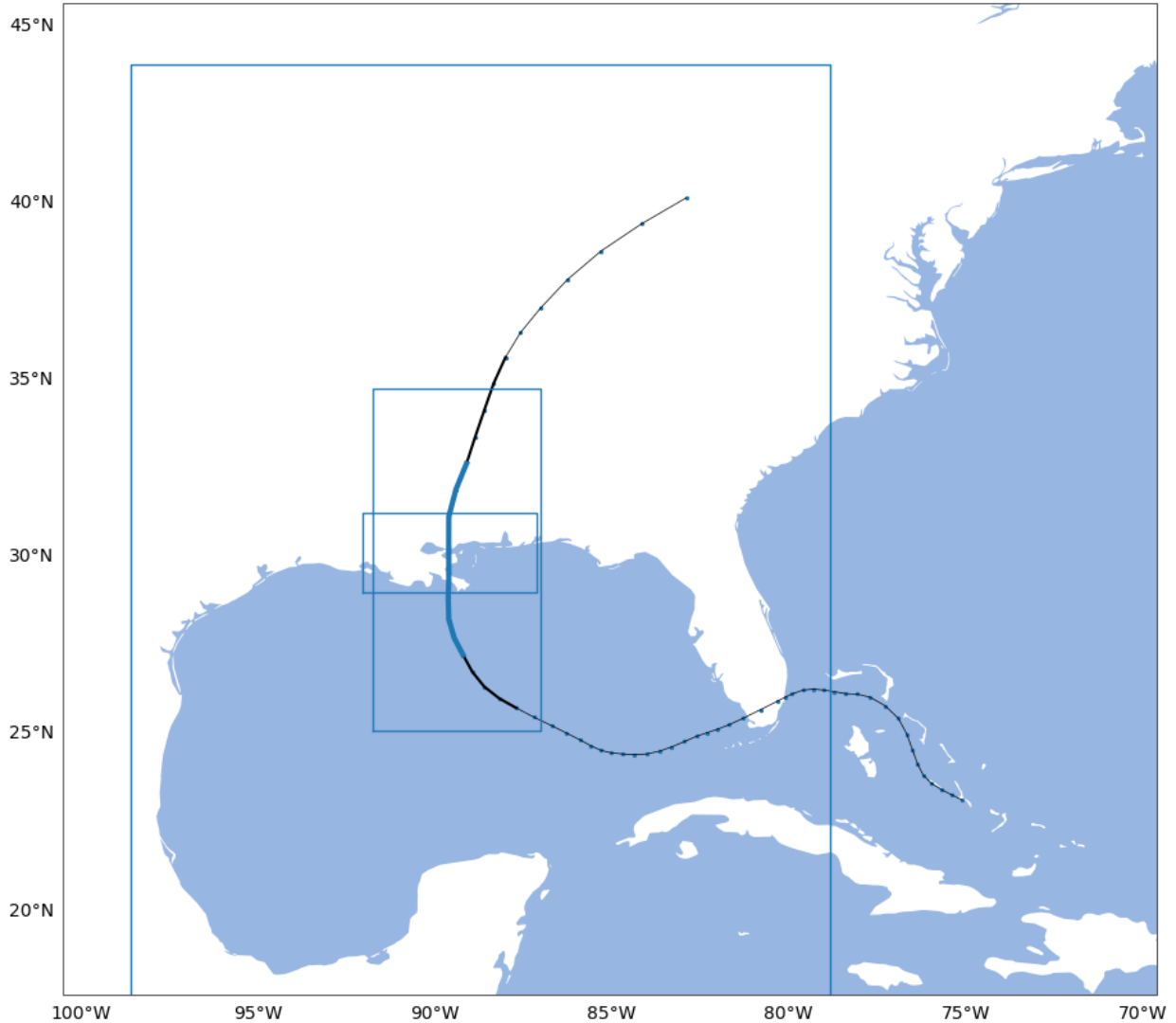
# The internal resolution of the flow solver is automatically adjusted to the `topo_
↳res_as` keyword argument.
# It is always chosen to be one order of magnitude higher than `topo_res_as` (e.g. ~
↳75 arc-seconds for topo_res_as=300).
# If the specified topography data has higher resolution, it will be aggregated to_
↳`topo_res_as`.
haz = TCSurgeGeoClaw.from_tc_tracks(
    tracks, centroids, TOPO_PATH,
    max_dist_offshore_km=5,
    geoclaw_kwargs=dict(
        topo_res_as=300,
        sea_level=sea_level_fun,
    ),
)

```

```

2024-01-31 09:37:47,030 - climada_petals.hazard.tc_surge_geoclaw - INFO - Reading sea_
↳level data from altimetry_2005.nc
2024-01-31 09:37:47,041 - climada_petals.hazard.tc_surge_geoclaw - INFO - Sea level_
↳data available within bounds (261.125, 16.125, 281.875, 43.875)
2024-01-31 09:37:47,044 - climada_petals.hazard.tc_surge_geoclaw - INFO - Sea level_
↳data available within period from 2005-08 till 2005-10
2024-01-31 09:37:47,096 - climada_petals.hazard.tc_surge_geoclaw - INFO - Found_
↳Clawpack version 5.9.2 in /home/tovogt/.climada/data/geoclaw/src/clawpack
2024-01-31 09:37:47,163 - climada.util.coordinates - INFO - Sampling from /home/
↳tovogt/.climada/data/GMT_intermediate_coast_distance_01d.tif
2024-01-31 09:37:47,251 - climada_petals.hazard.tc_surge_geoclaw - INFO - Computing_
↳TC surge of 1 tracks on 5289 centroids.
2024-01-31 09:37:47,253 - climada.util.coordinates - INFO - Sampling from srtm_300as.
↳tif
2024-01-31 09:37:47,275 - climada_petals.hazard.tc_surge_geoclaw - INFO - Determine_
↳georegions and temporal periods of landfall events ...

```



```

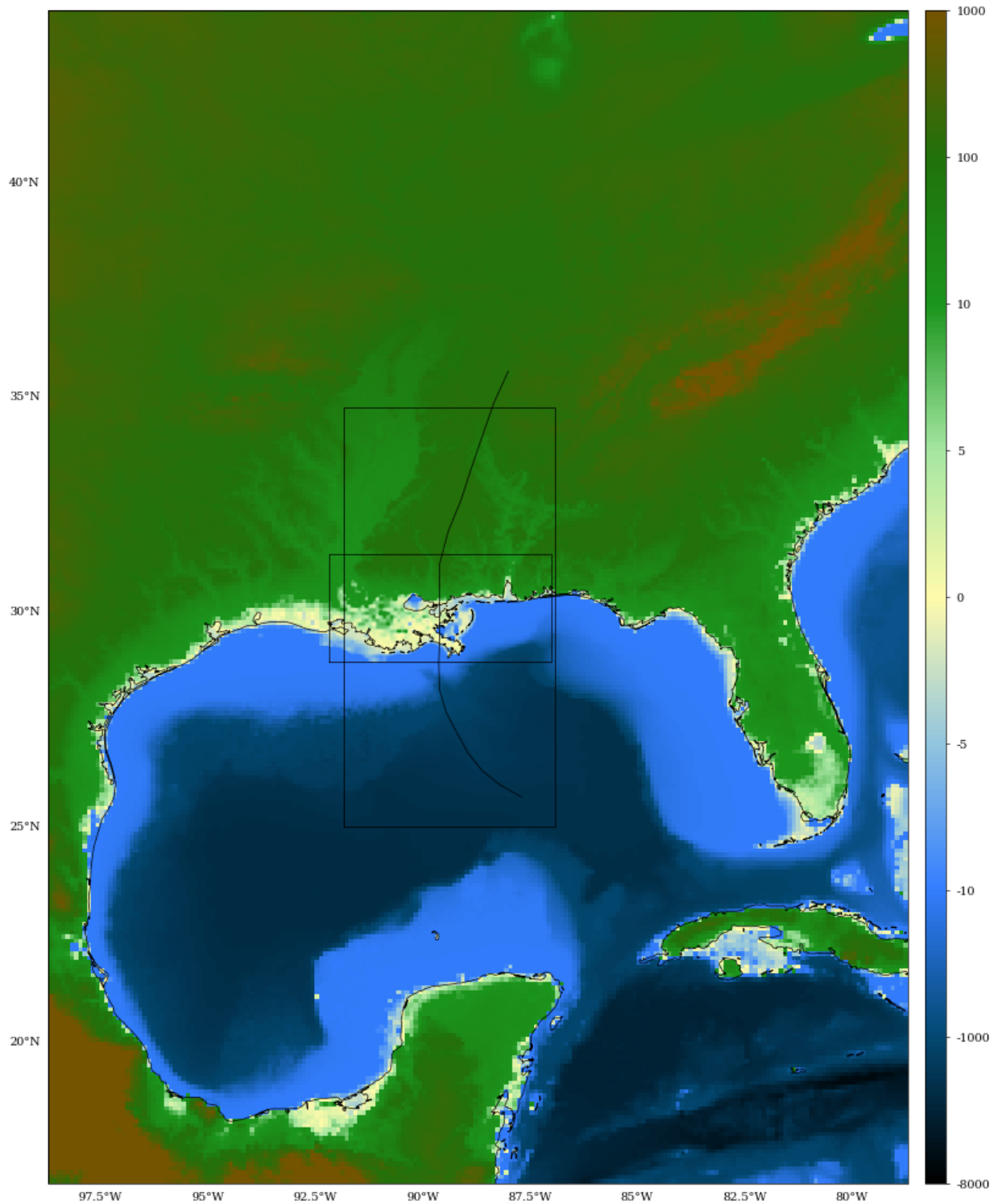
2024-01-31 09:37:49,800 - climada_petals.hazard.tc_surge_geoclaw - INFO - Starting 1
↳runs of GeoClaw ...
2024-01-31 09:37:49,802 - climada_petals.hazard.tc_surge_geoclaw - INFO - Prepare
↳GeoClaw to determine surge on 1967 centroids
2024-01-31 09:37:49,805 - climada_petals.hazard.tc_surge_geoclaw - INFO - Init
↳GeoClaw working directory: /home/tovogt/.climada/data/geoclaw/runs/2024-01-31-
↳09374752-2005236N23285/2005-08-29-00
2024-01-31 09:37:49,806 - climada_petals.hazard.tc_surge_geoclaw - INFO - Found
↳Clawpack version 5.9.2 in /home/tovogt/.climada/data/geoclaw/src/clawpack
2024-01-31 09:37:49,846 - climada_petals.hazard.tc_surge_geoclaw - INFO - GeoClaw
↳resolution in arc-seconds: ['889.75', '444.88', '222.44', '74.15']
2024-01-31 09:37:49,881 - climada_petals.hazard.tc_surge_geoclaw - INFO - Load
↳elevation data [360, (-98.59017181396484, 16.817861557006836, -78.81785583496094,
↳43.86089324951172)] from srtm_300as.tif
2024-01-31 09:37:49,929 - climada_petals.hazard.tc_surge_geoclaw - INFO - Load
↳elevation data [300, (-91.75763702392578, 25.070945739746094, -86.99645233154297,
↳34.70354461669922)] from srtm_300as.tif
2024-01-31 09:37:49,941 - climada_petals.hazard.tc_surge_geoclaw - INFO - Load

```

(continues on next page)

(continued from previous page)

```
↪elevation data [300, (-92.04249999999966, 28.957500000000344, -87.12416666666577, ↪  
↪31.20916666666688)] from srtm_300as.tif
```



```

2024-01-31 09:37:51,591 - climada_petals.hazard.tc_surge_geoclaw - INFO - Running_
↳GeoClaw in /home/tovogt/.climada/data/geoclaw/runs/2024-01-31-09374752-
↳2005236N23285/2005-08-29-00 ...
2024-01-31 09:37:52,409 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 0%
2024-01-31 09:38:06,493 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 10%
2024-01-31 09:38:19,996 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 20%
2024-01-31 09:38:30,294 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 30%
2024-01-31 09:38:37,937 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 40%
2024-01-31 09:38:47,096 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 50%
2024-01-31 09:38:54,911 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 60%
2024-01-31 09:39:01,754 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 70%
2024-01-31 09:39:07,986 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 80%
2024-01-31 09:39:13,421 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 90%
2024-01-31 09:39:18,736 - climada_petals.hazard.tc_surge_geoclaw - INFO - 2005-08-29-
↳00: 100%
2024-01-31 09:39:18,736 - climada_petals.hazard.tc_surge_geoclaw - INFO - Reading_
↳GeoClaw output ...

```

```

# The Hazard object can be used just like any other Hazard object, e.g.:
haz.write_hdf5("tc_surge_katrina_2005.h5")
haz.plot_intensity(event=0, smooth=False, vmin=0, vmax=4)

```

```

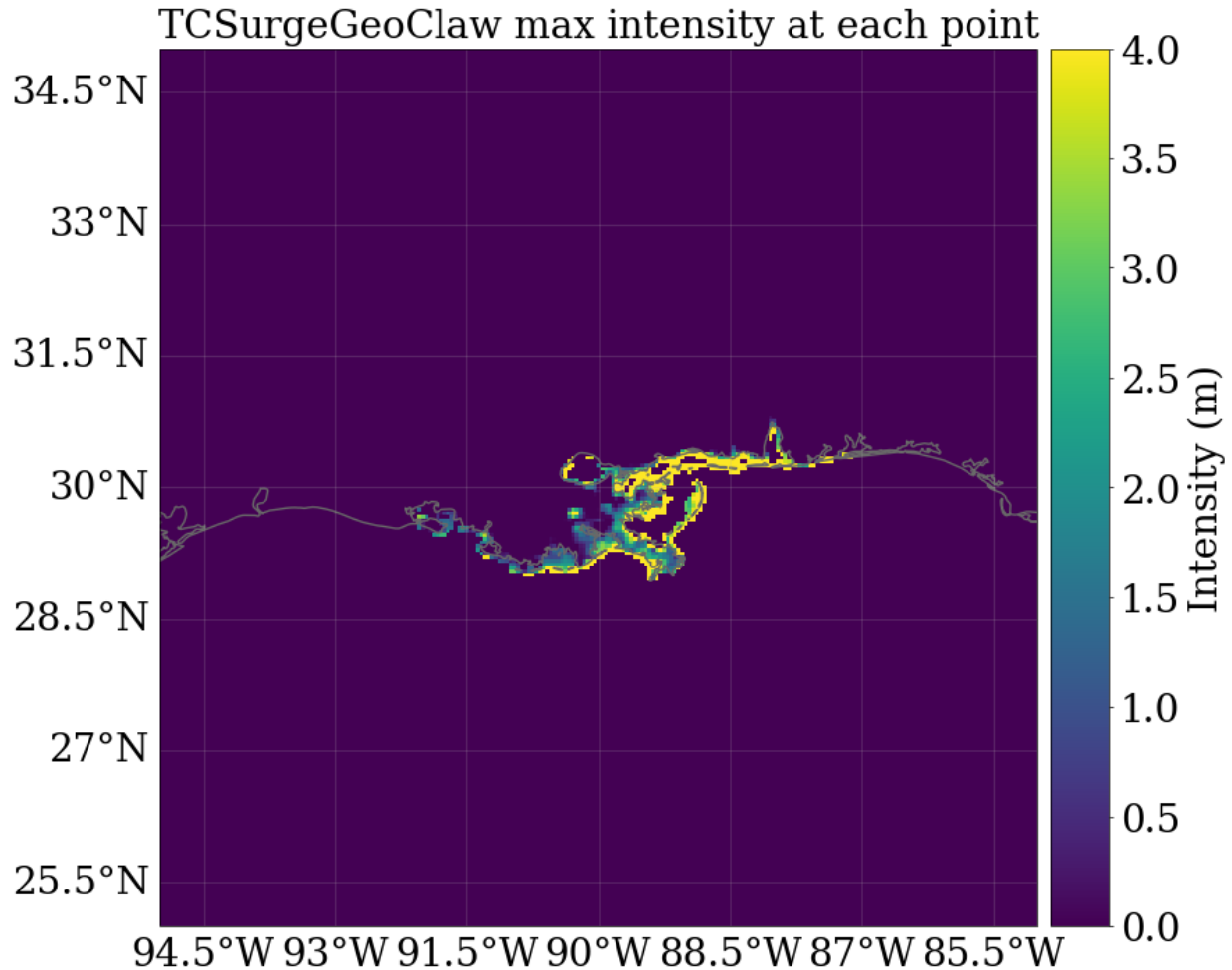
2024-01-31 09:39:18,761 - climada.hazard.base - INFO - Writing tc_surge_katrina_2005.
↳h5

```

```

<GeoAxes: title={'center': 'TCSurgeGeoClaw max intensity at each point'}>

```



3.8 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.

3.8.1 This tutorial describes

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

3.8.2 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`.

```
# 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
```

(continues on next page)

(continued from previous page)

```
# 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

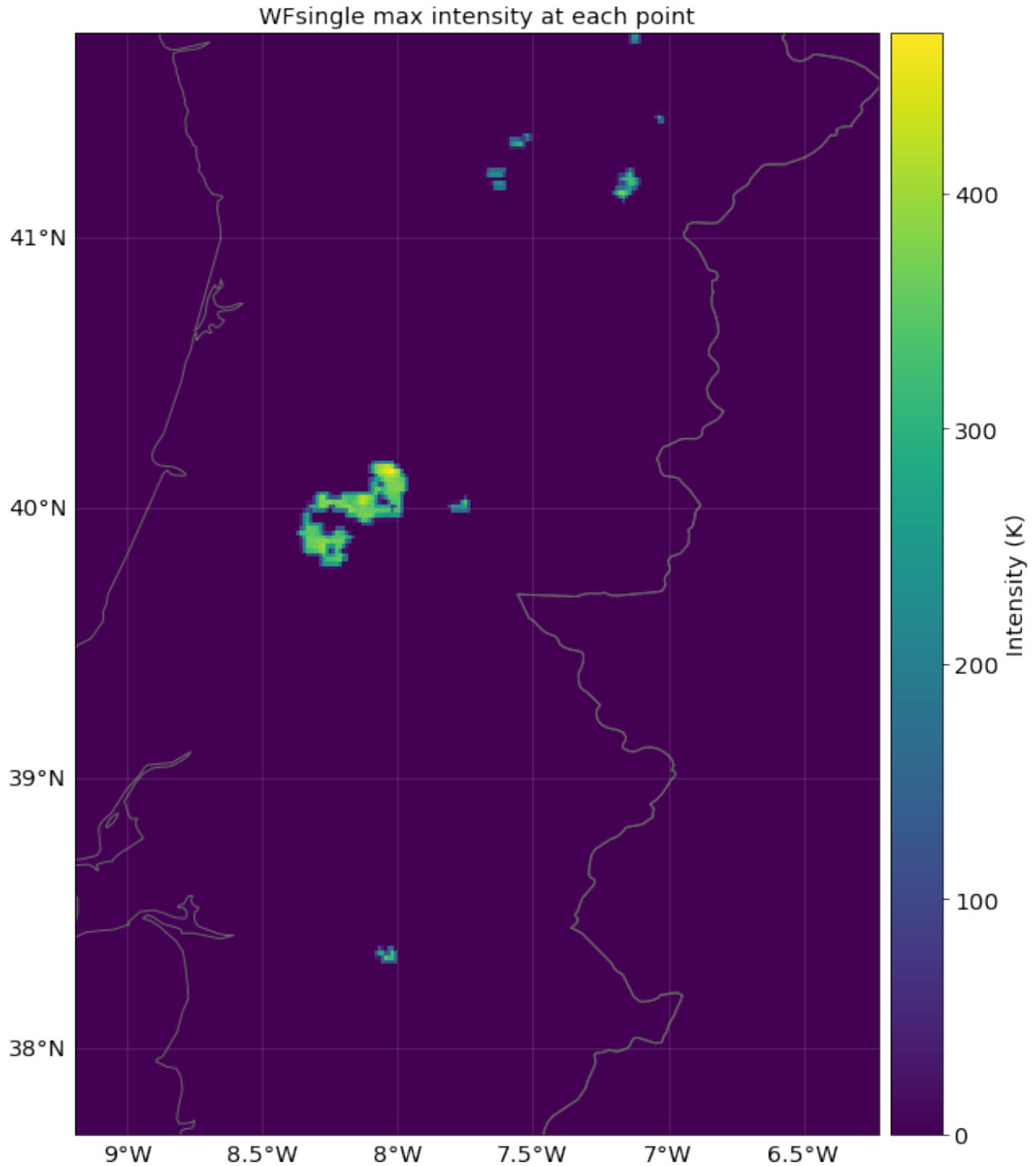
# plot the maximum intensity of all fires at each centroid
wf_pt.plot_intensity(event=0);
```

```
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.centri - 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)
```

```
<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.

```
from climada.hazard import Centroids
from climada.util.constants import ONE_LAT_KM

# focus on Coimbra region with 1.0 km resolution (MODIS resolution)
```

(continues on next page)

(continued from previous page)

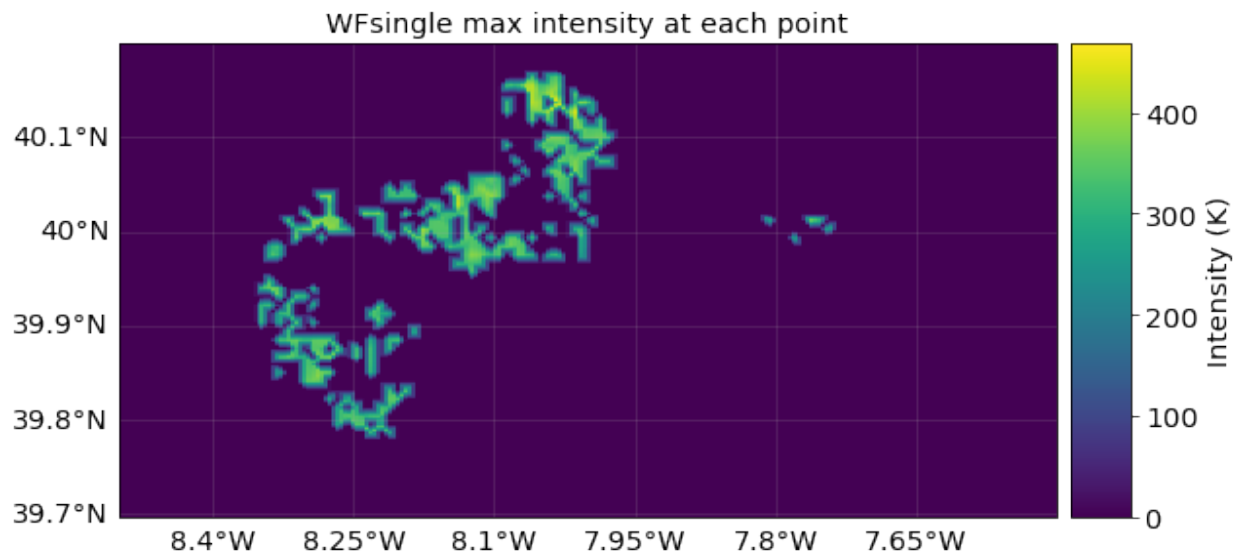
```
res = 1.0/ONE_LAT_KM
centr_zoom = Centroids.from_pnt_bounds((-8.5, 39.7, -7.5, 40.2), res)

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)
```

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



See more infos on [Wikipedia](#) around the fire.

3.8.3 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()`

```
# 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 underlying 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.centri - 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 underlying events is saved as well: [113. 201. 72.]

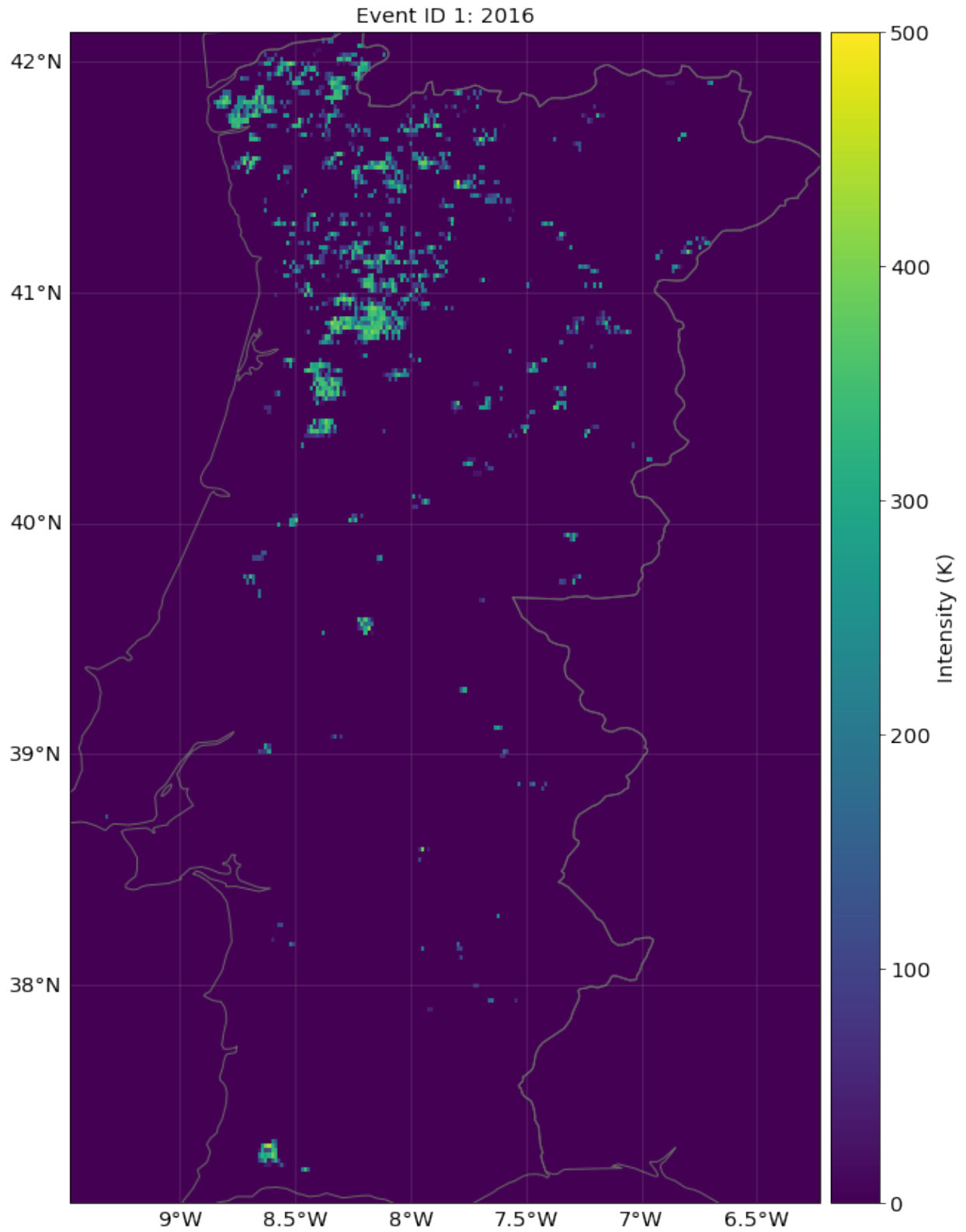
```

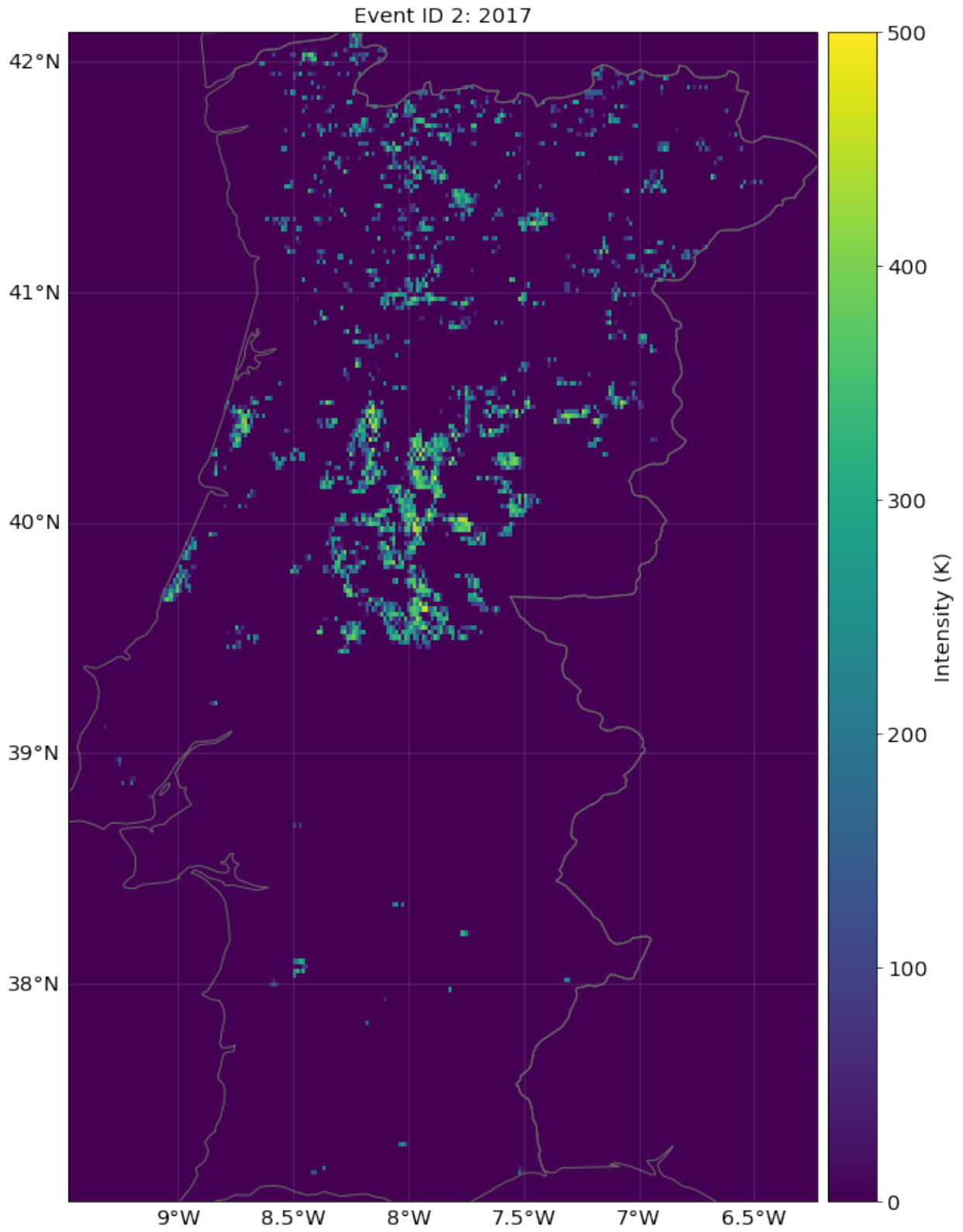
```

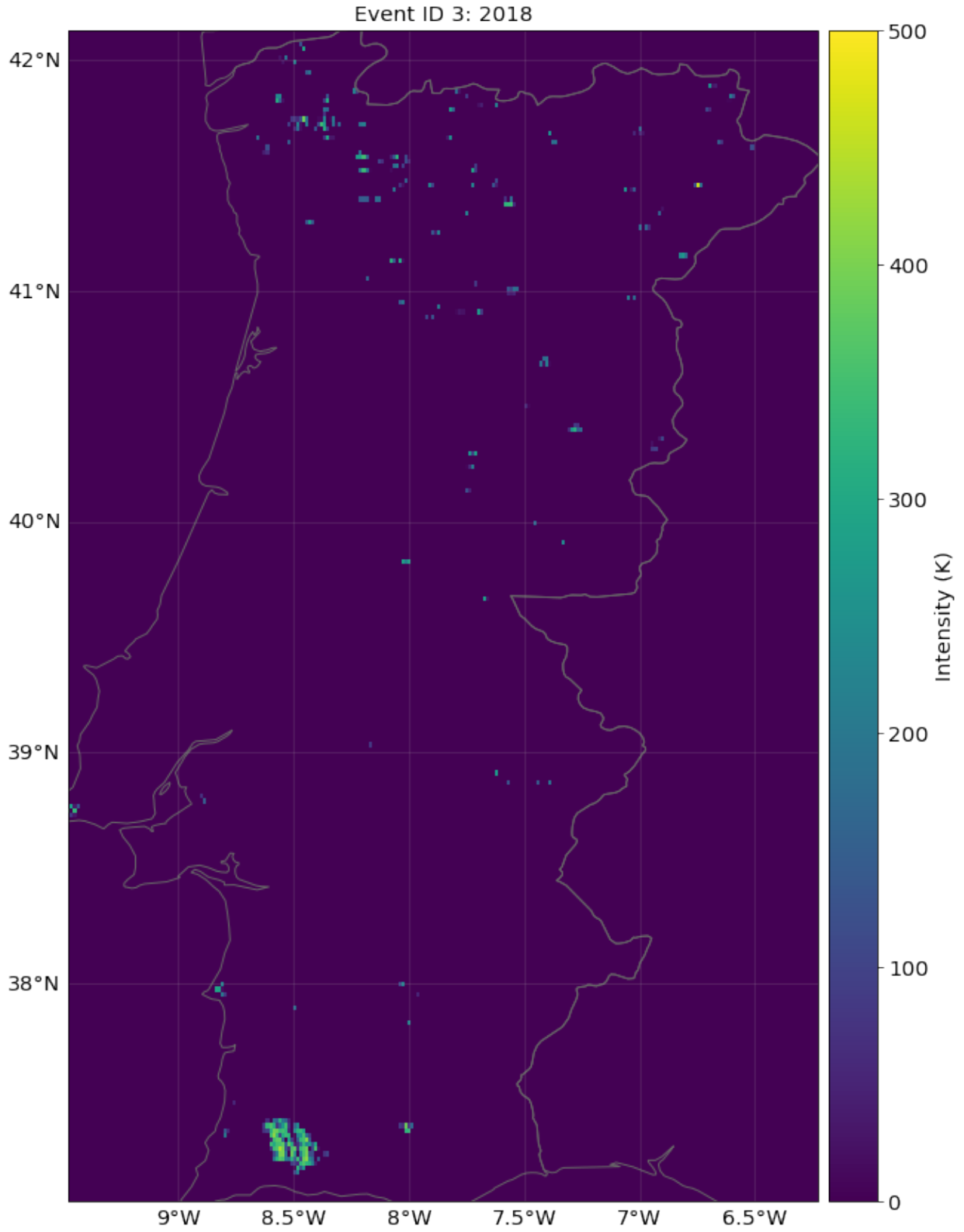
/home/yuyue/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.
↳py:1282: UserWarning: You will likely lose important projection information when
↳converting to a PROJ string from another format. See: https://proj.org/faq.html
↳#what-is-the-best-format-for-describing-coordinate-reference-systems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

```

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







3.8.4 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)

```
# 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
```

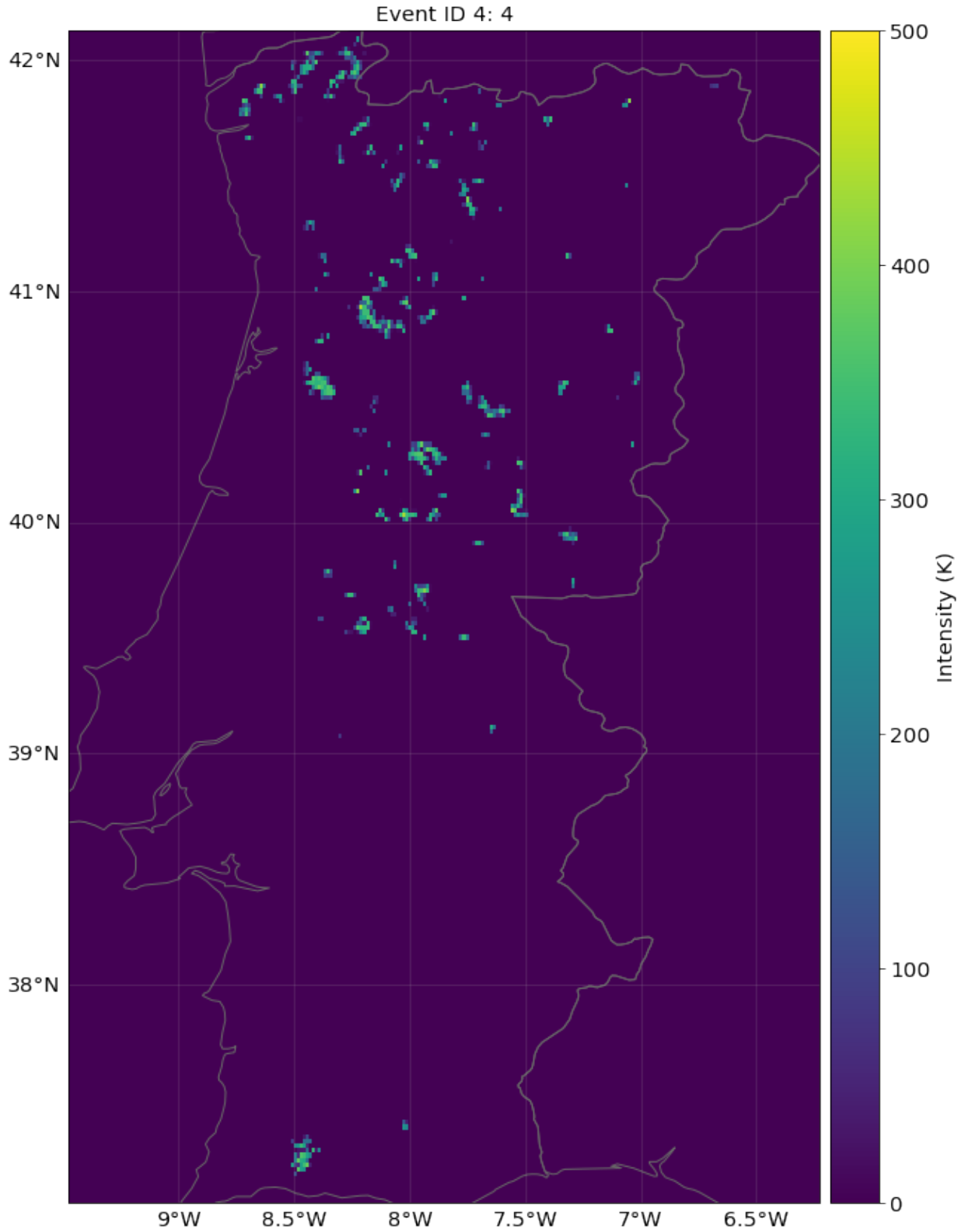
(continues on next page)

(continued from previous page)

```
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
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)
```

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



The overall fire propagation probability is stored as a parameter and can be
 ↪ modified easily (default is 0.21)

(continues on next page)

(continued from previous page)

```

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

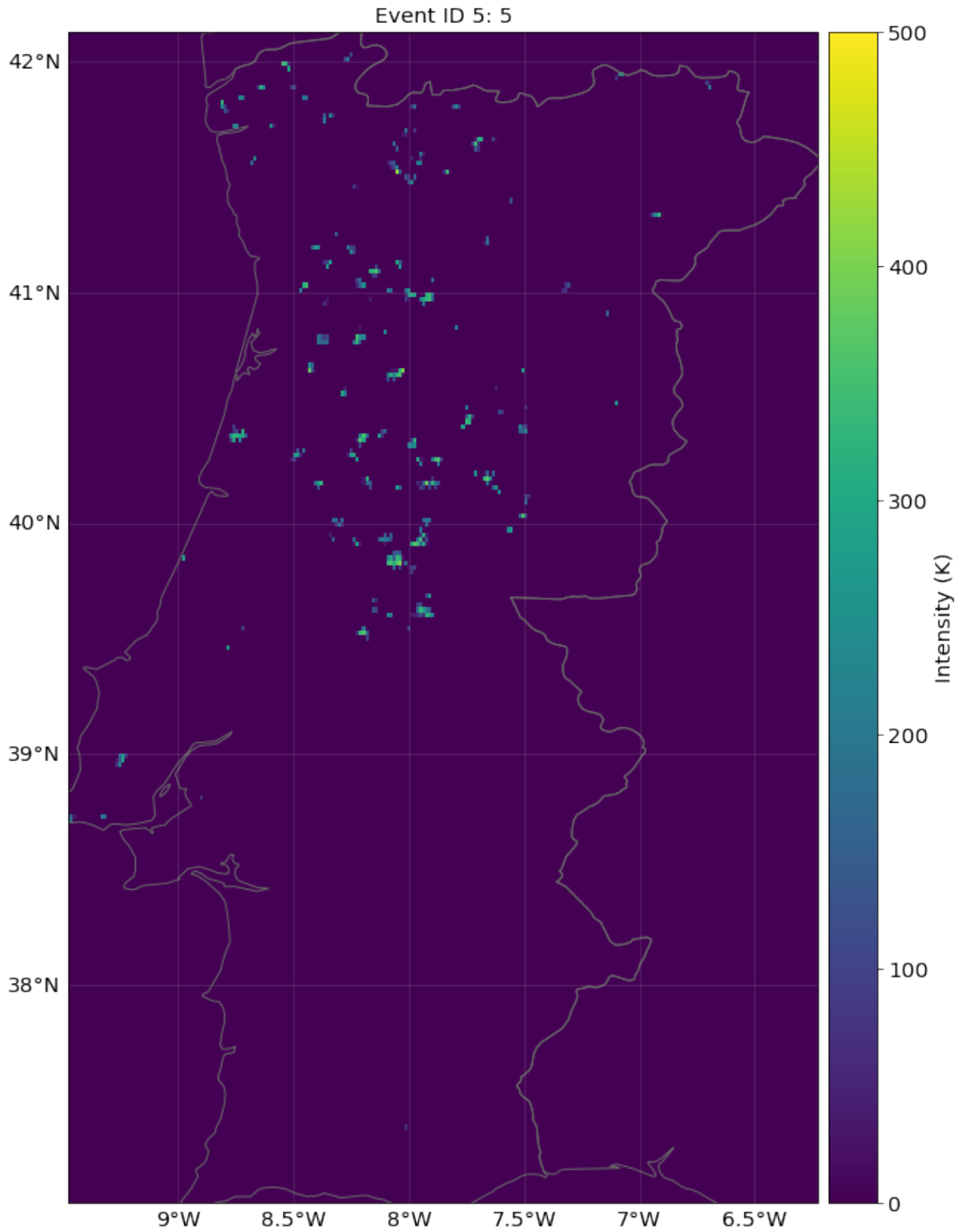
```

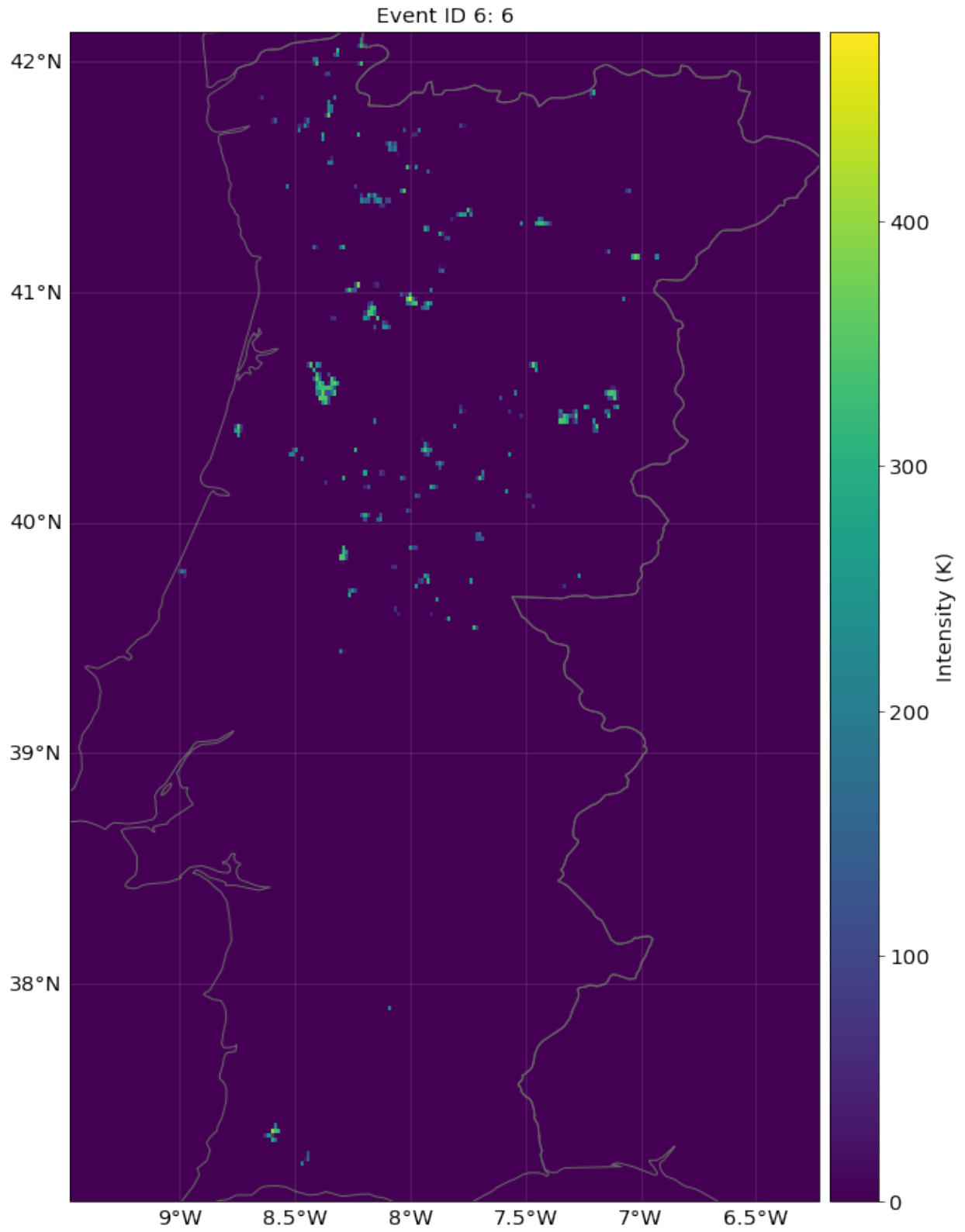
```

/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)

```

<GeoAxesSubplot:title={'center':'Event ID 6: 6'}>





3.8.5 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 vineyard does not glow in the night)

3.9 River Flood based on GloFAS Discharge Data

This tutorial will guide through the process of computing flood footprints from GloFAS river discharge data and using these footprints in CLIMADA impact calculations.

Executing this tutorial requires access to the Copernicus Data Store (CDS) and the input data for the flood footprint pipeline. Please have a look at the documentation of the *GloFAS River Flood module* and follow the instructions in its “Preparation” section before continuing.

The first step is setting up the “static” data required for the river flood model. The `setup_all()` function will download the flood hazard maps, the FLOPROS flood protection database, and the pre-computed Gumbel distribution fit parameters. This might take a while.

```
from climada.util import log_level
from climada_petals.hazard.rf_glofas import setup_all

with log_level("DEBUG", "climada_petals"):
    setup_all()
```

```
/home/roo/miniforge3/envs/climada_flood/lib/python3.9/site-packages/dask/dataframe/_
↳pyarrow_compat.py:17: FutureWarning: Minimal version of pyarrow will soon be
↳increased to 14.0.1. You are using 12.0.1. Please consider upgrading.
warnings.warn(
```

```
2024-02-29 14:54:26,872 - climada_petals.hazard.rf_glofas.setup - DEBUG - Rewriting
↳flood hazard maps to NetCDF files
2024-02-29 14:54:26,873 - climada_petals.hazard.rf_glofas.setup - DEBUG - Merging
↳flood hazard maps into single dataset
2024-02-29 14:58:21,780 - climada_petals.hazard.rf_glofas.setup - DEBUG - Downloading
↳FLOPROS database
2024-02-29 14:58:22,371 - climada_petals.hazard.rf_glofas.setup - DEBUG - Downloading
↳Gumbel fit parameters
```

```
from climada_petals.hazard.rf_glofas import RiverFloodInundation

countries = ["Germany", "Switzerland", "Austria"]

rf = RiverFloodInundation()
rf.download_forecast(
    countries=countries,
    forecast_date="2021-07-10",
```

(continues on next page)

(continued from previous page)

```

preprocess=lambda x: x.max(dim="step"),
system_version="operational", # Version mislabeled
)

ds_flood = rf.compute()
ds_flood

```

```

/home/roo/miniforge3/envs/climada_flood/lib/python3.9/site-packages/xarray/core/
↳dataset.py:282: UserWarning: The specified chunks separate the stored chunks along
↳dimension "latitude" starting at index 1377. This could degrade performance.
↳Instead, consider rechunking after loading.
warnings.warn(
/home/roo/miniforge3/envs/climada_flood/lib/python3.9/site-packages/xarray/core/
↳dataset.py:282: UserWarning: The specified chunks separate the stored chunks along
↳dimension "longitude" starting at index 3480. This could degrade performance.
↳Instead, consider rechunking after loading.
warnings.warn(

```

```

2024-02-29 14:58:26,019 - climada_petals.hazard.rf_glofas.cds_glofas_downloader -
↳INFO - Skipping request for file '/home/roo/climada/data/cds-download/240229-145826-
↳83e84eba8992a4a4d5f8af0a923e2458.grib' because it already exists

```

```

/home/roo/miniforge3/envs/climada_flood/lib/python3.9/site-packages/dask/array/
↳routines.py:325: PerformanceWarning: Increasing number of chunks by factor of 50
intermediate = blockwise(

```

```

<xarray.Dataset>
Dimensions:                (number: 50, time: 1, longitude: 1356, latitude: 1128)
Coordinates:
 * number                   (number) int64 1 2 3 4 5 6 7 8 ... 44 45 46 47 48 49 50
 * time                     (time) datetime64[ns] 2021-07-10
  surface                   float64 ...
  step                      timedelta64[ns] ...
 * longitude                (longitude) float64 5.854 5.863 5.871 ... 17.14 17.15
 * latitude                 (latitude) float64 55.15 55.14 55.13 ... 45.76 45.75
Data variables:
  flood_depth               (latitude, longitude, time, number) float32 dask.array
↳<chunksize=(762, 162, 1, 50), meta=np.ndarray>
  flood_depth_flopros      (latitude, longitude, time, number) float32 dask.array
↳<chunksize=(762, 162, 1, 50), meta=np.ndarray>

```

The returned dataset contains two variables, `flood_depth` and `flood_depth_flopros` which indicate the flood footprints without and with FLOPROS protection levels considered, respectively. Notice that intermediate data is stored in a cache directory by default, but the returned flood data is not. You can use `save_file()` to store it. The cache directory will be deleted as soon as the `RiverFloodInundation` object is.

```

from climada_petals.hazard.rf_glofas import save_file

save_file(ds_flood, "flood-2021-07-10.nc")

```

To use the data within CLIMADA we will have to “translate” it into a Hazard object. We provide a helper function for that: `climada_petals.hazard.rf_glofas.rf_glofas.hazard_series_from_dataset()`. Here we have to

specify the variable to be read as hazard intensity (we choose the one *without* FLOPROS protection levels) and the dimension of the dataset that indicates the “event” dimension of the Hazard object. If the dataset contained more dimensions than `event_dim`, latitude and longitude, the function `hazard_series_from_dataset` would return a pandas Series of Hazard objects indexed with these additional dimensions. We will see this later.

Notice that storing data and reading it from a file might in some cases be faster than loading the data directly into a Hazard object, because the latter requires some potentially costly transpositions of the underlying data structures.

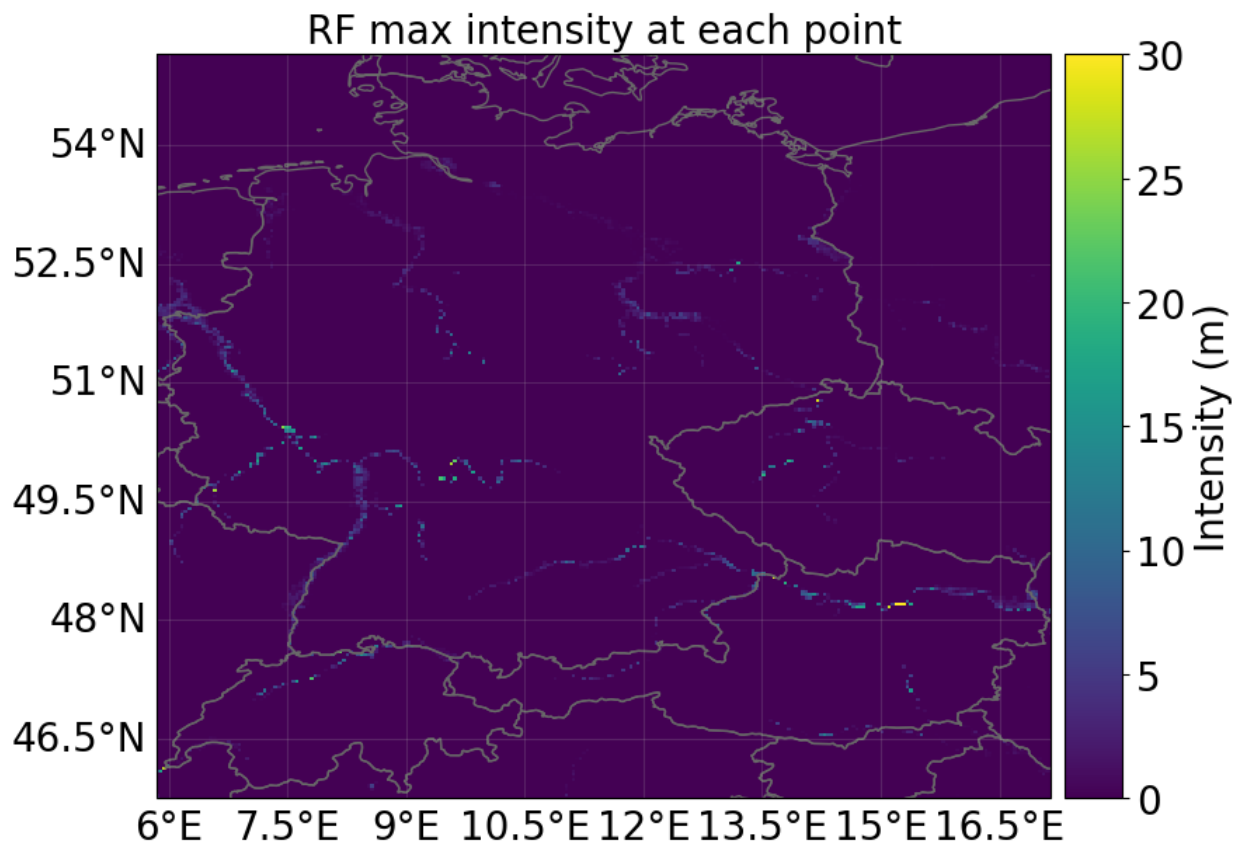
```
from climada_petals.hazard.rf_glofas import hazard_series_from_dataset

hazard = hazard_series_from_dataset(
    ds_flood, intensity="flood_depth", event_dim="number"
)[0]
hazard.plot_intensity(event=0)
```

```
2024-02-29 14:59:26,406 - climada.hazard.base - WARNING - Failed to read values of
↳ 'number' as dates. Hazard.event_name will be empty strings
```

```
/tmp/ipykernel_41108/360336388.py:3: FutureWarning: Series.__getitem__ treating keys
↳ as positions is deprecated. In a future version, integer keys will always be
↳ treated as labels (consistent with DataFrame behavior). To access a value by
↳ position, use `ser.iloc[pos]`
    hazard = hazard_series_from_dataset(
```

```
<GeoAxes: title={'center': 'RF max intensity at each point'}>
```



3.9.1 Exposure and Vulnerability

When looking at flood warnings and protection we typically want to determine how many people might be affected from floods or lose their homes. We therefore use data on population distribution as exposure. To that end, we create a `LitPop` exposure with the `pop` mode for the countries of interest.

Being affected by a flood can be considered a binary classification (yes/no), therefore we use a simple step function with a relatively low threshold of 0.2 m.

Notice that we set the impact function identifier to "RF" because this is the hazard type identifier of `RiverFlood`.

```
from climada.entity import LitPop
from climada.entity import ImpactFunc, ImpactFuncSet

# Create a population exposure
exposure = LitPop.from_population(["Germany", "Switzerland", "Austria"])
exposure.gdf["impf_RF"] = 1

# Create a impact function for being affected by flooding
impf_affected = ImpactFunc.from_step_impf(
    intensity=(0.0, 0.2, 100.0), impf_id=1, haz_type="RF"
)
impf_set_affected = ImpactFuncSet([impf_affected])
```

```
2024-02-29 14:59:53,885 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: DEU (276)...

2024-02-29 14:59:53,935 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:53,936 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2024-02-29 14:59:53,956 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:53,957 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2024-02-29 14:59:53,982 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:53,983 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2024-02-29 14:59:54,006 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:54,007 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2024-02-29 14:59:54,031 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:54,032 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2024-02-29 14:59:54,058 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:54,059 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2024-02-29 14:59:54,084 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:54,085 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
```

(continues on next page)

(continued from previous page)

```
2024-02-29 14:59:54,104 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:54,105 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:54,134 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:54,135 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:54,151 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:54,151 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:54,185 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:54,186 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:56,052 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,053 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:56,119 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,120 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:56,194 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,195 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:56,257 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,258 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:56,339 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,340 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:56,411 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,412 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:56,478 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,479 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:56,541 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,542 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
2024-02-29 14:59:56,614 - climada.entity.exposures.litpop.gpw_population - WARNING ->
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,615 - climada.entity.exposures.litpop.gpw_population - INFO - GPW->
↳Version v4.11
```

(continues on next page)

(continued from previous page)

```

2024-02-29 14:59:56,685 - climada.entity.exposures.litpop.gpw_population - WARNING - ↵
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2024-02-29 14:59:56,687 - climada.entity.exposures.litpop.gpw_population - INFO - GPW↵
↳Version v4.11
2024-02-29 14:59:58,417 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: CHE (756)...
2024-02-29 15:00:00,409 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: AUT (40)...
2024-02-29 15:00:01,222 - climada.entity.exposures.base - INFO - Hazard type not set↵
↳in impf_
2024-02-29 15:00:01,223 - climada.entity.exposures.base - INFO - category_id not set.
2024-02-29 15:00:01,224 - climada.entity.exposures.base - INFO - cover not set.
2024-02-29 15:00:01,224 - climada.entity.exposures.base - INFO - deductible not set.
2024-02-29 15:00:01,225 - climada.entity.exposures.base - INFO - centr_ not set.

```

3.9.2 Impact Calculation

We simply plug everything together and calculate the impact. Afterwards, we plot the average impact for 14 July 2021.

The averaging function takes the `frequency` parameter of the hazard into account. By default, all hazard events have a frequency of 1. This may result in unexpected values for average impacts. We therefore divide the frequency by the number of events before plotting.

```

import numpy as np
from climada.engine import ImpactCalc

hazard.frequency = hazard.frequency / hazard.size
impact = ImpactCalc(exposure, impf_set_affected, hazard).impact()
impact.plot_hexbin_eai_exposure(gridsize=100, lw=0)

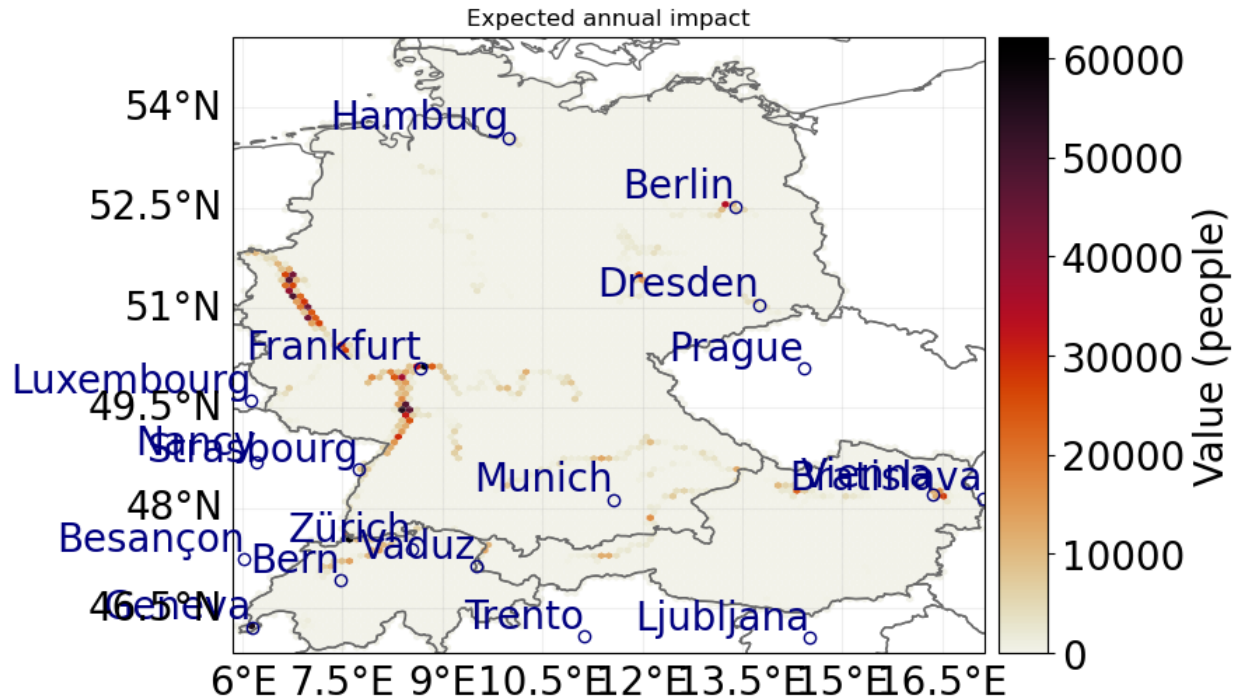
```

```

2024-02-29 15:00:01,268 - climada.entity.exposures.base - INFO - Matching 876313↵
↳exposures with 1529568 centroids.
2024-02-29 15:00:01,281 - climada.util.coordinates - INFO - No exact centroid match↵
↳found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2024-02-29 15:00:03,547 - climada.engine.impact_calc - INFO - Calculating impact for↵
↳2606616 assets (>0) and 50 events.

```

```
<GeoAxes: title={'center': 'Expected annual impact'}>
```



3.9.3 FLOPROS Database Protection Standards

So far, we ignored any protection standards, likely overestimating the impact of events. The FLOPROS database provides information on river flood protection standards. It is a supplement to the publication by P. Scussolini et al.: “FLOPROS: an evolving global database of flood protection standards” It was automatically loaded when you called `setup_all()`.

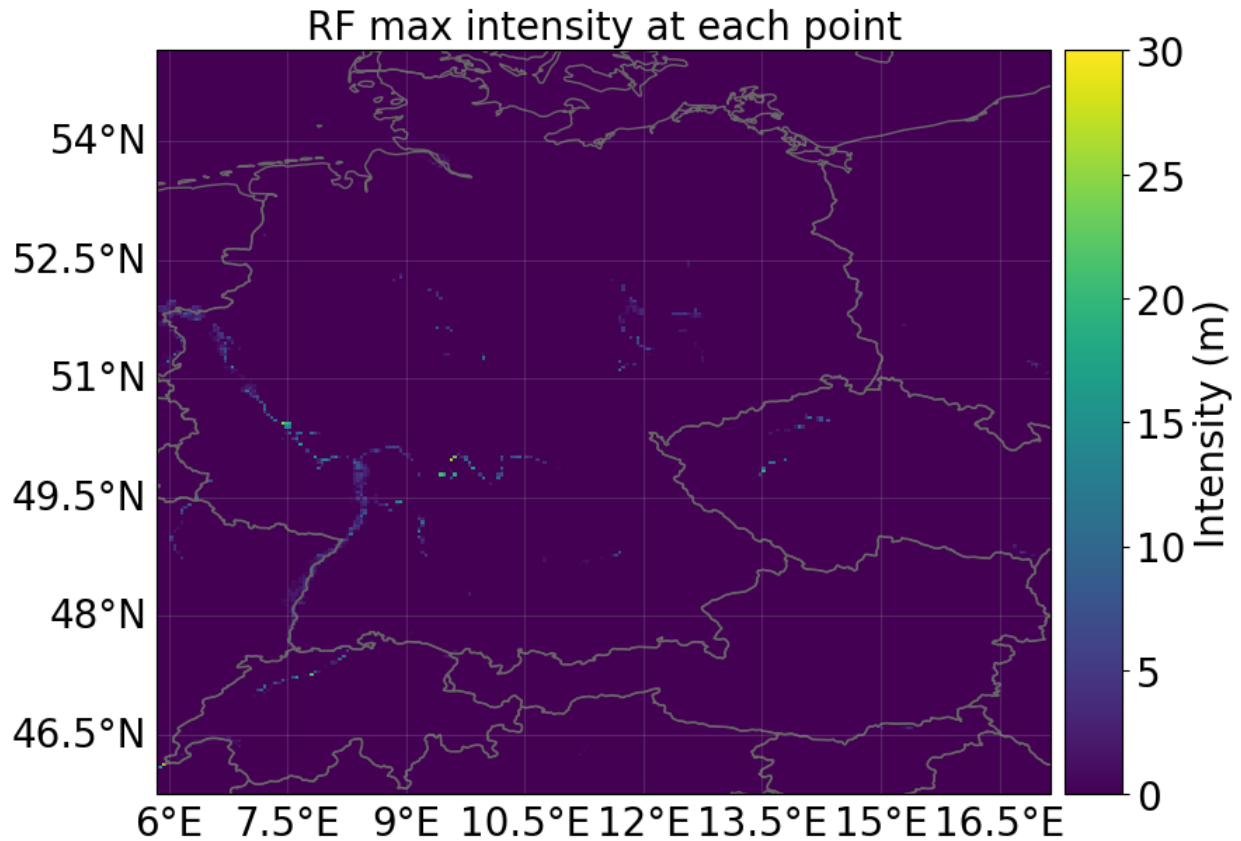
Let’s compare the hazard and corresponding impacts when considering the FLOPROS protection standards!

```
hazard_flopros = hazard_series_from_dataset(
    ds_flood, intensity="flood_depth_flopros", event_dim="number"
)[0]
hazard_flopros.plot_intensity(event=0)
```

```
2024-02-29 15:00:09,411 - climada.hazard.base - WARNING - Failed to read values of
↳ 'number' as dates. Hazard.event_name will be empty strings
```

```
/tmp/ipykernel_41108/1832447213.py:1: FutureWarning: Series.__getitem__ treating keys
↳ as positions is deprecated. In a future version, integer keys will always be
↳ treated as labels (consistent with DataFrame behavior). To access a value by
↳ position, use `ser.iloc[pos]`
    hazard_flopros = hazard_series_from_dataset(
```

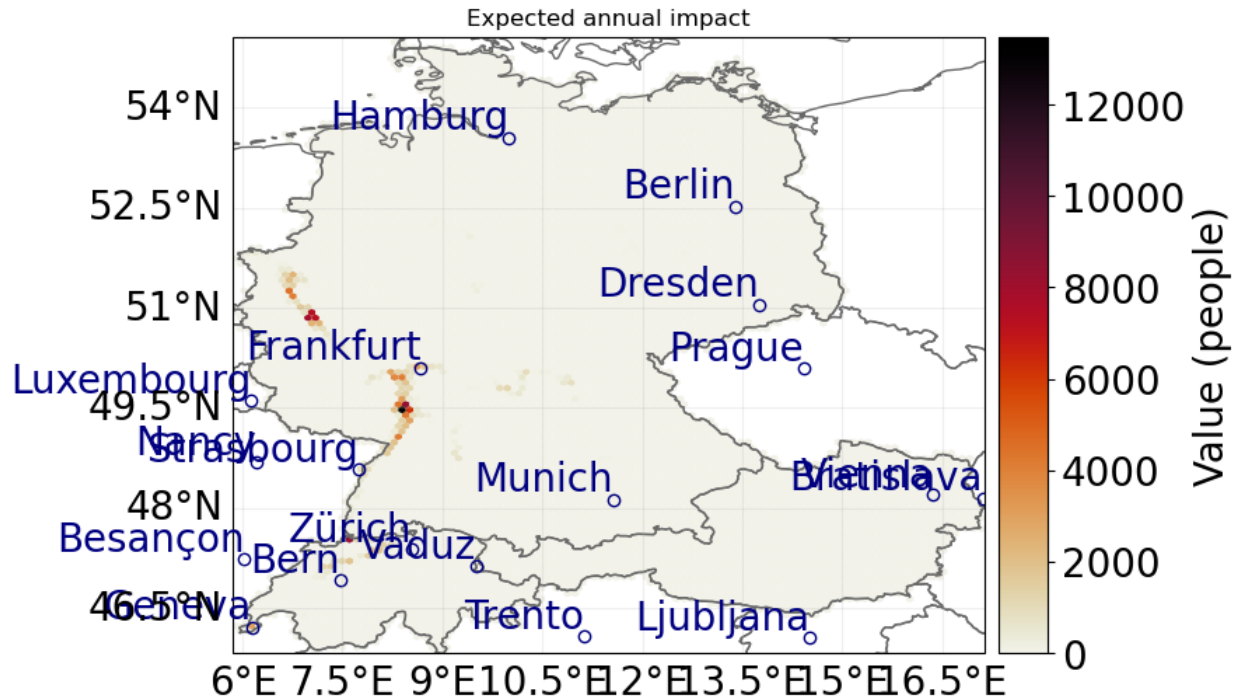
```
<GeoAxes: title={'center': 'RF max intensity at each point'}>
```



```
hazard_flopros.frequency = hazard_flopros.frequency / hazard_flopros.size
impact_flopros = ImpactCalc(exposure, impf_set_affected, hazard_flopros).impact()
impact_flopros.plot_hexbin_eai_exposure(gridsize=100, lw=0)
```

```
2024-02-29 15:00:37,496 - climada.entity.exposures.base - INFO - Exposures matching
↳centroids already found for RF
2024-02-29 15:00:37,497 - climada.entity.exposures.base - INFO - Existing centroids
↳will be overwritten for RF
2024-02-29 15:00:37,498 - climada.entity.exposures.base - INFO - Matching 876313
↳exposures with 1529568 centroids.
2024-02-29 15:00:37,514 - climada.util.coordinates - INFO - No exact centroid match
↳found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2024-02-29 15:00:39,504 - climada.engine.impact_calc - INFO - Calculating impact for
↳2606616 assets (>0) and 50 events.
```

```
<GeoAxes: title={'center': 'Expected annual impact'}>
```



```
import pandas as pd
import matplotlib.pyplot as plt

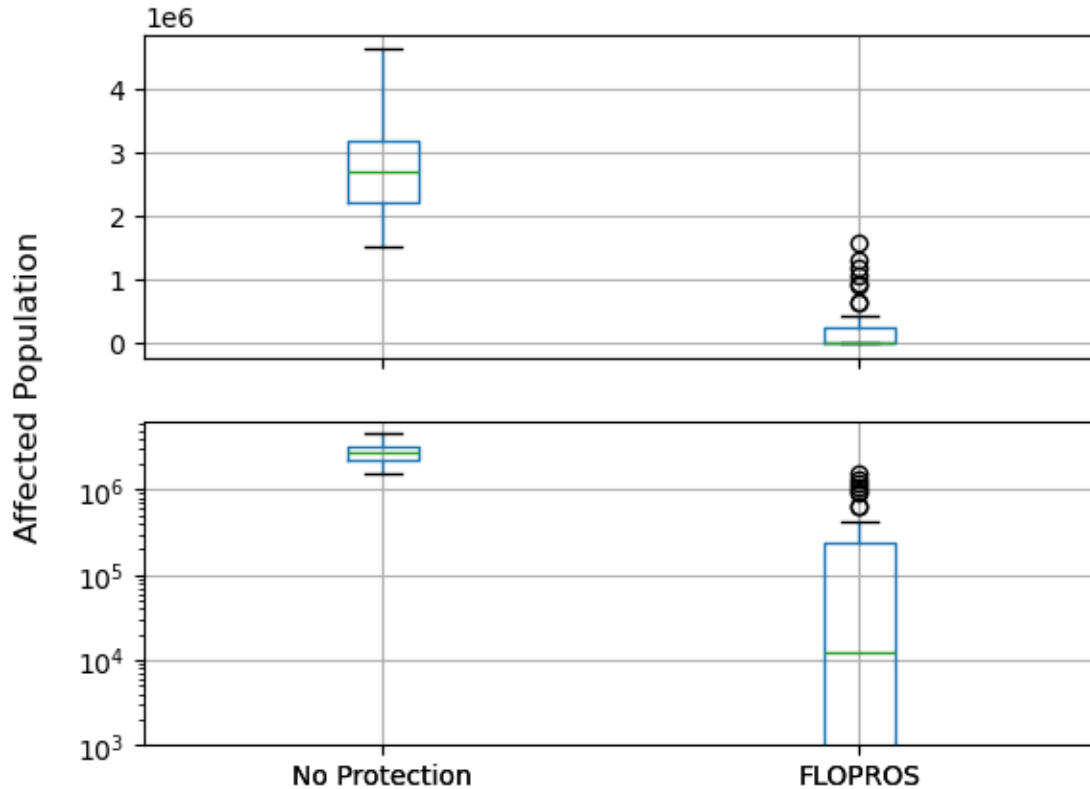
df_impact = pd.DataFrame.from_dict(
    data={"No Protection": impact.at_event, "FLOPROS": impact_flopros.at_event}
)

fig, axes = plt.subplots(2, 1, sharex=True)

for ax in axes:
    df_impact.boxplot(ax=ax)

axes[1].set_yscale("log")
axes[1].set_ylim(bottom=1e3)
fig.supylabel("Affected Population")
```

```
Text(0.02, 0.5, 'Affected Population')
```



3.9.4 Multidimensional Data

We computed the risk of population being affected by 0.2 m of river flooding from the forecast issued on 2021-07-10 over the next 10 days. We reduced the dimensionality of the data by simply taking the maximum discharge over the lead time of 10 days before computing the flood footprint. We can get a clearer picture of flood timings by defining sub-selection of the lead time and computing footprints for each of these. This is easy to do because we receive the downloaded discharge as xarray DataArray and can manipulate it however we like before putting it into the computation pipeline. However, increasing the amount of flood footprints to compute will also increase the computational cost.

```
discharge = rf.download_forecast(
    countries=countries,
    forecast_date="2021-07-10",
    system_version="operational", # Version mislabeled
)
discharge
```

```
2024-02-29 15:00:44,793 - climada_petals.hazard.rf_glofas.cds_glofas_downloader -
↳ INFO - Skipping request for file '/home/roo/climada/data/cds-download/240229-150044-
↳ 83e84eba8992a4a4d5f8af0a923e2458.grib' because it already exists
```

```
<xarray.DataArray 'dis24' (time: 1, number: 50, step: 10, latitude: 95,
                           longitude: 114)>
dask.array<broadcast_to, shape=(1, 50, 10, 95, 114), dtype=float32, chunksize=(1, 50,
↳ 10, 95, 114), chunktype=numpy.ndarray>
Coordinates:
  * number      (number) int64 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
```

(continues on next page)

(continued from previous page)

```

* time          (time) datetime64[ns] 2021-07-10
* step          (step) timedelta64[ns] 1 days 2 days 3 days ... 9 days 10 days
  surface       float64 ...
* latitude      (latitude) float64 55.15 55.05 54.95 54.85 ... 45.95 45.85 45.75
* longitude     (longitude) float64 5.85 5.95 6.05 6.15 ... 16.95 17.05 17.15
  valid_time    (step) datetime64[ns] dask.array<chunksize=(10,), meta=np.ndarray>
Attributes: (12/30)
  GRIB_paramId:                240024
  GRIB_dataType:                pf
  GRIB_numberOfPoints:         10830
  GRIB_typeOfLevel:            surface
  GRIB_stepUnits:              1
  GRIB_stepType:               avg
  ...                          ...
  GRIB_shortName:              dis24
  GRIB_totalNumber:            51
  GRIB_units:                  m**3 s**-1
  long_name:                    Mean discharge in the last 24 h...
  units:                        m**3 s**-1
  standard_name:                unknown

```

```

import xarray as xr

discharge_tf = xr.concat(
    [
        discharge.isel(step=slice(0, 2)).max(dim="step"),
        discharge.isel(step=slice(2, 5)).max(dim="step"),
        discharge.isel(step=slice(5, 10)).max(dim="step"),
    ],
    dim=pd.Index(["1-2 Days", "3-5 Days", "6-10 Days"], name="Time Frame"),
)
discharge_tf

```

```

<xarray.DataArray 'dis24' (Time Frame: 3, time: 1, number: 50, latitude: 95,
                          longitude: 114)>
dask.array<concatenate, shape=(3, 1, 50, 95, 114), dtype=float32, chunksize=(1, 1, 50,
↪ 95, 114), chunktype=numpy.ndarray>
Coordinates:
  * number      (number) int64 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * time        (time) datetime64[ns] 2021-07-10
  surface       float64 ...
  * latitude    (latitude) float64 55.15 55.05 54.95 54.85 ... 45.95 45.85 45.75
  * longitude   (longitude) float64 5.85 5.95 6.05 6.15 ... 16.95 17.05 17.15
  * Time Frame  (Time Frame) object '1-2 Days' '3-5 Days' '6-10 Days'

```

First, we have to clear the cached files to make sure the `RiverFloodInundation` object is not operating on the last computed results. Then, we call `compute` again, this time with a custom data array inserted as `discharge`. As additional arguments we add `reuse_regridder=True` to the parameters of `regrid`, which will instruct the object to save time by reusing the regridding matrix. This is possible because all computations in this tutorial operate on the same grid.

```

rf.clear_cache()
ds_flood_tf = rf.compute(discharge=discharge_tf, regrid_kws=dict(reuse_

```

(continues on next page)

(continued from previous page)

```

↪regridder=True))
save_file(ds_flood_tf, "flood-2021-07-10-tf.nc")
ds_flood_tf

```

```

/home/roo/miniforge3/envs/climada_flood/lib/python3.9/site-packages/dask/array/
↪routines.py:325: PerformanceWarning: Increasing number of chunks by factor of 50
intermediate = blockwise(

```

```

<xarray.Dataset>
Dimensions:          (number: 50, time: 1, Time Frame: 3, longitude: 1356,
                    latitude: 1128)
Coordinates:
  * number           (number) int64 1 2 3 4 5 6 7 8 ... 44 45 46 47 48 49 50
  * time             (time) datetime64[ns] 2021-07-10
    surface          float64 ...
  * Time Frame       (Time Frame) <U9 '1-2 Days' '3-5 Days' '6-10 Days'
    step             timedelta64[ns] ...
  * longitude        (longitude) float64 5.854 5.863 5.871 ... 17.14 17.15
  * latitude         (latitude) float64 55.15 55.14 55.13 ... 45.76 45.75
Data variables:
  flood_depth        (latitude, longitude, Time Frame, time, number) float32 dask.
↪array<chunks=(762, 162, 3, 1, 50), meta=np.ndarray>
  flood_depth_flopros (latitude, longitude, Time Frame, time, number) float32 dask.
↪array<chunks=(762, 162, 3, 1, 50), meta=np.ndarray>

```

```

ds_flood_tf.close()

```

Since the dataset contains more dimensions than `latitude`, `longitude`, and the dimension specified as `event_dim` (here: `number`), the function `hazard_series_from_dataset` will return a pandas Series of Hazard objects, with the remaining dimensions as (possibly multidimensional) index.

```

with xr.open_dataset("flood-2021-07-10-tf.nc", chunks="auto") as ds:
    hazard_series = hazard_series_from_dataset(ds, "flood_depth", "number")
    hazard_series_flopros = hazard_series_from_dataset(
        ds, "flood_depth_flopros", "number"
    )
hazard_series

```

```

2024-02-29 15:01:35,407 - climada.hazard.base - WARNING - Failed to read values of
↪'number' as dates. Hazard.event_name will be empty strings
2024-02-29 15:01:37,239 - climada.hazard.base - WARNING - Failed to read values of
↪'number' as dates. Hazard.event_name will be empty strings
2024-02-29 15:01:38,239 - climada.hazard.base - WARNING - Failed to read values of
↪'number' as dates. Hazard.event_name will be empty strings
2024-02-29 15:01:39,994 - climada.hazard.base - WARNING - Failed to read values of
↪'number' as dates. Hazard.event_name will be empty strings
2024-02-29 15:01:40,862 - climada.hazard.base - WARNING - Failed to read values of
↪'number' as dates. Hazard.event_name will be empty strings
2024-02-29 15:01:41,792 - climada.hazard.base - WARNING - Failed to read values of
↪'number' as dates. Hazard.event_name will be empty strings

```

```

time          Time Frame
2021-07-10   1-2 Days      <climada.hazard.base.Hazard object at 0x7f10f2...
              3-5 Days      <climada.hazard.base.Hazard object at 0x7f10f3...
              6-10 Days     <climada.hazard.base.Hazard object at 0x7f10f3...
dtype: object

```

We compute the impacts for all time frames of both series and visualize the resulting distributions with boxplots.

```

impact_series = pd.Series(
    [
        ImpactCalc(exposure, impf_set_affected, haz).impact(assign_centroids=False)
        for _, haz in hazard_series.items()
    ],
    index=hazard_series.index,
)
impact_series_flopros = pd.Series(
    [
        ImpactCalc(exposure, impf_set_affected, haz).impact(assign_centroids=False)
        for _, haz in hazard_series_flopros.items()
    ],
    index=hazard_series_flopros.index,
)

```

```

2024-02-29 15:01:41,831 - climada.engine.impact_calc - INFO - Calculating impact for
↳2606616 assets (>0) and 50 events.
2024-02-29 15:01:41,991 - climada.engine.impact_calc - INFO - Calculating impact for
↳2606616 assets (>0) and 50 events.
2024-02-29 15:01:42,161 - climada.engine.impact_calc - INFO - Calculating impact for
↳2606616 assets (>0) and 50 events.
2024-02-29 15:01:42,339 - climada.engine.impact_calc - INFO - Calculating impact for
↳2606616 assets (>0) and 50 events.
2024-02-29 15:01:42,434 - climada.engine.impact_calc - INFO - Calculating impact for
↳2606616 assets (>0) and 50 events.
2024-02-29 15:01:42,541 - climada.engine.impact_calc - INFO - Calculating impact for
↳2606616 assets (>0) and 50 events.

```

```

df_impacts = pd.concat(
    [
        pd.DataFrame.from_records(
            {str_idx: imp.at_event for idx, imp in impact_series.items() for str_idx
↳in idx if isinstance(str_idx, str)}
            | {"Protection": "No Protection"}
        ),
        pd.DataFrame.from_records(
            {str_idx: imp.at_event for idx, imp in impact_series_flopros.items() for
↳str_idx in idx if isinstance(str_idx, str)}
            | {"Protection": "FLOPROS"}
        ),
    ]
)
axes = df_impacts.boxplot(
    column=["1-2 Days", "3-5 Days", "6-10 Days"],

```

(continues on next page)

(continued from previous page)

```

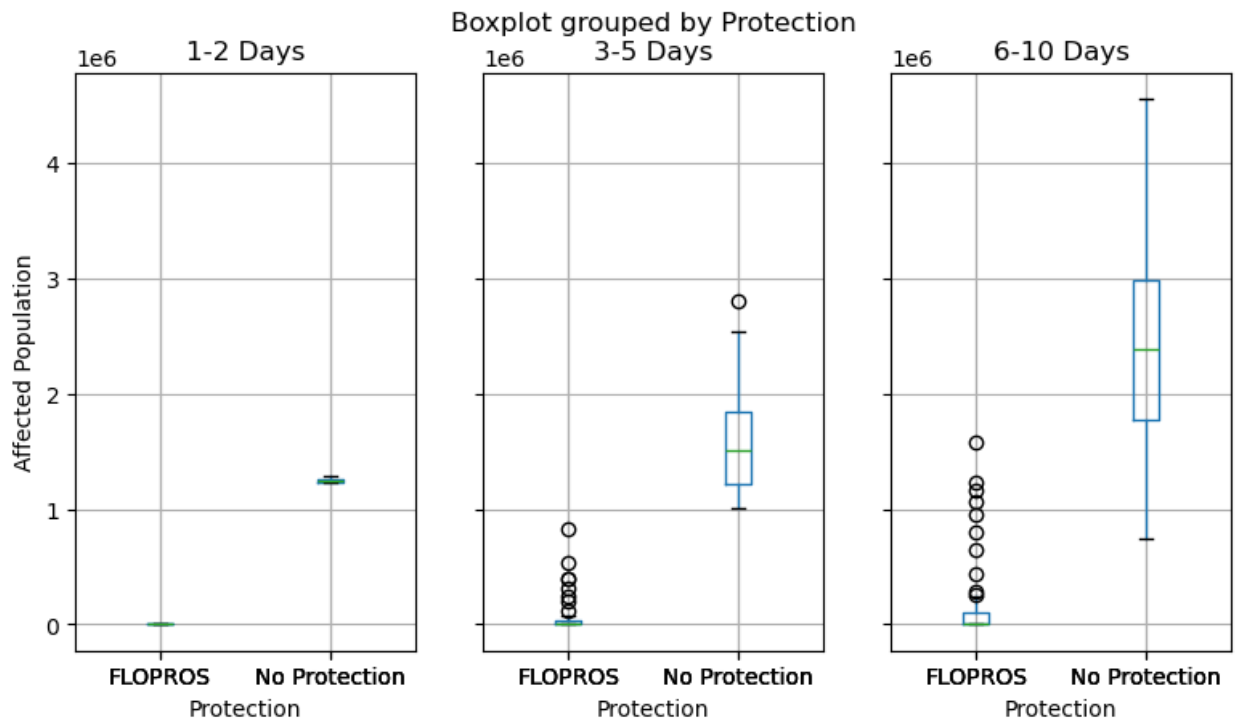
by="Protection",
figsize=(9, 4.8),
layout=(1, 3),
)
axes[0].set_ylabel("Affected Population")

```

```

Text(0, 0.5, 'Affected Population')

```



```

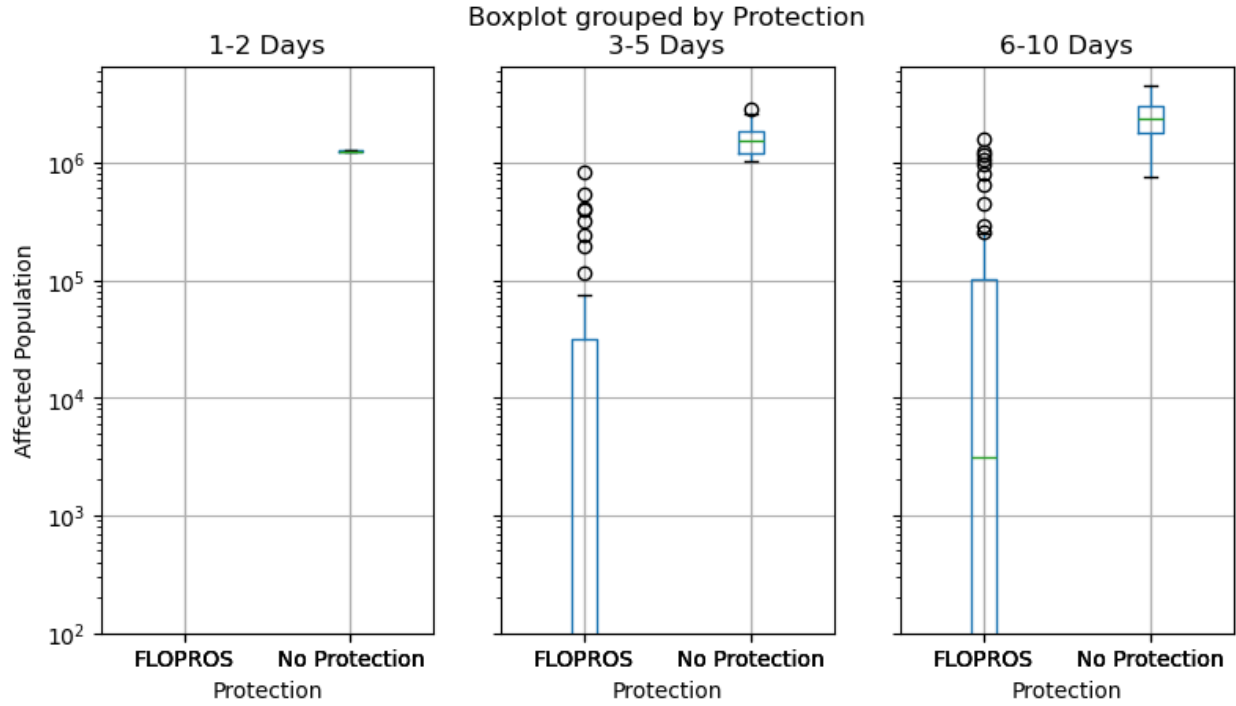
axes = df_impacts.boxplot(
    column=["1-2 Days", "3-5 Days", "6-10 Days"],
    by="Protection",
    figsize=(9, 4.8),
    layout=(1, 3),
)
axes[0].set_ylabel("Affected Population")
axes[0].set_yscale("log")
axes[0].set_ylim(bottom=1e2)

```

```

(100.0, 6561677.47926394)

```



4.1 Introduction

This tutorial shows how to use the `supplychain` module of CLIMADA. This class allows assessing indirect economic impacts via Input-Output (IO) based modeling.

This tutorial assumes you are familiar with direct impact computation with CLIMADA with the `Exposures`, `Hazard` and `Impact` classes. Likewise, this tutorial assumes you are familiar with IO based modeling¹.

To avoid an unnecessarily dense tutorial we cover the basic functionality, but a lot of methods and functions presented here offer some advanced uses, for which you can find documentation via the API reference and their docstring.

4.2 Goal of this tutorial

The goal of this tutorial is to present **indirect** impact computations for the different available approaches, by presenting how to set up a global supply chain risk analysis for tropical cyclones hitting Japan.

4.3 What approaches are available?

Here, we **briefly** describe the available approaches. We **strongly** advise you to find more detailed documentation online or in the literature if you are not familiar with these concepts.

4.3.1 Leontief

The Leontief approach to conducting an indirect impact assessment involves using a Multi-Regional Input Output Table (MRIOT) to quantify the ripple effects of a **change in final demand** throughout the economy. The key steps are:

1. Obtain the Leontief inverse matrix from the intermediate demand matrix of the MRIOT (or transaction matrix), which represents the total (direct and indirect) requirements of each sector per unit of final demand. This captures the interdependencies between different sectors.
2. Multiply the Leontief inverse matrix by the change in final demand (the “exogenous variable”) to calculate the indirect impacts on production in each sector.
3. The indirect effects analysis assesses the secondary needs arising from interactions among different sectors, considering the infinite iterations within the production system.
4. This allows for a comprehensive assessment of the indirect impacts beyond the first-order, direct impacts of the change in final demand.

¹ We recommend the following book: Miller, R. E., & Blair, P. D. (2009). Input-Output Analysis: Foundations and Extensions (2nd ed.). Cambridge: Cambridge University Press.

4.3.2 Ghosh

The Ghosh approach to conducting an indirect impact assessment is similar to the Leontief approach, but with some key differences:

1. The Ghosh model focuses on the supply-side of the economy, rather than the demand-side as in the Leontief model. It analyzes the impacts of changes in primary inputs (most often the value-added, which is what we use in the module) on the output of each sector.
2. The Ghosh inverse matrix represents the total (direct and indirect) output requirements per unit of primary input, capturing the forward linkages in the production system.
3. To assess the indirect impacts, the Ghosh inverse matrix is multiplied by the change in primary inputs (the “exogenous variable”) to calculate the resulting changes in sectoral outputs.

Note however, that the Ghosh model has been criticized for its lack of economic plausibility, as it assumes that consumption is unresponsive to changes in income. Which is considered an unrealistic assumption.

4.3.3 ARIO (with the `boario` package)

ARIO stands for Adaptive Regional Input-Output. It is an hybrid input-output agent-based dynamic economic model, designed to compute indirect costs from economic shocks. Its first version dates back to 2008 and has originally been developed to assess the indirect costs of natural disasters [Hallegatte 2008, Hallegatte 2013].

CLIMADA employs the `boario` python package which implements the ARIO model.

Here are its keys elements:

- The economy is modelled as a set of economic sectors and a set of regions (the initial equilibrium state of the economy is built based on a MRIOT).
- Each economic sector produces its generic product and draws inputs from an inventory.
- Each sector answers to a total demand consisting of a final demand (household consumption, public spending and private investments) of all regions (local demand and exports) and intermediate demand (through input inventory resupply).

The model then describes how exogenous shocks propagate across the economy at each time step.

4.4 Tutorial

```
import numpy as np
import pandas as pd

from climada.engine.impact_calc import ImpactCalc
from climada.entity import ImpactFuncSet, ImpfTropCyclone
from climada.util.api_client import Client
from climada_petals.engine.supplychain import DirectShocksSet, StaticIOModel, \
↳BoARIOModel, get_mriot
import datetime as dt

client = Client()
```

```
ERROR 1: PROJ: proj_create_from_database: Open of /home/sjuhel/miniforge3/envs/
↳supplychaindev/share/proj failed
```

4.4.1 Calculate direct economic impacts

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

```
exp_jpn = client.get_litpop('JPN')
```

```
2025-05-23 14:04:54,479 - climada.entity.exposures.base - INFO - Reading /home/sjuhel/
↳climada/data/exposures/litpop/LitPop_150arcsec_JPN/v3/LitPop_150arcsec_JPN.hdf5
```

Then, we load tropical cyclones that hit the Japan in 2019 from the CLIMADA Data API.

```
tc_jpn = client.get_hazard('tropical_cyclone', properties={'country_iso3alpha':'JPN',
↳'event_type': 'observed'})
```

```
target_year = 2019
events_in_target_year = np.array([
    tc_jpn.event_name[i] for i in range(len(tc_jpn.event_name)) if
    dt.datetime.fromordinal(tc_jpn.date[i]).year == target_year
])
```

```
tc_jpn_target_year = tc_jpn.select(event_names = events_in_target_year)
```

```
2025-05-23 14:04:59,948 - climada.hazard.io - INFO - Reading /home/sjuhel/climada/
↳data/hazard/tropical_cyclone/tropical_cyclone_0synth_tracks_150arcsec_historical_
↳JPN_1980_2020/v2/tropical_cyclone_0synth_tracks_150arcsec_historical_JPN_1980_2020.
↳hdf5
```

Then we define vulnerability by loading impact functions for tropical cyclones:

```
# 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.

```
# Calculate direct impacts to Japan due to TC
imp_calc = ImpactCalc(exp_jpn, impf_set, tc_jpn_target_year)
direct_impact_jpn = imp_calc.impact()
```

```
2025-05-23 14:04:59,995 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2025-05-23 14:04:59,996 - climada.entity.exposures.base - INFO - Matching 21999_
↳exposures with 21962 centroids.
2025-05-23 14:05:00,005 - climada.util.coordinates - INFO - No exact centroid match_
↳found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2025-05-23 14:05:00,035 - climada.util.coordinates - WARNING - Distance to closest_
↳centroid is greater than 100km for 1 coordinates.
2025-05-23 14:05:00,037 - climada.engine.impact_calc - INFO - Calculating impact for_
↳65976 assets (>0) and 111 events.
```

4.4.2 Load a Multi Regional Input Output Table (MRIOT)

At the core of Input Output modeling lies (Multi-Regional) Input-Output Tables, or MRIOTs, a depiction of the economy production structure over multiple regions.

This module uses the `pymrio` python package and custom functions to download, parse and save MRIOTs automatically. Currently supported tables are:

- [EXIOBASE3](#) (1995-2020; 44 countries; 163 industries)
- [WIOD16](#) (2000-2014; 43 countries; 56 industries)
- [OECD23](#) (1995-2020; 66 countries; 45 industries)
- [EORA26²](#)

To load an MRIOT from the available ones, simply use:

```
mriot = get_mriot("WIOD16", mriot_year=2010)
```

Note that the first time this function is called for a specific MRIOT and year, it requires to download and parse the file. By default, the function then saves the parsed data in the climada data folder, which greatly speeds up subsequent calls to the function.

`mriot` is a `pymrio.IOSystem`, representing a Multi-Regional Input Output Table (MRIOT). You can have a look at [pymrio's documentation](#) to better understand this object.

Here are some important available attributes or methods, which you can have a look at:

- `mriot.Z` : the inter-industry flow matrix (also named intermediate demand or transaction matrix), i.e., the absolute flow of goods from region,sector to region,sector (used to produce new goods) in monetary terms.
- `mriot.Y` : the final demand matrix, i.e, the absolute flow of goods from region,sector to region (used as final products by consumers) in monetary terms. Often subdivided into final demand categories (state, households, etc).
- `mriot.x` : the absolute total amount of goods produced by each region, sector, should be equal to the sum of Z and Y (summing the columns together)
- `mriot.A` : the technical coefficient matrix
- `mriot.get_sectors()` : the sector typology
- `mriot.get_regions()` : the region typology

4.4.3 Instantiate a DirectShocksSet object

In order to evaluate indirect economic impacts from an `Impact` object, we first need to translate that object which defines impacts per centroid (coordinates) into an object which defines impacts on industries ((region, sector) couples). This module implements a `DirectShocksSet` for that very purpose: a set of direct shocks on the economy (one for each event held by the `Impact` object).

To do this we need to map the geographic exposure and impact to a) the countries or regions of the MRIOT and b) its sectors.

Mapping to countries is straightforward, as exposure contains latitude and longitude information, and even a regional id. Thus the region mapping is done automatically for EXIOBASE3, WIOD16 and OECD23 MRIOTs. For custom defined MRIOTs, it will work if the countries of the MRIOT are in ISO3 format.

Mapping to sectors is done by selecting the sectors assumed to be impacted and distributing the impact to the sectors. The default distribution is made proportionally to each sector's production.

² Note that as EORA26 requires connection to an account to be downloaded, you need to download the files yourself and put them in the climada data folder. Parsing is still automatically handled.

Results are highly dependant on this modeling choice which requires carefull consideration.

The minimum requirements to instantiate a `DirectShocksSet` are:

- a MRIOT (in the form of a `pymrio.IOSystem`,
- an `Exposures` object,
- an `Impact` object,

In addition it is strongly advised to also give:

- the list of sectors affected by the `Impact`,
- and how the impact is distributed to the sectors.

Further note that exposure values are converted to the “monetary factor” of the MRIOT, assuming they are in unit value in exposure. For instance, EXIOBASE3 is in millions, so exposure values are divided by 1'000'000.

```
direct_shocks = DirectShocksSet.from_exp_and_imp(
    mriot=mriot,
    exposure=exp_jpn,
    impact=direct_impact_jpn,
    shock_name="TCs in JPN",
    affected_sectors=["Forestry and logging", "Manufacture of motor vehicles, trailers_
↳and semi-trailers", "Mining and quarrying"],
    impact_distribution=None, # None distribute the impact proportionally to the_
↳production of each impacted sectors
)
```

```
2025-05-23 14:05:01,156 - climada.util.coordinates - INFO - Setting region_id 21999_
↳points.
```

You can have a look at the different attributes and methods available (mainly those not starting by “_”):

```
[avail for avail in direct_shocks.__dir__() if avail[0] != "_"]
```

```
['mriot_sectors',
 'mriot_regions',
 'mriot_industries',
 'event_ids',
 'mriot_name',
 'monetary_factor',
 'name',
 'exposure_assets',
 'impacted_assets',
 'event_dates',
 'from_exp_and_imp',
 'from_assets_and_imp',
 'event_ids_with_impact',
 'impacted_assets_not_null',
 'exposure_assets_not_null',
 'relative_impact',
 'relative_impact_not_null',
 'combine']
```

This object stores the “translated” exposure and impact as `pd.Series` and `pd.DataFrame`.

Note that both the translated exposure and impacts use the full typology of regions and sectors from the MRIOT, which results in multiple null values. For convenience, you can access filtered version of the Series/DataFrame via:

```
direct_shocks.exposure_assets_not_null
```

```
region  sector
JPN     Forestry and logging          3.760612e+05
        Manufacture of motor vehicles, trailers and semi-trailers 1.692583e+07
        Mining and quarrying         1.603689e+06
dtype: float64
```

```
direct_shocks.impacted_assets_not_null
```

```
region          JPN  \
sector  Forestry and logging
event_id
3236          0.000315
3242          5.271078
3243          0.010664
3244          38.337255
3248          821.041044
3250          2.049857
3253          220.733284
3254          0.000030
3255          6977.844799
3257          0.001525

region          \
sector  Manufacture of motor vehicles, trailers and semi-trailers
event_id
3236          0.014181
3242          237.241644
3243          0.479978
3244          1725.490101
3248          36953.564080
3250          92.260325
3253          9934.803622
3254          0.001354
3255          314060.103045
3257          0.068660

region          \
sector  Mining and quarrying
event_id
3236          0.001344
3242          22.478182
3243          0.045477
3244          163.486818
3248          3501.278036
3250          8.741486
3253          941.303243
3254          0.000128
3255          29756.581490
```

(continues on next page)

(continued from previous page)

3257	0.006505
------	----------

Most approaches use the `relative_impact` as a basis for the shock, which is defined as the amount of impacted assets over the amount of exposure assets:

```
direct_shocks.relative_impact_not_null
```

```

region          JPN  \
sector  Forestry and logging
event_id
3236          8.378554e-10
3242          1.401655e-05
3243          2.835774e-08
3244          1.019442e-04
3248          2.183265e-03
3250          5.450860e-06
3253          5.869611e-04
3254          7.997938e-11
3255          1.855508e-02
3257          4.056513e-09

region          \
sector  Manufacture of motor vehicles, trailers and semi-trailers
event_id
3236          8.378554e-10
3242          1.401655e-05
3243          2.835774e-08
3244          1.019442e-04
3248          2.183265e-03
3250          5.450860e-06
3253          5.869611e-04
3254          7.997938e-11
3255          1.855508e-02
3257          4.056513e-09

region
sector  Mining and quarrying
event_id
3236          8.378554e-10
3242          1.401655e-05
3243          2.835774e-08
3244          1.019442e-04
3248          2.183265e-03
3250          5.450860e-06
3253          5.869611e-04
3254          7.997938e-11
3255          1.855508e-02
3257          4.056513e-09

```

Combining multiple DirectShocksSet

In the possible case that you have different source of exposure data for different sectors, you can define multiple `DirectShocksSet` (each corresponding to the impacts of the same events but over different exposure), and combine them into one single `DirectShocksSet` using the `combine` class method, for instance, assuming `exp_jpn_forestry` and `exp_jpn_mining` are `Exposures` object defining exposure data for the Forestry sector and the Mining sector respectively:

```
direct_shocks_forestry = DirectShocksSet.from_exp_and_imp(
    mriot=mriot,
    exposure=exp_jpn_forestry,
    impact=direct_impact_jpn,
    shock_name="TCs in JPN on forestry",
    affected_sectors=["Forestry and logging"],
    impact_distribution=None, # None distribute the impact proportionally to the
    ↪ production of each impacted sectors

direct_shocks_mining = DirectShocksSet.from_exp_and_imp(
    mriot=mriot,
    exposure=exp_jpn_mining,
    impact=direct_impact_jpn,
    shock_name="TCs in JPN on mining",
    affected_sectors=["Mining and quarrying"],
    impact_distribution=None, # None distribute the impact proportionally to the
    ↪ production of each impacted sectors

direct_shocks = DirectShocksSet.combine([direct_shocks_forestry,direct_shocks_mining],
    ↪ kind="merge")
)
```

This `combine` method can also be used with `kind="concat"`, to concatenate `DirectShocksSet` from different impacts into one.

Defining custom DirectShocksSet

You can also define `DirectShocksSet` directly with a `Series` of exposure assets:

```
exposure_assets = pd.Series([your_exposure_data],
                            index=pd.MultiIndex.from_product(
                                [list_of_regions, list_of_sectors],
                                names=["region", "sectors"]
                            )
                        )

direct_shocks = DirectShocksSet.from_assets_and_imp(exposure_assets=exposure_assets,
                                                    impact=your_Impact_object,
                                                    affected_sectors=your_list_of_affected_sectors,
                                                    impact_distribution=your_impact_distribution
                                                    )
```

4.4.4 Leontief (and Ghosh) static models

The mathematical idea behind Input-Output (I-O) analysis revolves around using linear algebra to represent the interconnections between industries in an economy. It's based on the Leontief model, named after economist Wassily Leontief, who developed the theory circa 1960.

The main goal is to see how changes in demand affect the entire economic system by solving for the total output required

to meet that demand, through the “Leontief inverse”. This matrix shows how a change in final demand for one industry’s output affects the overall economy (assuming the “recipes” of production and the trade relationship remain the same).

In the Ghosh model, the focus is on how changes in the supply of inputs (like raw materials or labor) affect the total output of industries. Note however that the Ghosh model assumes a supply-driven economy, which is in general not verified. Thus results with this approach should be taken with caution.

In the module we employ the already well implemented methods of the `pymrio` package to compute these indirect effects. For the Leontief approach, the change (decrease) in final demand is computed as the `relative_impact` times the original final demand. For the Ghosh approach, the change (decrease) in value added is computed as the `relative_impact` times the original value added.

```
model = StaticIOModel(mriot, direct_shocks)
```

Computation of the indirect impacts is straightforward:

```
res = model.calc_indirect_impacts()
```

`res` is a `pd.DataFrame` which contains the results per (event, region, sector, method), for both the Leontief approach (shock assumed as a degraded demand) and the Ghosh approach (shock assumed as a degraded value added), and for three different metrics:

1. absolute production change: the absolute loss or gain of production
2. relative production change: the change in production relative to the original production in the MRIOT
3. production lost to shock ratio: the change in production relative to the direct impact on the (region,sector) (with 0 if the sector wasn’t directly shocked).

By default, only events with non null (direct) impacts are considered. To include all events, you can use `model.calc_indirect_impacts(event_ids=None)`. You can also compute for a specific set of events with `model.calc_indirect_impacts(event_ids=[<list of event ids>])`

```
res
```

```

  event_id region                sector \
0      3236  AUS  Accommodation and food service activities
1      3236  AUS  Accommodation and food service activities
2      3236  AUS  Accommodation and food service activities
3      3236  AUS  Accommodation and food service activities
4      3236  AUS  Accommodation and food service activities
...      ...   ...
197115   3257  USA  Wholesale trade, except of motor vehicles and ...
197116   3257  USA  Wholesale trade, except of motor vehicles and ...
197117   3257  USA  Wholesale trade, except of motor vehicles and ...
197118   3257  USA  Wholesale trade, except of motor vehicles and ...
197119   3257  USA  Wholesale trade, except of motor vehicles and ...

  method                metric                value
0      ghosh  absolute production change -9.897446e-08
1      ghosh  production lost to sector shock ratio  0.000000e+00
2      ghosh  production lost to total shock ratio -6.248332e-06
3      ghosh  relative production change -1.487147e-12
4      leontief  absolute production change -2.236537e-08
...      ...   ...
197115   ghosh  relative production change -3.018526e-13

```

(continues on next page)

(continued from previous page)

```
197116 leontief          absolute production change -1.634949e-06
197117 leontief  production lost to sector shock ratio  0.000000e+00
197118 leontief  production lost to total shock ratio -2.131873e-05
197119 leontief          relative production change -1.304762e-12
```

```
[197120 rows x 6 columns]
```

The tidy format of the results allows for easy plotting and data manipulation.

Top ten impacted sectors in Japan (relative to their production), with the Leontief method:

```
import seaborn as sns
from matplotlib.ticker import PercentFormatter

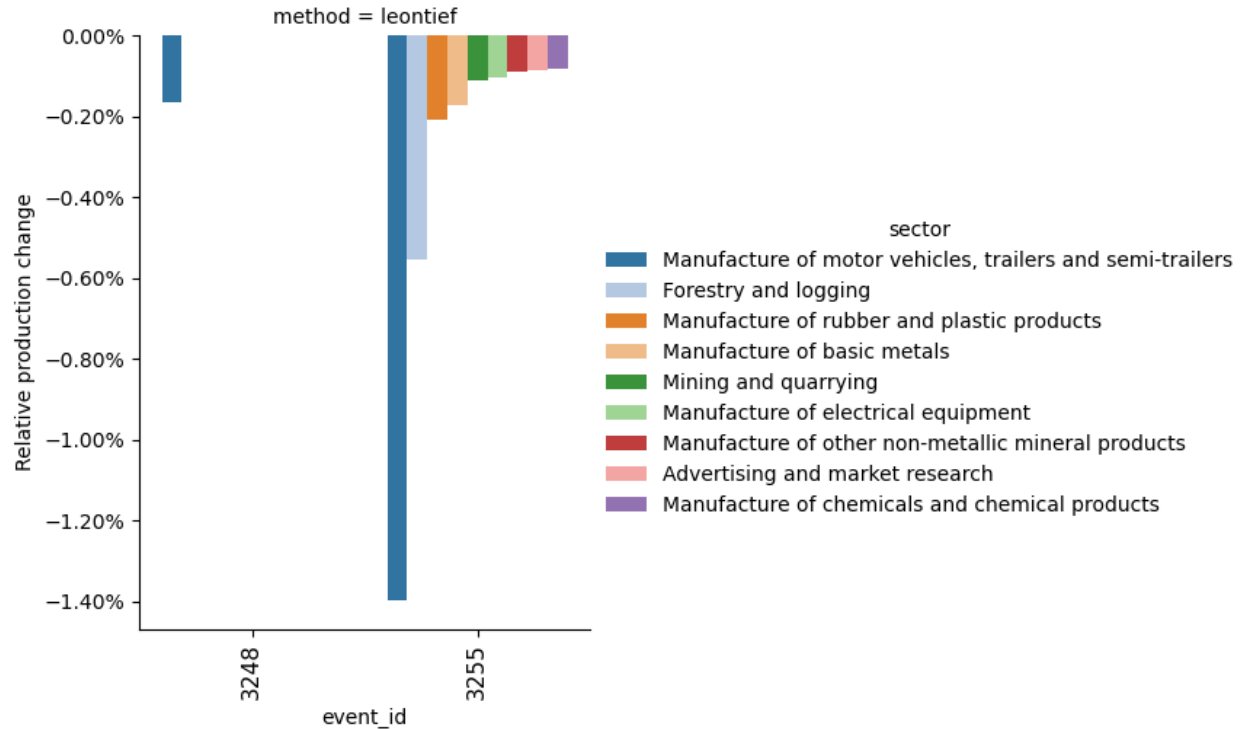
plot_df = res.copy()
plot_df = plot_df.loc[
    (plot_df["region"] == "JPN")
    & (plot_df["metric"] == "relative production change")
    & (plot_df["method"] == "leontief")
].sort_values("value").head(10)

g = sns.catplot(
    plot_df,
    kind="bar",
    x="event_id",
    y="value",
    hue="sector",
    row="method",
    #aspect=4,
    palette="tab20",
    # col = "metric"
)

g.set_ylabels("Relative production change")
g.axes.flat[0].yaxis.set_major_formatter(PercentFormatter(1))

# g.axes
g.set_xticklabels(g.axes[0, 0].get_xticklabels(), rotation=90, fontsize=11)
```

```
<seaborn.axisgrid.FacetGrid at 0x7e7189b24e50>
```



Top ten impacted sectors outside Japan (relative to their production), with the Leontief method:

Note how the impacts are two orders of magnitude below the previous ones.

```
import seaborn as sns
from matplotlib.ticker import PercentFormatter

plot_df = res.copy()
plot_df = plot_df.loc[
    (plot_df["region"] != "JPN")
    & (plot_df["metric"] == "relative production change")
    & (plot_df["method"] == "leontief")
].sort_values("value").head(10)

g = sns.catplot(
    plot_df,
    kind="bar",
    x="event_id",
    y="value",
    hue="sector",
    col="region",
    col_wrap=3,
    palette="tab20",
)

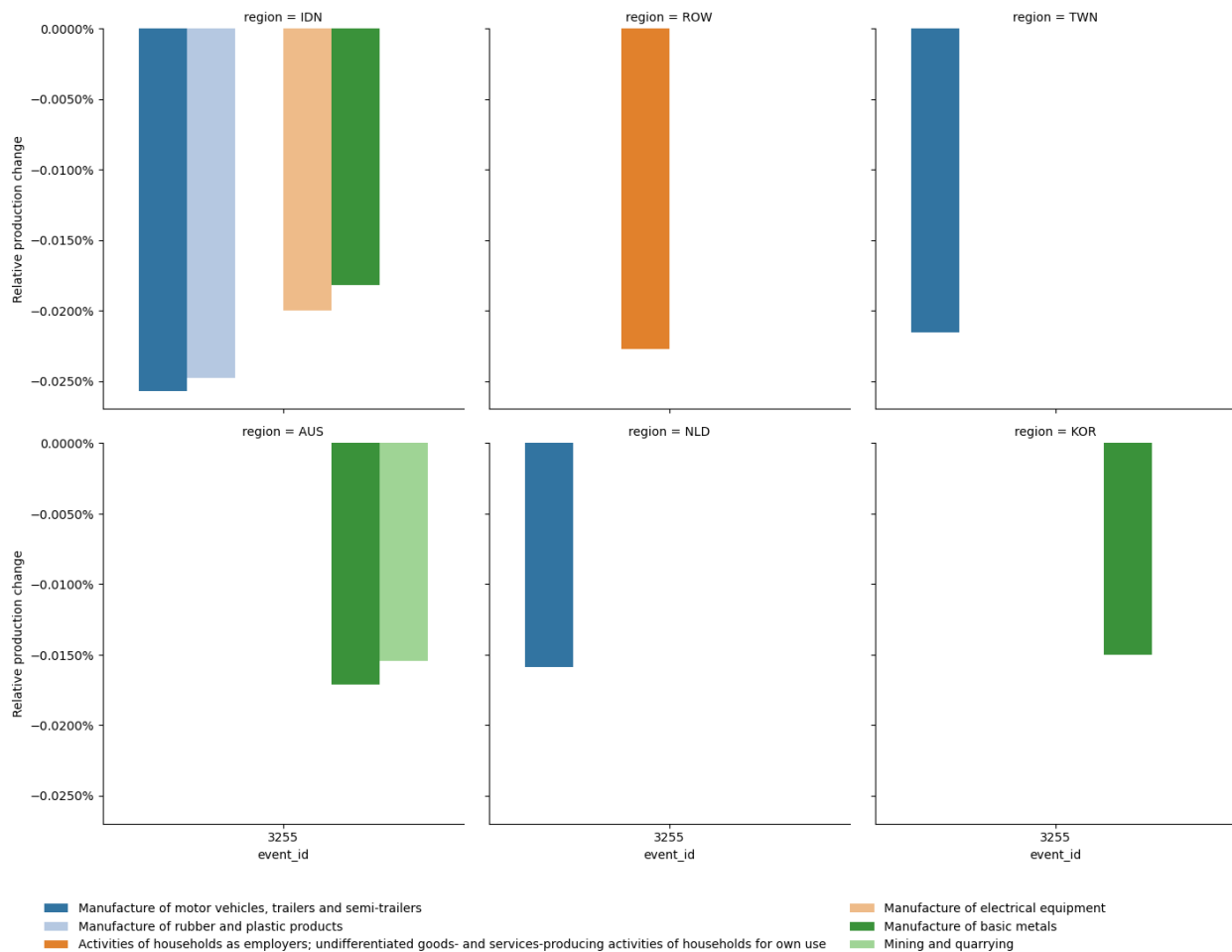
sns.move_legend(g, "lower center",
    bbox_to_anchor=(.3, -0.1), ncol=2, title=None, frameon=False)

g.set_ylabels("Relative production change")
```

(continues on next page)

(continued from previous page)

```
g.axes.flat[0].yaxis.set_major_formatter(PercentFormatter(1))
```



4.4.5 Dynamic models

Static I-O models simply compute the change in equilibrium due to a shock. As such the recovery phase (and possible arising complications) is not accounted for.

Dynamic models fill this gap. There is a vast literature on the subject as many types of model exists, and it cannot be summarised within this notebook. In addition, only the ARIO model is currently available within the module, via the `boario` package.

If you are interested feel free to get in touch, and I can also suggest these two reviews:

1. Botzen et al. 2019. <https://www.journals.uchicago.edu/doi/10.1093/reep/rez004>
2. Coronese, Luzatti. 2022. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4101276

ARIO with `BoARIO`

The ARIO model offers a more “advanced” approach to compute indirect impacts. To understand more about the model we refer you to its seminal papers [Hallegatte 2008](#) and [Hallegatte 2013](#), as well as [the documentation](#) of the `boario` package which is the implementation used in CLIMADA.

`boario` implements different kind of events, to account for a broad typology of impacts:

- `recovery` events, where the direct impact is translated into a loss of production capacity which is recovered over time exogenously.
- `rebuild` events, where the direct impact is translated into a loss of productive capital (assets) resulting in a loss of production capacity. In this case assets are also recovered over time, but through an endogenous reconstruction, which involves an additional final demand in the model.

For more details, we again refer to the `boario` documentation on [How to define events](#)

```
# We redefine a direct shock on all sectors this time
direct_shocks = DirectShocksSet.from_exp_and_imp(
    mriot,
    exp_jpn,
    direct_impact_jpn,
    shock_name="TCs in JPN",
    affected_sectors="all",
    impact_distribution=None,
)
```

```
2025-05-23 14:05:07,771 - climada.util.coordinates - INFO - Setting region_id 21999_
↳points.
```

There are multiple things that can be parameterized within `boario`, the simulation itself, as well as the model and the events.

These parameters are passed to the package using three dictionaries read as kwargs by the different modules/classes of the `boario` package. This aspect still requires work as some parameters configurations may conflict with the coupling.

If you want to dig deeper, have a look at BoARIO's documentation, in particular:

- [Simulation context](#) and [Simulation object](#) for simulation parameters.
- [ARIOPsiModel object](#) for the model parameters.
- [Event object](#) for the events parameters.

We raise your attention on several important points:

The use of `boario`, although it offers more depth in results and in modelling possibilities, also requires you to configure it more than the previous approaches. For this a good understanding of the model is required. The purpose of this tutorial is to show what is available in CLIMADA, but it does not replace an extensive reading about the model, its assumptions and its limitations.

Some options for `boario` are set by default either by CLIMADA or BoARIO, if you do not specify them. This *does not mean* that these default values are good in general as no generic case exists. You should get familiar with the parameters and reflect on what choices are fitting for your case.

`boario` computations are much longer with respect to the previous approach. A progress bar is shown by default which indicates the estimated time a simulation requires. This time is highly dependent on:

- The size of the MRIOT used (number of regions x number of sectors)
- The number of events to simulate

```
dyn_model = BoARIOModel(
    mriot,
    direct_shocks,
    simulation_kwargs={"show_progress": True},
    model_kwargs={"order_type": "alt"},
```

(continues on next page)

(continued from previous page)

```
event_kwargs={"recovery_tau": 180},
)
```

```
/home/sjuhel/miniforge3/envs/supplychaindev/lib/python3.11/site-packages/boario/model_
↳base.py:180: UserWarning: Custom monetary factor found in the IOSystem, continuing_
↳with this one (1000000)
warnings.warn(
```

```
2025-05-23 14:05:09,499 - climada_petals.engine.supplychain.core - WARNING - Impact_
↳for following events was too small to have an effect, skipping them for efficiency:_
↳[249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 903, 904, 905, 906, 907, 908,_
↳909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 1479, 1480, 1481, 1482, 1483,
↳1484, 1485, 1486, 1487, 1488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 1496, 1497,
↳1498, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 3232, 3233, 3234, 3235, 3237,
↳3238, 3239, 3240, 3241, 3245, 3246, 3247, 3249, 3251, 3252, 3256, 3258, 3259, 3260,
↳3261, 3262, 3263, 3264, 3265, 3848, 3849, 3850, 3851, 3852, 3853, 3854, 3855, 3856,
↳3857, 3858, 3859, 3860, 3861, 3862, 3863, 3864, 3865, 3866, 3867, 3868, 3869]
```

```
/home/sjuhel/miniforge3/envs/supplychaindev/lib/python3.11/site-packages/boario/event.
↳py:853: UserWarning: Impact for some industries (True total), is smaller than 1e-05_
↳and will be considered as 0. by the model.
self._check_negligeable_impact(impact)
```

The time required to run the simulation heavily depends on the MRIOTs (the number of regions * sectors), and the number of steps to simulate of course. With WIOD and the choosen events, it should take around 2 minutes on a recent computer. EXIOBASE 3 is more in the 30-45 minutes range. An estimate of the progress should be shown during the simulation.

Also note that some warnings may be raised by events that are too small to have a meaningful impacts.

```
res = dyn_model.run_sim()
```

```
Processed: Step: 725 ~ 100% Time: 0:03:06
```

```
Processed: Step: 725 ~ 100% Time: 0:03:06
```

`res` contains a Simulation object from `boario` which among other things contains all the time series of the state variables.

For instance you can plot the evolution of the `production_realised` in relative terms using the following. The second graph, below, shows the impacts and occurrences of the different (non-zero) events (note the log scale). Also note, that we focus between step 160 and 500.

During this simulation, no “shortage” of input happened, something you can check with `res.model.had_shortage`, and which would show a characteristic shape in the production output.

However you can easily observe the “direct” effect of each shock and the following recovery.

```
import matplotlib.pyplot as plt
occ = [ev.occurrence for ev in res.all_events]
imps = [ev.impact.sum() for ev in res.all_events]

sector, region = ("JPN", "Manufacture of basic metals")

fig,axs = plt.subplots(figsize=(18,8),nrows=2, sharex=True)
```

(continues on next page)

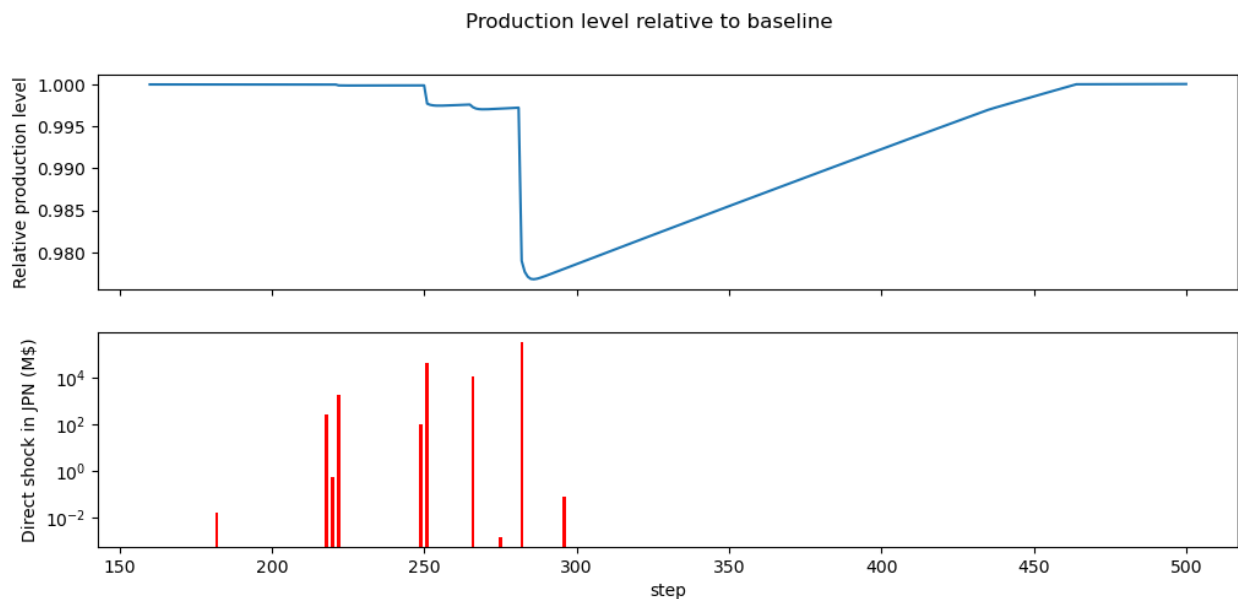
(continued from previous page)

```
(res.production_realised / res.production_realised.loc[0]).loc[
    160:500, (sector, region)
].plot(legend=False, figsize=(12,5), ax=axes[0])

bars = axes[1].bar(occ,imps,width=1,color='r',label='Direct impact',alpha=1)
axes[1].set_yscale("log")
axes[0].set_ylabel("Relative production level")
axes[1].set_ylabel("Direct shock in JPN (M$)")
axes[1].set_xlabel("step")

fig.suptitle(f"Production level of {sector, region} relative to baseline")
```

```
Text(0.5, 0.98, 'Production level relative to baseline')
```



In the next simulation, we change the model parameters to observe a shortage (note that this is purely for showcasing, and does not aim at reflecting something real, users should absolutely read BoARIO's documentation to understand how they can use it and set it up for their study).

The so called “shortage of inputs” is easily observed by the characteristic inflexion point which is not related to an event at step ~340.

```
dyn_model = BoARIOModel(
    mriot,
    direct_shocks,
    simulation_kwargs={"show_progress": True},
    model_kwargs={"order_type": "alt", "psi_param": 0.999, "alpha_max": 1.0},
    event_kwargs={"recovery_tau": 90},
)
res = dyn_model.run_sim()
```

```
/home/sjuhel/miniforge3/envs/supplychaindev/lib/python3.11/site-packages/boario/model_
↳base.py:180: UserWarning: Custom monetary factor found in the IOSystem, continuing_
```

(continues on next page)

(continued from previous page)

```
↳with this one (1000000)
  warnings.warn(
```

```
2025-05-23 14:23:31,851 - climada_petals.engine.supplychain.core - WARNING - Impact
↳for following events was too small to have an effect, skipping them for efficiency:
↳[249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 903, 904, 905, 906, 907, 908,
↳909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 1479, 1480, 1481, 1482, 1483,
↳1484, 1485, 1486, 1487, 1488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 1496, 1497,
↳1498, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 3232, 3233, 3234, 3235, 3237,
↳3238, 3239, 3240, 3241, 3245, 3246, 3247, 3249, 3251, 3252, 3256, 3258, 3259, 3260,
↳3261, 3262, 3263, 3264, 3265, 3848, 3849, 3850, 3851, 3852, 3853, 3854, 3855, 3856,
↳3857, 3858, 3859, 3860, 3861, 3862, 3863, 3864, 3865, 3866, 3867, 3868, 3869]
```

```
/home/sjuhel/miniforge3/envs/supplychaindev/lib/python3.11/site-packages/boario/event.
↳py:853: UserWarning: Impact for some industries (True total), is smaller than 1e-05
↳and will be considered as 0. by the model.
  self._check_negligeable_impact(impact)
Processed: Step: 725 ~ 100% Time: 0:03:36
Processed: Step: 725 ~ 100% Time: 0:03:36
```

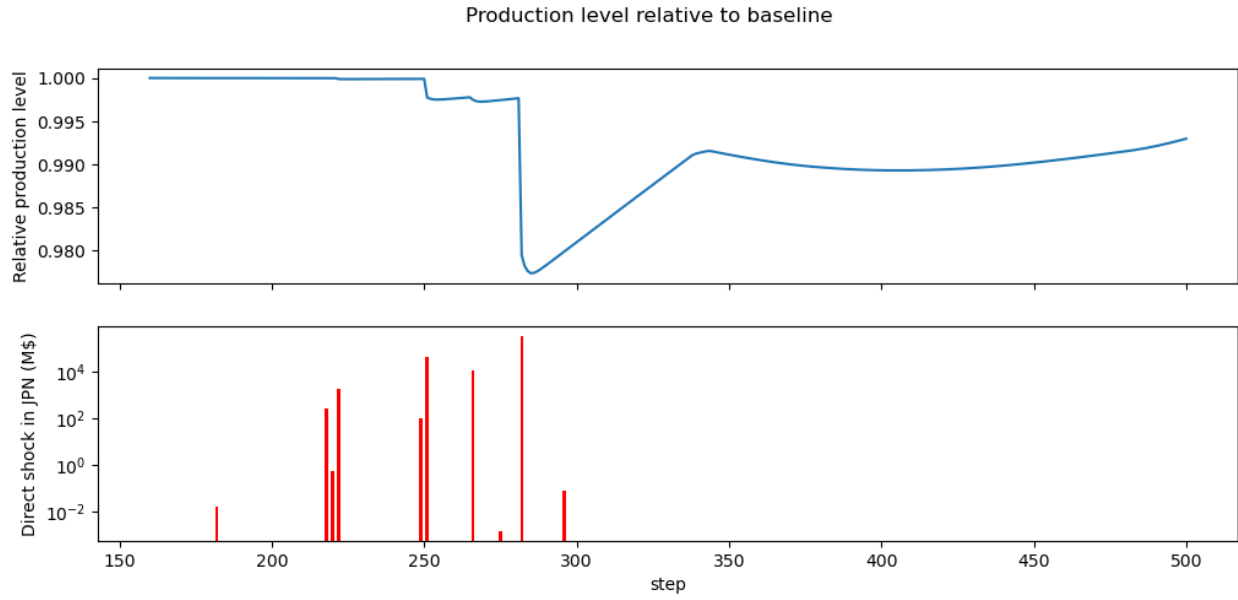
```
occ = [ev.occurrence for ev in res.all_events]
imps = [ev.impact.sum() for ev in res.all_events]

fig,axs = plt.subplots(figsize=(18,8),nrows=2, sharex=True)
(res.production_realised / res.production_realised.loc[0]).loc[
    160:500, ("JPN", "Manufacture of basic metals")
].plot(legend=False, figsize=(12,5), ax=axs[0])

bars = axs[1].bar(occ, imps, width=1, color='r', label='Direct impact', alpha=1)
axs[1].set_yscale("log")
axs[0].set_ylabel("Relative production level")
axs[1].set_ylabel("Direct shock in JPN (M$)")
axs[1].set_xlabel("step")

fig.suptitle("Production level relative to baseline")
```

```
Text(0.5, 0.98, 'Production level relative to baseline')
```



CAT BOND TUTORIAL

In this tutorial we create and evaluate our own parametric sovereign CAT bonds. A more detailed description of how the CAT bonds are created is included in the paper “Lowering premiums and increasing investor returns by issuing multi-country CAT bonds” which is published as a preprint (<http://dx.doi.org/10.2139/ssrn.6030454>). To create the CAT bonds we will perform the following steps:

1. Define exposure, hazard and vulnerability for each country which should be protected by a CAT bond.
2. Define the attachment (minimum payout) and exhaustion point (maximum payout = principal).
3. Devide the exposed area of each country into subareas. Each subarea will have its own payout function which uses the hazard intensity to derive the final payout based on a linear scale between the minimum and maximum payout.
4. Calculate the expected damage for each hazard event and subarea in the simulation period. Use the expected damage per subarea per event to calibrate the payout functions per subarea.
5. Simulate the losses of the bonds based on the payouts per hazard event.
6. Calculate the premiums for the bond. We use three different methods:
 - Benchmark Pricing: Premiums are set to achieve a predefined sharpe ratio.
 - Chatoro-Pricing: Based on Chatoro et al. (<https://doi.org/10.1016/j.irfa.2022.102431>) we derive the premiums based on the expected loss of our CAT bond and current market conditions.
 - IBRD-Pricing: Based on the risk multiple (expected loss / annual premium rate) of previously issued parametric sovereign CAT bonds from the World Bank.
7. Simulate the net cash flow of the bond based on the premiums and the losses.

Additionally to the single-country CAT bonds we will also create multi-country CAT bonds. For this we:

1. Derive optimal pools using minimum risk diversification as the objective (as proposed by Ciullo et al. (<https://doi.org/10.1038/s41467-023-36539-4>)). For this we set the constraint as:
 - a fixed number of non-empty pools has to be created
 - a maximum principal is not allowed to be exceeded per multi-country CAT bond
2. Calculate the minimum principal necessary, so that the multi-country CAT bond covers all damages equally as the single-country CAT bonds.
3. Once this is done, the losses are simulated, the premiums calculated and the net cash flow is derived.

5.1 Import packages

```
%load_ext autoreload
%autoreload 2
# import standard packages
from datetime import date

# import CAT bond module parts from CLIMADA-PETALS
from climada_petals.engine.cat_bonds.subareas import Subareas
from climada_petals.engine.cat_bonds.subarea_calculations import SubareaCalculations
from climada_petals.engine.cat_bonds.sng_bond_simulation import _
↳SingleCountryBondSimulation
from climada_petals.engine.cat_bonds.mlt_bond_simulation import MultiCountryBond
from climada_petals.engine.cat_bonds.premium_class import PremiumCalculations
from climada_petals.engine.cat_bonds.helpers_tutorial_cat_bonds import create_
↳earthquake_mmi_impact_function, remap_event_years_even

# import CLIMADA core modules
from climada.entity import ImpfSetTropCyclone
from climada.util.api_client import Client
client = Client()
```

The autoreload extension is already loaded. To reload it, use:

```
%reload_ext autoreload
```

5.2 Set CAT Bond Basic Settings

We first need to set the basics for the creation of CAT bonds. This includes:

1. Exhaustion point (Exposure_Share (set as share) or Return_Period (set as years) or None (-> set in the unit of the exposure layer)).
2. Attachment point (Exposure_Share (set as share) or Return_Period (set as years) or None (-> set in the unit of the exposure layer)).
3. Term of the bond in years.
4. Start year of the bond simulation.
5. Number of terms to simulate.
6. The parametric index statistic (either a float which is used to calculate percentiles of hazard intensity in each subarea or a string called 'mean' which calculates the average of the hazard intensity in each subarea).
7. Resolution for the creation of subareas. Has to be set for each country. Don't set them to small or you will end up with too many subareas which slows down computation.
8. Define whether your CAT bond is considered peak or multi peril, has a hybrid trigger and if it is investment graded.
9. Target sharpe ratio which has to be met by setting premiums accordingly.

```
### Bond Basics ###
countries = [659, 388, 822] # St. Kitts and Nevis, Jamaica, Samoa
exhaustion_point = 0.5 # principal should be 50% of exposure
attachment_point = 0.05 # minimum payout should be 5% of exposure
exhaustion_point_method = 'Exposure_Share' # could also be "Return_Period" or None
```

(continues on next page)

(continued from previous page)

```

attachment_point_method = 'Exposure_Share' # could also be "Return_Period" or None
term = 3 # term of bond in years
start_year = 1980 # start year for simulation
num_of_terms = 1000 // term # number of terms to simulate
par_index = 60 # statistic for parametric index (e.g. 60 for 60th percentile)

### Subarea Basics ###
resolution_samoa = 0.5 # resolution used to derive subareas for Samoa
resolution_jamaica = 0.5 # resolution used to derive subareas for Jamaica
resolution_st_kitts = 0.05 # resolution used to derive subareas for St. Kitts and
↳Nevis

### Pricing Basics ###
peak_peril = 0 # indicator if bond is considered peak/multi peril (1) or not (0)
hybrid_trigger = 0 # indicator if bond contains a hybrid trigger (1) or not (0)
investment_graded = 0 # indicator if bond is investment graded (1) or not (0)
target_sharpe = 0.5 # target Sharpe ratio for benchmark pricing

```

As you can see we will create CAT bonds for St. Kitts and Nevis, Jamaica and Samoa. For each country the attachment point will equal 5% of the exposure and the exhaustion point 50%. Check the other parameters for the simulation, subarea, and pricing settings.

The starting date for the simulation of the CAT bond will be in 1980. The term will be set to 3 years, a common value for CAT bonds. Our hazard data covers 1000 years of tropical cyclone events, therefore, the number of terms to simulate will be 333, which results in the simulation of 999 years.

5.3 Define Exposure, Hazard, and Vulnerability per Country

For each country in the list we first define exposure, hazard and vulnerability.

1. Hazard: For each country we import tropical cyclone hazard data sets from the CLIMADA data API, based on STORM [<https://www.science.org/doi/10.1126/sciadv.abm8438>]. This dataset contains synthetic TC track covering a 1000 year period. **ATTENTION:** Please be aware that the assigned dates (years and month) for the hazard events matter for the CAT bond simulation. It may be necessary to assign dates to each event, depending on what hazard data set you use.
2. Exposure: LitPop objects constructed using the countries GDP from 2020 as total asset value which is then disaggregated proportional to a combination of nightlight intensity and gridded population data.
3. Vulnerability: Functions are taken from Eberenz et al. 2021: [<https://doi.org/10.5194/nhess-21-393-2021>] which have calibrated regional TC wind impact functions.

```

### GET HAZARD ###
# get hazard for jamaica
hazard_jamaica = client.get_hazard(
    "tropical_cyclone",
    properties={
        "event_type": "synthetic",
        "model_name": "STORM",
        "country_name": "Jamaica",
        "climate_scenario": None
    },
)

```

(continues on next page)

```

# get hazard for st. kitts and nevis
hazard_st_kitts = client.get_hazard(
    "tropical_cyclone",
    properties={
        "event_type": "synthetic",
        "model_name": "STORM",
        "country_iso3alpha": "KNA",
        "climate_scenario": None
    },
)

# get hazard for samoa
hazard_samoa = client.get_hazard(
    "tropical_cyclone",
    properties={
        "event_type": "synthetic",
        "model_name": "STORM",
        "country_name": "Samoa",
        "climate_scenario": None
    },
)

#adjust dates of hazard data sets -> the assigned year for each event in the STORM TC
->hazard data sets from the CLIMADA data API is 1980.
# As the simulation of CAT bonds depends on the distribution of events over time, we
->need to remap the event years for each hazard data set.
# Here, we remap the event years to be evenly distributed over a period of 1000 years
->starting from the defined start year (1980).
# Please be aware that this is a simplified approach for the purpose of this tutorial
->and may not reflect the actual temporal distribution
# of tropical cyclone events in these regions.
hazard_jamaica = remap_event_years_even(hazard_jamaica, start_year=start_year, n_
->years=1000)
hazard_st_kitts = remap_event_years_even(hazard_st_kitts, start_year=start_year, n_
->years=1000)
hazard_samoa = remap_event_years_even(hazard_samoa, start_year=start_year, n_
->years=1000)

### GET EXPOSURE ###
exp_jamaica = client.get_litpop('JAM') # get exposure for jamaica
exp_samoa = client.get_litpop('WSM') # get exposure for samoa
exp_st_kitts = client.get_litpop('KNA') # get exposure for st. kitts and nevis
exp_jamaica.gdf['impf_TC'] = 1 # map impact function ID for Jamaica-> Region North
->America
exp_samoa.gdf['impf_TC'] = 4 # map impact function ID for Samoa -> Region Oceania
exp_st_kitts.gdf['impf_TC'] = 1 # map impact function ID for st. kitts and nevis ->
->Region North America 1

### GET IMPACT FUNCTION ###
impfset = ImpfSetTropCyclone.from_calibrated_regional_ImpfSet() # get impact function
->for tropical cyclone

```

```

2026-05-01 13:16:37,395 - climada.hazard.io - INFO - Reading C:\Users\Kai\climada\
↳data\hazard\tropical_cyclone\TC_JAM_0300as_STORM\v1.1\TC_JAM_0300as_STORM.hdf5
2026-05-01 13:16:44,157 - climada.hazard.io - INFO - Reading C:\Users\Kai\climada\
↳data\hazard\tropical_cyclone\TC_KNA_0300as_STORM\v1.1\TC_KNA_0300as_STORM.hdf5
2026-05-01 13:16:50,744 - climada.hazard.io - INFO - Reading C:\Users\Kai\climada\
↳data\hazard\tropical_cyclone\TC_WSM_0300as_STORM\v1.1\TC_WSM_0300as_STORM.hdf5
2026-05-01 13:17:05,885 - climada.entity.exposures.base - INFO - Reading C:\Users\Kai\
↳climada\data\exposures\litpop\LitPop_150arcsec_JAM\v3\LitPop_150arcsec_JAM.hdf5
2026-05-01 13:17:07,158 - climada.entity.exposures.base - INFO - Reading C:\Users\Kai\
↳climada\data\exposures\litpop\LitPop_150arcsec_WSM\v3\LitPop_150arcsec_WSM.hdf5
2026-05-01 13:17:08,421 - climada.entity.exposures.base - INFO - Reading C:\Users\Kai\
↳climada\data\exposures\litpop\LitPop_150arcsec_KNA\v3\LitPop_150arcsec_KNA.hdf5

```

5.4 Initialise Subareas Class

Now we actually start to create a CAT bond. First, we create an instance of the Subareas class for each country which stores the subareas of the resulting CAT bonds. Note: The subarea class can also be generated with an geodataframe. Then the subareas are not calculated from the resolution but from the given geodataframe. Be sure that the given subareas cover the entire area exposed.

```

st_kitts_subareas = Subareas.from_resolution(hazard_st_kitts, impfset, exp_st_kitts,
↳resolution=resolution_st_kitts) # derive subareas for St. Kitts and Nevis
jamaica_subareas = Subareas.from_resolution(hazard_jamaica, impfset, exp_jamaica,
↳resolution=resolution_jamaica) # derive subareas for Jamaica
samoa_subareas = Subareas.from_resolution(hazard_samoa, impfset, exp_samoa,
↳resolution=resolution_samoa) # derive subareas for Samoa
# Plot subareas per CAT bond
jamaica_subareas.plot()
st_kitts_subareas.plot()
samoa_subareas.plot()

```

```

2026-05-01 13:17:59,907 - climada_petals - INFO - Rasterizing exposure with width: 4,
↳height: 5, resolution: 0.05
2026-05-01 13:17:59,947 - climada_petals - INFO - Exposure perimeter polygon created.
2026-05-01 13:17:59,950 - climada_petals - INFO - Number of polygons in exposure
↳perimeter: 2
2026-05-01 13:17:59,984 - climada_petals - INFO - Merging overlapping grid cells into
↳single polygons.
2026-05-01 13:18:00,033 - climada_petals - INFO - Subareas created.
2026-05-01 13:18:00,051 - climada_petals - INFO - Rasterizing exposure with width: 42,
↳height: 15, resolution: 0.05

```

```

c:\Users\Kai\miniforge3\envs\climada_env\Lib\site-packages\shapely\set_operations.
↳py:133: RuntimeWarning: invalid value encountered in intersection
return lib.intersection(a, b, **kwargs)

```

```

2026-05-01 13:18:00,289 - climada_petals - INFO - Exposure perimeter polygon created.
2026-05-01 13:18:00,289 - climada_petals - INFO - Number of polygons in exposure
↳perimeter: 1
2026-05-01 13:18:00,444 - climada_petals - INFO - Merging overlapping grid cells into
↳single polygons.

```

(continues on next page)

(continued from previous page)

```

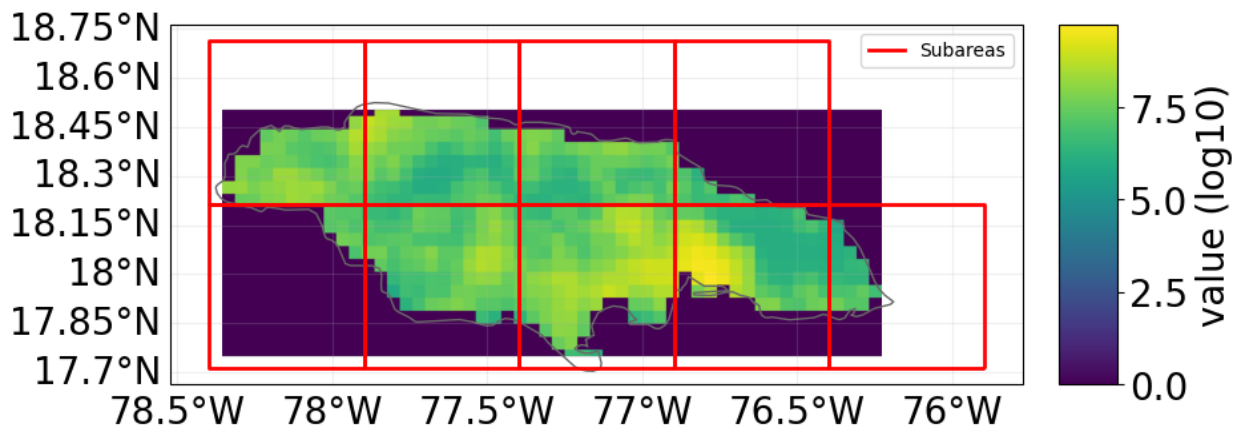
2026-05-01 13:18:00,489 - climada_petals - INFO - Subareas created.
2026-05-01 13:18:00,502 - climada_petals - INFO - Rasterizing exposure with width: 25,
↳ height: 10, resolution: 0.05
2026-05-01 13:18:00,591 - climada_petals - INFO - Exposure perimeter polygon created.
2026-05-01 13:18:00,593 - climada_petals - INFO - Number of polygons in exposure.
↳perimeter: 2
2026-05-01 13:18:00,599 - climada_petals - INFO - Polygon smaller than resolution;
↳adding polygon bounding box.
2026-05-01 13:18:00,607 - climada_petals - INFO - Polygon smaller than resolution;
↳adding polygon bounding box.
2026-05-01 13:18:00,624 - climada_petals - INFO - Merging overlapping grid cells into
↳single polygons.
    
```

```

c:\Users\Kai\miniforge3\envs\climada_env\Lib\site-packages\shapely\set_operations.
↳py:133: RuntimeWarning: invalid value encountered in intersection
return lib.intersection(a, b, **kwargs)
c:\Users\Kai\miniforge3\envs\climada_env\Lib\site-packages\shapely\set_operations.
↳py:133: RuntimeWarning: invalid value encountered in intersection
return lib.intersection(a, b, **kwargs)
    
```

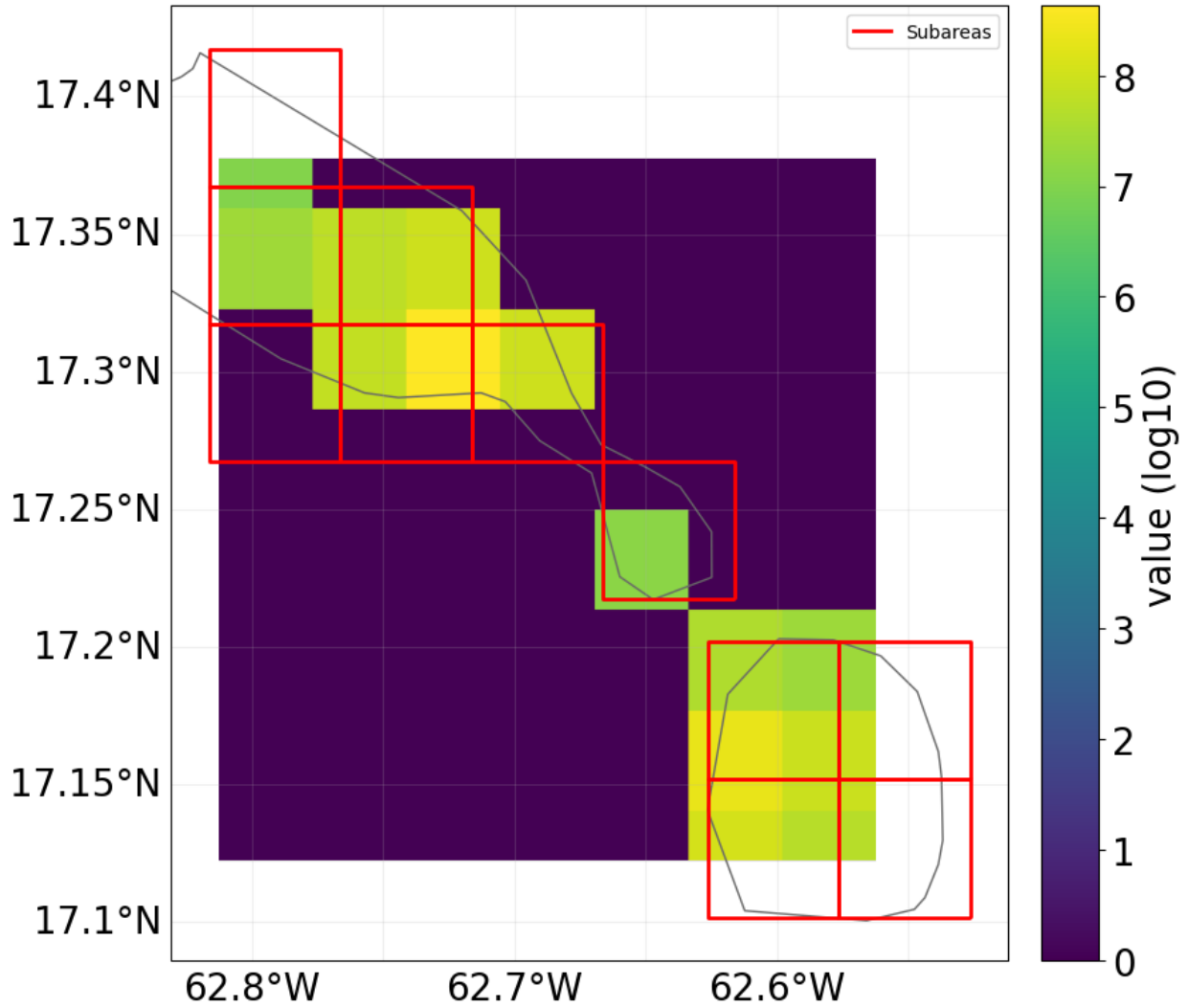
```

2026-05-01 13:18:00,685 - climada_petals - INFO - Subareas created.
2026-05-01 13:18:00,691 - climada.util.coordinates - INFO - Raster from resolution 0.
↳04166665999999708 to 0.04166665999999708.
    
```

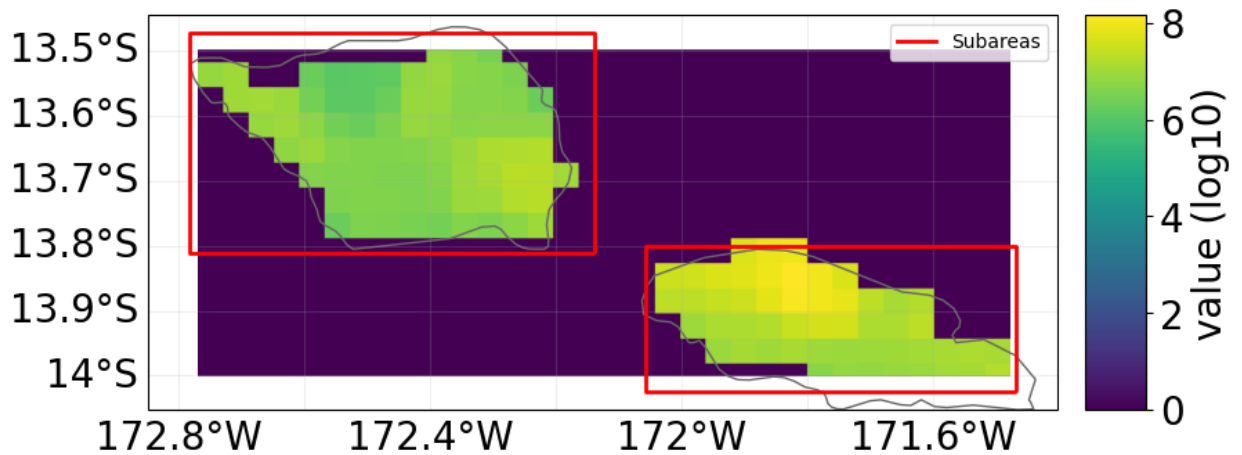


```

2026-05-01 13:18:44,199 - climada.util.coordinates - INFO - Raster from resolution 0.
↳04166665999999708 to 0.04166665999999708.
    
```



```
2026-05-01 13:19:21,353 - climada.util.coordinates - INFO - Raster from resolution 0.
→041666659999975764 to 0.041666659999975764.
```



We can now see for each country the distribution of exposed assets as well as the subareas (red rectangles) covering the

countries. Each country is divided by a different number of subareas depending on the chosen resolution and country size.

5.5 Subarea Calculations

Based on the Subareas class and the generated subareas per CAT bond we can now derive the setups for each bond with the SubareaCalculations class. We first initialise the class and then calculate the payout vs damage dataframe. This dataframe contains for each hazard event given to the Subarea class, the damage and the payout and stores information on the hazard event year and month. In order to be able to create this dataframe the following computations are performed when calling the method `create_pay_vs_dam()`:

1. Calculate the initial guess for the payout function optimization if it is not given with class initialization.
2. Calculate the damage per hazard event per subarea.
3. Derive the attachment and exhaustion point (principal) if not given as a monetary value (`exhaustion/attachment_point_method == None`).
4. Optimize the payout functions per subarea.
5. Calculate the payouts per event and subarea.
6. Return the final payout vs damage dataframe. Please review the logging messages for the attachment/exhaustion points and check the `pay_vs_dam` dataframes per CAT bond to get a feeling for the data.

```
st_kitts_sub_calc = SubareaCalculations(subareas=st_kitts_subareas, index_stat=par_
↳index) # initialize subarea calculations for St. Kitts and Nevis
jamaica_sub_calc = SubareaCalculations(subareas=jamaica_subareas, index_stat=par_
↳index) # initialize subarea calculations for Jamaica
samoa_sub_calc = SubareaCalculations(subareas=samoa_subareas, index_stat=par_index) #_
↳initialize subarea calculations for Samoa
st_kitts_sub_calc.create_pay_vs_dam(attachment_point, exhaustion_point, methods_
↳attachment_point=attachment_point_method, methods_exhaustion_point=exhaustion_point_
↳method) # derive payout vs damage dataframe for St. Kitts and Nevis
jamaica_sub_calc.create_pay_vs_dam(attachment_point, exhaustion_point, methods_
↳attachment_point=attachment_point_method, methods_exhaustion_point=exhaustion_point_
↳method) # derive payout vs damage dataframe for Jamaica
samoa_sub_calc.create_pay_vs_dam(attachment_point, exhaustion_point, methods_
↳attachment_point=attachment_point_method, methods_exhaustion_point=exhaustion_point_
↳method) # derive payout vs damage dataframe for Samoa
```

```
2026-05-01 13:20:02,980 - climada_petals - INFO - Calculating impact and subarea-
↳level impacts per event.
2026-05-01 13:20:02,990 - climada.entity.exposures.base - INFO - Matching 14_
↳exposures with 2 centroids.
2026-05-01 13:20:03,000 - climada.util.coordinates - INFO - No exact centroid match_
↳found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2026-05-01 13:20:03,017 - climada.engine.impact_calc - INFO - Calculating impact for_
↳42 assets (>0) and 712800 events.
2026-05-01 13:20:03,517 - climada_petals - INFO - Calculating parametric index per_
↳subarea.
2026-05-01 13:20:05,873 - climada_petals - INFO - The attachment point and the_
↳principal of the CAT bond is: 67258571.85 and 672585718.5 [USD], respectively.
2026-05-01 13:20:05,875 - climada_petals - INFO - Calibrating payout functions by_
↳optimizing trigger thresholds to minimize basis risk.
2026-05-01 13:20:12,630 - climada_petals - INFO - Calculating pay-versus-damage table_
```

(continues on next page)

(continued from previous page)

```

↳for all events using optimized payout functions.
2026-05-01 13:20:12,750 - climada_petals - INFO - Calculating impact and subarea-
↳level impacts per event.
2026-05-01 13:20:12,777 - climada.entity.exposures.base - INFO - Matching 542↳
↳exposures with 137 centroids.
2026-05-01 13:20:12,781 - climada.util.coordinates - INFO - No exact centroid match↳
↳found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2026-05-01 13:20:12,800 - climada.engine.impact_calc - INFO - Calculating impact for↳
↳1626 assets (>0) and 712800 events.
2026-05-01 13:20:13,624 - climada_petals - INFO - Calculating parametric index per↳
↳subarea.
2026-05-01 13:20:19,444 - climada_petals - INFO - The attachment point and the↳
↳principal of the CAT bond is: 3850326010.64 and 38503260106.4 [USD], respectively.
2026-05-01 13:20:19,460 - climada_petals - INFO - Calibrating payout functions by↳
↳optimizing trigger thresholds to minimize basis risk.
2026-05-01 13:20:42,535 - climada_petals - INFO - Calculating pay-versus-damage table↳
↳for all events using optimized payout functions.
2026-05-01 13:20:42,882 - climada_petals - INFO - Calculating impact and subarea-
↳level impacts per event.
2026-05-01 13:20:42,898 - climada.entity.exposures.base - INFO - Matching 131↳
↳exposures with 30 centroids.
2026-05-01 13:20:42,905 - climada.util.coordinates - INFO - No exact centroid match↳
↳found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2026-05-01 13:20:42,922 - climada.engine.impact_calc - INFO - Calculating impact for↳
↳393 assets (>0) and 712800 events.
2026-05-01 13:20:43,264 - climada_petals - INFO - Calculating parametric index per↳
↳subarea.
2026-05-01 13:20:45,886 - climada_petals - INFO - The attachment point and the↳
↳principal of the CAT bond is: 108634907.055 and 1086349070.55 [USD], respectively.
2026-05-01 13:20:45,886 - climada_petals - INFO - Calibrating payout functions by↳
↳optimizing trigger thresholds to minimize basis risk.
2026-05-01 13:20:50,904 - climada_petals - INFO - Calculating pay-versus-damage table↳
↳for all events using optimized payout functions.

```

5.6 Single-Country CAT Bond Simulation

1. Now we can simulate the losses of the bonds. For this we create an instance of the `SingleCountryBondSimulation` per CAT bond. Each object needs information on the term of the bond (given in years) and the number of terms to simulate. Afterwards we can call the method `init_loss_simulation()` per CAT bond object
2. With the information on the losses per bond (given as a dataframe containing the relative loss, month + year per loss) we can derive the annual premium rates by creating an instance of the `PremiumCalculations` class and then using the desired premium calculation methods. This module contains three different possibilities to derive premiums:
 - `calc_chatoro_premium()` (Chatoro et al. ([<https://doi.org/10.1016/j.irfa.2022.102431>]) premium)
 - `calc_ibrd_premium()` (World Bank CAT bonds based premium)
 - `calc_benchmark_premium()` (Target sharpe ration premium)
3. Given the annual premium rates we can calculate the net cash flows and premium payments of each bond with `init_return_simulation()`. Premiums are paid at the first of each month relative to the remaining principal of the bond. The simulation is performed for one of the three derived premium rates. Premium rates can also be defined manually if desired.

4. Inspect the metrics of the bond.

```

### ST. KITTS AND NEVIS BOND SIMULATION ###
st_kitts_bond_sim = SingleCountryBondSimulation(subarea_calc=st_kitts_sub_calc,
↳term=term, start_year=start_year, number_of_terms=num_of_terms) # initialize bond
↳simulation for St. Kitts and Nevis
st_kitts_bond_sim.init_loss_simulation() # derive loss simulation for St. Kitts and
↳Nevis
st_kitts_premiums = PremiumCalculations(bond_simulation_class=st_kitts_bond_sim) #
↳initialize premium calculations for St. Kitts and Nevis
st_kitts_premiums.calc_chatoro_premium(peak_multi=peak_peril, investment_
↳graded=investment_graded, hybrid_trigger=hybrid_trigger) # derive Chatoro premium
↳for St. Kitts and Nevis
st_kitts_premiums.calc_ibrd_premium() # derive IBRD premium for St. Kitts and Nevis
st_kitts_premiums.calc_benchmark_premium(target_sharpe = target_sharpe) # derive
↳benchmark premium for St. Kitts and Nevis using target sharpe ratio
st_kitts_bond_sim.init_return_simulation(premium=st_kitts_premiums.chatoro_prem_rate)
↳# simulate returns for St. Kitts and Nevis using Chatoro premium
display(st_kitts_bond_sim.loss_metrics)
print(f"Benchmark Sharpe Ratio premium rate: {round(st_kitts_premiums.benchmark_prem_
↳rate*100,1)}%")
print(f"Chatoro premium rate: {round(st_kitts_premiums.chatoro_prem_rate*100,1)}%")
print(f"IBRD premium rate: {round(st_kitts_premiums.ibrd_prem_rate*100,1)}%")

```

```

2026-05-01 13:21:00,529 - climada_petals - INFO - Expected Loss = 0.04244080553734771
2026-05-01 13:21:00,529 - climada_petals - INFO - Attachment Probability = 0.
↳0940940940940941
2026-05-01 13:21:00,589 - climada_petals - INFO - Using default GCIndex value of 180
2026-05-01 13:21:00,590 - climada_petals - INFO - Using default BBSpread value of 1.6
2026-05-01 13:21:00,591 - climada_petals - INFO - Calculated Chatoro premium rate: 0.
↳1200875106245345
2026-05-01 13:21:00,661 - climada_petals - INFO - Fitted IBRD premium parameters: a=4.
↳0575743650981355, k=0.6461612968444108, b=1.5306263398727946
2026-05-01 13:21:00,663 - climada_petals - INFO - Calculated IBRD premium rate: 0.
↳07605424523337957

```

```

{'Expected_annual_loss': 0.04244080553734771,
'Annual_attachment_probability': 0.0940940940940941,
'Total_payout': 28516534606.369743,
'Total_damages': 284652715459.8426,
'Annual_Value_at_risk_95': 0.3573700925824105,
'Annual_Expected_shortfall_95': 0.6654718898374853,
'Annual_Value_at_risk_99': 0.9835766088333577,
'Annual_Expected_shortfall_99': 0.9985392480081969}

```

```

Benchmark Sharpe Ratio premium rate: 13.9%
Chatoro premium rate: 12.0%
IBRD premium rate: 7.6%

```

```

### JAMAICA BOND SIMULATION ###
jamaica_bond_sim = SingleCountryBondSimulation(subarea_calc=jamaica_sub_calc,
↳term=term, start_year=start_year, number_of_terms=num_of_terms) # initialize bond

```

(continues on next page)

(continued from previous page)

```

↳simulation for Jamaica
jamaica_bond_sim.init_loss_simulation() # derive loss simulation for Jamaica
jamaica_premiums = PremiumCalculations(bond_simulation_class=jamaica_bond_sim) #
↳initialize premium calculations for Jamaica
jamaica_premiums.calc_chatoro_premium(peak_multi=peak_peril, investment_
↳graded=investment_graded, hybrid_trigger=hybrid_trigger) # derive Chatoro premium
↳for Jamaica
jamaica_premiums.calc_ibrd_premium() # derive IBRD premium for Jamaica
jamaica_premiums.calc_benchmark_premium(target_sharpe = target_sharpe) # derive
↳benchmark premium for Jamaica using target sharpe ratio
jamaica_bond_sim.init_return_simulation(premium=jamaica_premiums.chatoro_prem_rate) #
↳simulate returns for Jamaica using Chatoro premium
display(jamaica_bond_sim.loss_metrics)
print(f"Benchmark Sharpe Ratio premium rate: {round(jamaica_premiums.benchmark_prem_
↳rate*100,1)}%")
print(f"Chatoro premium rate: {round(jamaica_premiums.chatoro_prem_rate*100,1)}%")
print(f"IBRD premium rate: {round(jamaica_premiums.ibrd_prem_rate*100,1)}%")

```

```

2026-05-01 13:21:29,906 - climada_petals - INFO - Expected Loss = 0.051824053385856
2026-05-01 13:21:29,908 - climada_petals - INFO - Attachment Probability = 0.
↳07007007007007007
2026-05-01 13:21:29,910 - climada_petals - INFO - Using default GCIndex value of 180
2026-05-01 13:21:29,920 - climada_petals - INFO - Using default BBSpread value of 1.6
2026-05-01 13:21:29,921 - climada_petals - INFO - Calculated Chatoro premium rate: 0.
↳1332109210654582
2026-05-01 13:21:29,989 - climada_petals - INFO - Fitted IBRD premium parameters: a=4.
↳0575743650981355, k=0.6461612968444108, b=1.5306263398727946
2026-05-01 13:21:29,991 - climada_petals - INFO - Calculated IBRD premium rate: 0.
↳0867106178613396

```

```

{'Expected_annual_loss': 0.051824053385856,
 'Annual_attachment_probability': 0.07007007007007007,
 'Total_payout': 1993399612276.2878,
 'Total_damages': 11806709656203.236,
 'Annual_Value_at_risk_95': 0.4542079579357388,
 'Annual_Expected_shortfall_95': 0.928143105093355,
 'Annual_Value_at_risk_99': 1.0,
 'Annual_Expected_shortfall_99': 1}

```

```

Benchmark Sharpe Ratio premium rate: 18.2%
Chatoro premium rate: 13.3%
IBRD premium rate: 8.7%

```

```

### SAMOA BOND SIMULATION ###
samoa_bond_sim = SingleCountryBondSimulation(subarea_calc=samoa_sub_calc, term=term,
↳start_year=start_year, number_of_terms=num_of_terms) # initialize bond simulation
↳for Samoa
samoa_bond_sim.init_loss_simulation() # derive loss simulation for Samoa
samoa_premiums = PremiumCalculations(bond_simulation_class=samoa_bond_sim) #
↳initialize premium calculations for Samoa
samoa_premiums.calc_chatoro_premium(peak_multi=peak_peril, investment_

```

(continues on next page)

(continued from previous page)

```

↳graded=investment_graded, hybrid_trigger=hybrid_trigger) # derive Chatoro premium
↳for Samoa
samoa_premiums.calc_ibrd_premium() # derive IBRD premium for Samoa
samoa_premiums.calc_benchmark_premium(target_sharpe = target_sharpe) # derive
↳benchmark premium for Samoa using target sharpe ratio
samoa_bond_sim.init_return_simulation(premium=samoa_premiums.chatoro_prem_rate) #
↳simulate returns for Samoa using Chatoro premium
display(samoa_bond_sim.loss_metrics)
print(f"Benchmark Sharpe Ratio premium rate: {round(samoa_premiums.benchmark_prem_
↳rate*100,1)}%")
print(f"Chatoro premium rate: {round(samoa_premiums.chatoro_prem_rate*100,1)}%")
print(f"IBRD premium rate: {round(samoa_premiums.ibrd_prem_rate*100,1)}%")

```

```

2026-05-01 13:21:40,582 - climada_petals - INFO - Expected Loss = 0.04468195241318505
2026-05-01 13:21:40,584 - climada_petals - INFO - Attachment Probability = 0.
↳06306306306306306
2026-05-01 13:21:40,587 - climada_petals - INFO - Using default GCIndex value of 180
2026-05-01 13:21:40,587 - climada_petals - INFO - Using default BBSpread value of 1.6
2026-05-01 13:21:40,587 - climada_petals - INFO - Calculated Chatoro premium rate: 0.
↳12322197864508061
2026-05-01 13:21:40,663 - climada_petals - INFO - Fitted IBRD premium parameters: a=4.
↳0575743650981355, k=0.6461612968444108, b=1.5306263398727946
2026-05-01 13:21:40,663 - climada_petals - INFO - Calculated IBRD premium rate: 0.
↳0784958663906661

```

```

{'Expected_annual_loss': 0.04468195241318505,
 'Annual_attachment_probability': 0.06306306306306306,
 'Total_payout': 48491657276.9485,
 'Total_damages': 284830602028.7631,
 'Annual_Value_at_risk_95': 0.3407105992182873,
 'Annual_Expected_shortfall_95': 0.8423344605469362,
 'Annual_Value_at_risk_99': 1.0,
 'Annual_Expected_shortfall_99': 1}

```

```

Benchmark Sharpe Ratio premium rate: 17.4%
Chatoro premium rate: 12.3%
IBRD premium rate: 7.8%

```

5.7 Multi-Country CAT Bond Simulation

Now that we have created 3 different single-country CAT bonds (Jamaica, St. Kitts and Nevis, Samoa) we can construct a multi-country CAT bond. To do so, we first create a dictionary of the single-country CAT bonds using the country codes as dictionary keys.

```

#create dictionary of single-country bond simulation classes
sng_bonds_dic = {countries[0]: st_kitts_bond_sim,
                 countries[1]: jamaica_bond_sim,
                 countries[2]: samoa_bond_sim}

```

For the construction of multi-country CAT bonds we have two different options:

1. We manually choose for which countries the multi-country CAT bond should be created.

2. We derive optimal risk diverse pools, and create a CAT bond for each pool. Pools can be derived under the following constraints:

- The number of pools which are created is fixed
- A defined maximum principal of a CAT bond can not be exceeded

For the first option we build a multi-country CAT bond from all three single-country CAT bonds:

1. Initialize an instance of the multi-country CAT bond class with the term of the bond in years, the start year of the bond simulation and the the number of terms to be simulated.
2. Calculate the minimum principal of the multi-country CAT bond which is sufficient to cover all payouts which were made during all terms of the single-country CAT bonds -> `init_required_principal()`
3. Simulate the losses of the multi-country CAT bond -> `init_loss_simulation()`
4. Initialize a `PremiumCalculations` class and derive the desired premiums -> `PremiumCalculations` -> `calc_chatoro_premium()` or `calc_ibrd_premium()` or `calc_benchmark_premium()`
5. Simulate the net cash flow and premium payments of the multi-country CAT bond with the desired annual premium rate -> `init_return_simulation()`

```
#initialize multi-country bond simulation class
mlt_cat_bond = MultiCountryBond(country_dictionary=sng_bonds_dic, term=term, start_
↳year=start_year, number_of_terms=num_of_terms)
# calculate minimal required principal to achieve the same coverage as the single-
↳country bonds
mlt_cat_bond.init_required_principal()
mlt_cat_bond.init_loss_simulation(principal=mlt_cat_bond.requ_principal, confidence_
↳levels=[0.95,0.99]) # derive loss simulation for multi-country bond
mlt_bond_premiums = PremiumCalculations(bond_simulation_class=mlt_cat_bond) #↳
↳initialize premium calculations for multi-country bond
mlt_bond_premiums.calc_chatoro_premium(peak_multi=peak_peril, investment_
↳graded=investment_graded, hybrid_trigger=hybrid_trigger) # derive Chatoro premium_
↳for multi-country bond
mlt_bond_premiums.calc_ibrd_premium() # derive IBRD premium for multi-country bond
mlt_bond_premiums.calc_benchmark_premium(target_sharpe = target_sharpe) # derive_
↳benchmark premium for multi-country bond using target sharpe ratio
mlt_cat_bond.init_return_simulation(premium=mlt_bond_premiums.chatoro_prem_rate) #↳
↳simulate returns for multi-country bond using Chatoro premium
display(mlt_cat_bond.loss_metrics)
print(f"Benchmark Sharpe Ratio premium rate: {round(mlt_bond_premiums.benchmark_prem_
↳rate*100,1)}")
print(f"Chatoro premium rate: {round(mlt_bond_premiums.chatoro_prem_rate*100,1)}")
print(f"IBRD premium rate: {round(mlt_bond_premiums.ibrd_prem_rate*100,1)}")
print(f"The required principal for the multi-country CAT bond is {round(mlt_cat_bond.
↳requ_principal,2)} million USD.")
```

```
2026-05-01 13:22:05,876 - climada_petals - INFO - Expected Loss = 0.05290199205161236
2026-05-01 13:22:05,879 - climada_petals - INFO - Attachment Probability = 0.
↳17017017017017017
2026-05-01 13:22:05,928 - climada_petals - INFO - Using default GCIndex value of 180
2026-05-01 13:22:05,934 - climada_petals - INFO - Using default BBSpread value of 1.6
2026-05-01 13:22:05,938 - climada_petals - INFO - Calculated Chatoro premium rate: 0.
↳13471852608338505
2026-05-01 13:22:06,060 - climada_petals - INFO - Fitted IBRD premium parameters: a=4.
```

(continues on next page)

(continued from previous page)

```

↪0575743650981355, k=0.6461612968444108, b=1.5306263398727946
2026-05-01 13:22:06,064 - climada_petals - INFO - Calculated IBRD premium rate: 0.
↪08800682254615821

```

```

{'Expected_annual_loss': 0.05290199205161236,
 'Annual_attachment_probability': 0.17017017017017017,
 'Total_payout': 2070407804159.607,
 'Total_damages': 12376192973691.842,
 'Annual_Value_at_risk_95': 0.4491547597685044,
 'Annual_Expected_shortfall_95': 0.921940279565346,
 'Annual_Value_at_risk_99': 0.9997180370170589,
 'Annual_Expected_shortfall_99': 0.9999749212580538}

```

```

Benchmark Sharpe Ratio premium rate: 18.3
Chatoro premium rate: 13.5
IBRD premium rate: 8.8
The required principal for the multi-country CAT bond is 39175845824.9 million USD.

```

5.7.1 Compare the Single-Country CAT Bonds to the Multi-Country CAT Bond

Now that we have created a multi-country CAT bond, we can compare it to the previously generated single-country bonds. One very important question for the countries would be: How much coverage do I get for what price?

```

print(f"Coverage single-country bond Jamaica: {jamaica_bond_sim.loss_metrics['Total_
↪payout'] / jamaica_bond_sim.loss_metrics['Total_damages'] * 100:.2f}%")
print(f"Coverage single-country bond Samoa: {samoa_bond_sim.loss_metrics['Total_payout
↪'] / samoa_bond_sim.loss_metrics['Total_damages'] * 100:.2f}%")
print(f"Coverage single-country bond St. Kitts and Nevis: {st_kitts_bond_sim.loss_
↪metrics['Total_payout'] / st_kitts_bond_sim.loss_metrics['Total_damages'] * 100:.2f}
↪%")
print(f"Coverage multi-country bond: Jamaica: {mlt_cat_bond.tot_coverage_
↪cty[countries[1]]['coverage'] * 100:.2f}%")
print(f"Coverage multi-country bond: Samoa: {mlt_cat_bond.tot_coverage_
↪cty[countries[2]]['coverage'] * 100:.2f}%")
print(f"Coverage multi-country bond: St. Kitts and Nevis: {mlt_cat_bond.tot_coverage_
↪cty[countries[0]]['coverage'] * 100:.2f}%")

```

```

Coverage single-country bond Jamaica: 16.88%
Coverage single-country bond Samoa: 17.02%
Coverage single-country bond St. Kitts and Nevis: 10.02%
Coverage multi-country bond: Jamaica: 16.88%
Coverage multi-country bond: Samoa: 17.02%
Coverage multi-country bond: St. Kitts and Nevis: 10.02%

```

The coverage per country does not change between single- and multi-country CAT bonds. This is intended as the calculated minimum principal should cover all payouts equally as the single-country CAT bonds.

```

print(f"The multi-country CAT bond principal is {(1-((mlt_cat_bond.requ_principal)/
↪(st_kitts_sub_calc.principal + jamaica_sub_calc.principal + samoa_sub_calc.
↪principal)))*100:.2f}% lower than the total principal of the individual CAT bonds.")
print(f"In the case of the multi country CAT bond the countries have to pay

```

(continues on next page)

(continued from previous page)

```

↪(((jamaica_bond_sim.return_metrics['total_premiums'] + samoa_bond_sim.return_
↪metrics['total_premiums'] + st_kitts_bond_sim.return_metrics['total_premiums']) -
↪mlt_cat_bond.premiums['Total_premiums'].sum() * mlt_cat_bond.requ_principal):.2f}
↪USD less premiums compared to individual CAT bonds.")
print(f"Premium savings Jamaica: {100 - ((mlt_cat_bond.premiums[countries[1]].sum() *
↪mlt_cat_bond.requ_principal) / jamaica_bond_sim.return_metrics['total_premiums'] *
↪100):.2f}%")
print(f"Premium savings Samoa: {100 - ((mlt_cat_bond.premiums[countries[2]].sum() *
↪mlt_cat_bond.requ_principal) / samoa_bond_sim.return_metrics['total_premiums'] *
↪100):.2f}%")
print(f"Premium savings St. Kitts and Nevis: {100 - ((mlt_cat_bond.
↪premiums[countries[0]].sum() * mlt_cat_bond.requ_principal) / st_kitts_bond_sim.
↪return_metrics['total_premiums'] * 100):.2f}%")
print(f"Sharpe ratios multi-country bond: {round(mlt_cat_bond.sharpe_ratio, 2)} vs
↪sharpe ratios single-country bonds: {round(st_kitts_bond_sim.return_metrics['sharpe_
↪ratio'], 2)} (St. Kitts and Nevis), {round(jamaica_bond_sim.return_metrics['sharpe_
↪ratio'], 2)} (Jamaica), {round(samoa_bond_sim.return_metrics['sharpe_ratio'], 2)}
↪(Samoa)")

```

```

The multi-country CAT bond principal is 2.70% lower than the total principal of the
↪individual CAT bonds.
In the case of the multi country CAT bond the countries have to pay 74085644190.41
↪USD less premiums compared to individual CAT bonds.
Premium savings Jamaica: 1.21%
Premium savings Samoa: 7.32%
Premium savings St. Kitts and Nevis: 12.90%
Sharpe ratios multi-country bond: 0.31 vs sharpe ratios single-country bonds: 0.4 (St.
↪ Kitts and Nevis), 0.31 (Jamaica), 0.31 (Samoa)

```

Yet, the required principal needed to achieve this coverage as well as resulting premium payments are overall significantly lower. Using the Chatoro-Pricing approach and the implemented premium sharing mechanism where the share of the premium per country is based on the countries share of the marginal expected loss, St.Kitts and Nevis saves the most premiums (12.9%), while Samoa can save 7.3% and Jamaica 1.2%. In total over the simulation period almost 74 billion USD in premium payments can be saved. However, the sharpe ratio which is achieved by investors is overall at the lower end for the multi-country CAT bond compared to the single-country CAT bonds. This could be adjusted by raising the premiums for which we have a significant margin before losing the advantage of premium savings for the selected countries.

5.7.2 Create a CAT Bond which is Issued with Multiple Tranches

CAT bonds are often issued as tranches where each tranche covers a different level of risk. This can attract a greater number of investors as they can choose the most suitable product for themselves. In this module we implemented tranches by splitting the principal into different levels. By calling `init_return_simulation_tranches()` we can insert a annual premium rate per tranche and the share of the principal it covers. For example we can set two tranches, specifying that the first tranche covers 20% of the principal and the second tranche the remaining 80%. The first tranche then covers the first 20% of the principal. So all occurring losses are covered by the first tranche until the specified share of principal (20%) is depleted. Then all losses which could not be paid for by the first tranche will be paid by the second tranche.

```

mlt_cat_bond.init_return_simulation_tranches(premiums=[mlt_bond_premiums.chatoro_prem_
↪rate,mlt_bond_premiums.chatoro_prem_rate], tranches=[0.2,0.8]) # simulate returns
↪for multi-country bond tranches using Chatoro premium and 20%-80% tranche structure
print(f"Sharpe ratio tranche 1 (20%): {round(mlt_cat_bond.sharpe_ratio_tranches[0],

```

(continues on next page)

(continued from previous page)

```

    ↪2) ;")
print(f"Sharpe ratio tranche 2 (80%): {round(mlt_cat_bond.sharpe_ratio_tranches[1], 2)}")
    ↪2) ;")

```

```

Sharpe ratio tranche 1 (20%): 0.24
Sharpe ratio tranche 2 (80%): 0.32

```

By comparing the sharpe ratios of each tranche we can see that the second tranche offers a much better deal of investors. This is not surprising as we set the premiums to be equal in both tranches, but the second tranche is much more unlikely to be triggered. The first tranche will loose money as soon as the bond gets triggerd but the second tranche does not loose anything as long as the 20% of the principal, which are covered by the first tranche, are not depleted. Consequently, the second tranche will have better returns and a reduced volatility which results in a much higher sharpe ratio.

5.7.3 Construct Risk Diverse Pools for Multi-Country CAT Bond Simulation

For the second option we will use a wrapper function which first derives optimal risk diverse pools under the constraint that two pools have to be created (using the three previously calculated single-country CAT bonds). The wrapper function then creates a multi-country CAT bond class instance for each pool, derives the required principal, and simulates the losses. Premiums and returns can be calculated the same way as shown before or by choosing manually an annual premium rate.

```

pool_n_dic, pool_allocation_n_pool, algorithm_result_n_pool = MultiCountryBond.
    ↪simulate_bond_pool_n(country_dictionary=sng_bonds_dic,
    ↪
    ↪         term=term,
    ↪
    ↪         start_year=start_year,
    ↪
    ↪         number_of_terms=num_of_terms,
    ↪
    ↪         number_pools=2,
    ↪
    ↪         n_opt_rep=2) # simulate multiple multi-country bond pools based on a
    ↪fixed number of pools optimization approach
print(f"Countries in pool 1: {pool_n_dic[1].countries}")
display(pool_n_dic[1].loss_metrics)
print("-----")
print(f"Countries in pool 2: {pool_n_dic[2].countries}")
display(pool_n_dic[2].loss_metrics)

```

```

2026-05-01 13:24:19,509 - climada_petals - INFO - Starting pooling optimization for 2
    ↪pools and 3 countries.
2026-05-01 13:25:24,045 - climada_petals - INFO - Completed pooling optimization.
{1: [659], 2: [388, 822]}
1 [659]
2026-05-01 13:25:24,045 - climada_petals - INFO - Pool 1: Countries [659]
2026-05-01 13:25:34,842 - climada_petals - INFO - Expected Loss = 0.042440805537347706
2026-05-01 13:25:34,842 - climada_petals - INFO - Attachment Probability = 0.
    ↪0940940940940941
2026-05-01 13:25:34,842 - climada_petals - INFO - Completed loss simulation for pool
    ↪1.
2 [388, 822]
2026-05-01 13:25:34,852 - climada_petals - INFO - Pool 2: Countries [388, 822]

```

(continues on next page)

(continued from previous page)

```

2026-05-01 13:25:56,022 - climada_petals - INFO - Expected Loss = 0.053084730984071994
2026-05-01 13:25:56,022 - climada_petals - INFO - Attachment Probability = 0.
↳13313313313313313
2026-05-01 13:25:56,040 - climada_petals - INFO - Completed loss simulation for pool↳
↳2.
Countries in pool 1: [659]

```

```

{'Expected_annual_loss': 0.042440805537347706,
 'Annual_attachment_probability': 0.0940940940940941,
 'Total_payout': 28516534606.36974,
 'Total_damages': 284652715459.84265,
 'Annual_Value_at_risk_95': 0.3573700925824104,
 'Annual_Expected_shortfall_95': 0.6654718898374853,
 'Annual_Value_at_risk_99': 0.9835766088333576,
 'Annual_Expected_shortfall_99': 0.9985392480081968}

```

```

-----
Countries in pool 2: [388, 822]

```

```

{'Expected_annual_loss': 0.053084730984071994,
 'Annual_attachment_probability': 0.13313313313313313,
 'Total_payout': 2041891269553.2368,
 'Total_damages': 12091540258232.0,
 'Annual_Value_at_risk_95': 0.4542079579357387,
 'Annual_Expected_shortfall_95': 0.9281431050933547,
 'Annual_Value_at_risk_99': 0.9999999999999998,
 'Annual_Expected_shortfall_99': nan}

```

As an alternative to a fixed number of pools we can set a maximum principal. The maximum principal is applied to each multi-country CAT bond so that the sum of the principal of the single-country CAT bonds does not exceed the maximum principal per pool. We will use a wrapper function which first derives optimal risk diverse pool under the the maximum principal constraint. The wrapper function then creates a multi-country CAT bond class instance for each pool, derives the required principal, and simulates the losses. Premiums and returns can be calculated the same way as shown before or by choosing manually an annual premium rate.

```

pool_max_principal_dic, pool_allocation_max_prin, algorithm_result_max_prin =↳
↳MultiCountryBond.simulate_bond_max_principal_pool(country_dictionary=sng_bonds_dic,
↳
↳                                     term=term,
↳
↳                                     start_year=start_year,
↳
↳                                     number_of_terms=num_of_terms,
↳
↳                                     maximum_principal=70000000000,
↳
↳                                     n_opt_rep=2) # simulate multiple multi-
↳country bond pools based on a maximum principal optimization approach
print(f"Countries in pool 1: {pool_max_principal_dic[1].countries}")
display(pool_max_principal_dic[1].loss_metrics)

```

```

2026-05-01 13:25:56,835 - climada_petals - INFO - Starting pooling optimization for
↳pools with a maximum principal of 70000000000 and 3 countries.
2026-05-01 13:27:24,279 - climada_petals - INFO - Completed pooling optimization.
2026-05-01 13:27:24,279 - climada_petals - INFO - Pool 2: Countries [659]
2026-05-01 13:27:37,960 - climada_petals - INFO - Expected Loss = 0.042440805537347706
2026-05-01 13:27:37,963 - climada_petals - INFO - Attachment Probability = 0.
↳0940940940940941
2026-05-01 13:27:37,969 - climada_petals - INFO - Completed loss simulation for pool
↳2.
2026-05-01 13:27:37,971 - climada_petals - INFO - Pool 1: Countries [388, 822]
2026-05-01 13:28:00,327 - climada_petals - INFO - Expected Loss = 0.053084730984071994
2026-05-01 13:28:00,330 - climada_petals - INFO - Attachment Probability = 0.
↳13313313313313313
2026-05-01 13:28:00,339 - climada_petals - INFO - Completed loss simulation for pool
↳1.
Countries in pool 1: [388, 822]

```

```

{'Expected_annual_loss': 0.053084730984071994,
 'Annual_attachment_probability': 0.13313313313313313,
 'Total_payout': 2041891269553.2368,
 'Total_damages': 12091540258232.0,
 'Annual_Value_at_risk_95': 0.4542079579357387,
 'Annual_Expected_shortfall_95': 0.9281431050933547,
 'Annual_Value_at_risk_99': 0.9999999999999998,
 'Annual_Expected_shortfall_99': nan}

```

As we can see only one pool and therefore also only one bond was created. This is because the maximum principal is higher than the sum of all three principals of the single-country CAT bonds. As all countries together in one pool achieve the highest risk diversification the countries are not splitted into different pools.

5.8 Earthquake CAT Bond Simulation

In this section we create a custom earthquake CAT bond for Belize. By doing so we demonstrate the frameworks ability to simulate CAT bonds for a variety of underlying data sets irrespective of the chosen hazard.

To construct an earthquake bond for Belize we perform the following steps:

1. Download hazard and exposure from the CLIMADA API together with an dummy MMI-based earthquake impact function.
2. Create subareas from the exposure grid using `Subareas`.
3. Calibrate payout functions per subarea using `SubareaCalculations`.
4. Simulate losses with `SingleCountryBondSimulation`.
5. Calculate premiums with `PremiumCalculations`.
6. Simulate returns.

```

hazard_belize = client.get_hazard(
    "earthquake",
    properties={
        "country_name": "Belize",
    },

```

(continues on next page)

(continued from previous page)

```

)
# Limit hazard to events between 1980 and 2020
hazard_belize = hazard_belize.select(date=(date(1980, 1, 1).toordinal(), date(2020, 1,
↳12, 31).toordinal()))

# Get exposure for Belize
exp_belize = client.get_litpop('BLZ')
exp_belize.gdf["impf_EQ"] = 1 # linkt to impact function
impfset_eq = create_earthquake_mmi_impact_function() # create impact function for_
↳earthquake MMI

# --- Subareas ---
eq_subareas = Subareas.from_resolution(
    hazard_belize, impfset_eq, exp_belize, resolution=0.5
)

# --- Subarea Calculations (payout calibration) ---
eq_sub_calc = SubareaCalculations(subareas=eq_subareas, index_stat=100)
eq_sub_calc.create_pay_vs_dam(
    attachment_point=0.001,
    exhaustion_point=0.025,
    methods_attachment_point="Exposure_Share",
    methods_exhaustion_point="Exposure_Share",
)

# --- Bond Simulation ---
eq_bond_sim = SingleCountryBondSimulation(
    subarea_calc=eq_sub_calc, term=term, start_year=start_year, number_of_terms=num_
↳of_terms
)
eq_bond_sim.init_loss_simulation()

# --- Premiums ---
eq_premiums = PremiumCalculations(bond_simulation_class=eq_bond_sim)
eq_premiums.calc_chatoro_premium(
    peak_multi=peak_peril, investment_graded=investment_graded, hybrid_trigger=hybrid_
↳trigger
)
eq_premiums.calc_ibrd_premium()
eq_premiums.calc_benchmark_premium(target_sharpe=target_sharpe)

# --- Return Simulation ---
eq_bond_sim.init_return_simulation(premium=eq_premiums.chatoro_prem_rate)

# --- Display Results ---
display(eq_bond_sim.loss_metrics)
print(f"Benchmark Sharpe Ratio premium rate: {round(eq_premiums.benchmark_prem_
↳rate*100, 1)}%")
print(f"Chatoro premium rate: {round(eq_premiums.chatoro_prem_rate*100, 1)}%")
print(f"IBRD premium rate: {round(eq_premiums.ibrd_prem_rate*100, 1)}%")

```

```
2026-05-01 13:28:01,410 - climada.hazard.io - INFO - Reading C:\Users\Kai\climada\  
↳data\hazard\earthquake\earthquake_hist_above4_84\v1.1\earthquake_hist_above4_84.hdf5  
2026-05-01 13:28:02,921 - climada.entity.exposures.base - INFO - Reading C:\Users\Kai\  
↳climada\data\exposures\litpop\LitPop_150arcsec_BLZ\v3\LitPop_150arcsec_BLZ.hdf5  
2026-05-01 13:28:03,031 - climada_petals - INFO - Rasterizing exposure with width: 28,  
↳ height: 51, resolution: 0.05  
2026-05-01 13:28:03,286 - climada_petals - INFO - Exposure perimeter polygon created.  
2026-05-01 13:28:03,286 - climada_petals - INFO - Number of polygons in exposure_  
↳perimeter: 7  
2026-05-01 13:28:03,741 - climada_petals - INFO - Polygon smaller than resolution;_  
↳adding polygon bounding box.  
2026-05-01 13:28:03,750 - climada_petals - INFO - Polygon smaller than resolution;_  
↳adding polygon bounding box.  
2026-05-01 13:28:03,752 - climada_petals - INFO - Polygon smaller than resolution;_  
↳adding polygon bounding box.  
2026-05-01 13:28:03,766 - climada_petals - INFO - Polygon smaller than resolution;_  
↳adding polygon bounding box.  
2026-05-01 13:28:03,778 - climada_petals - INFO - Polygon smaller than resolution;_  
↳adding polygon bounding box.  
2026-05-01 13:28:03,786 - climada_petals - INFO - Polygon smaller than resolution;_  
↳adding polygon bounding box.  
2026-05-01 13:28:03,801 - climada_petals - INFO - Merging overlapping grid cells into_  
↳single polygons.  
2026-05-01 13:28:03,872 - climada_petals - INFO - Subareas created.  
2026-05-01 13:28:03,881 - climada_petals - INFO - Calculating impact and subarea-  
↳level impacts per event.  
2026-05-01 13:28:03,886 - climada.entity.exposures.base - INFO - Matching 1105_  
↳exposures with 1081 centroids.  
2026-05-01 13:28:03,899 - climada.util.coordinates - INFO - No exact centroid match_  
↳found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100  
2026-05-01 13:28:03,913 - climada.engine.impact_calc - INFO - Calculating impact for_  
↳3315 assets (>0) and 22671 events.  
2026-05-01 13:28:04,005 - climada_petals - INFO - Calculating parametric index per_  
↳subarea.
```

```
c:\Users\Kai\miniforge3\envs\climada_env\Lib\site-packages\shapely\set_operations.  
↳py:133: RuntimeWarning: invalid value encountered in intersection  
return lib.intersection(a, b, **kwargs)
```

```
2026-05-01 13:28:04,983 - climada_petals - INFO - The attachment point and the_  
↳principal of the CAT bond is: 4009642.747 and 100241068.675 [USD], respectively.  
2026-05-01 13:28:04,985 - climada_petals - INFO - Calibrating payout functions by_  
↳optimizing trigger thresholds to minimize basis risk.  
2026-05-01 13:28:06,420 - climada_petals - INFO - Calculating pay-versus-damage table_  
↳for all events using optimized payout functions.  
2026-05-01 13:28:21,060 - climada_petals - INFO - Expected Loss = 0.  
↳0025115228318194657  
2026-05-01 13:28:21,062 - climada_petals - INFO - Attachment Probability = 0.  
↳003003003003003003  
2026-05-01 13:28:21,067 - climada_petals - INFO - Using default GCIndex value of 180  
2026-05-01 13:28:21,067 - climada_petals - INFO - Using default BBSpread value of 1.6  
2026-05-01 13:28:21,067 - climada_petals - INFO - Calculated Chatoro premium rate: 0.
```

(continues on next page)

(continued from previous page)

```
↪0642424158325827
2026-05-01 13:28:21,122 - climada_petals - INFO - Fitted IBRD premium parameters: a=4.
↪0575743650981355, k=0.6461612968444108, b=1.5306263398727946
2026-05-01 13:28:21,124 - climada_petals - INFO - Calculated IBRD premium rate: 0.
↪012508317611382458
```

```
{'Expected_annual_loss': 0.0025115228318194657,
'Annual_attachment_probability': 0.003003003003003003,
'Total_payout': 251505974.93058226,
'Total_damages': 325694382.5094174,
'Annual_Value_at_risk_95': 0.0,
'Annual_Expected_shortfall_95': 0.836337102995882,
'Annual_Value_at_risk_99': 0.0,
'Annual_Expected_shortfall_99': 0.836337102995882}
```

```
Benchmark Sharpe Ratio premium rate: 2.6%
Chatoro premium rate: 6.4%
IBRD premium rate: 1.3%
```


BLACKMARBLE CLASS

The `BlackMarble` class inherits the `Exposures` 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()

```
%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()`:

```
bm = BlackMarble()
bm.set_countries?
```

```
Signature:
bm.set_countries(
    countries,
    ref_year=2016,
```

(continues on next page)

(continued from previous page)

```

    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, ..., countrysn_ISO3_code: gdpn}` and `inc_grp = {country1_ISO3_code: inc_grp1, ..., countrysn_ISO3_code: inc_grp1}`

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.

```

# 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
↳', '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', 'Curaçao', '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', 'São 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>

```

(continues on next page)

(continued from previous page)

```

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
    96         cntry_info, cntry_admin1 = country_iso_geom(countries, shp_file,
---> 97                                     admin_key_dict[admin_
-> 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)
    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:

```

```

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.

```

```

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()

```

```

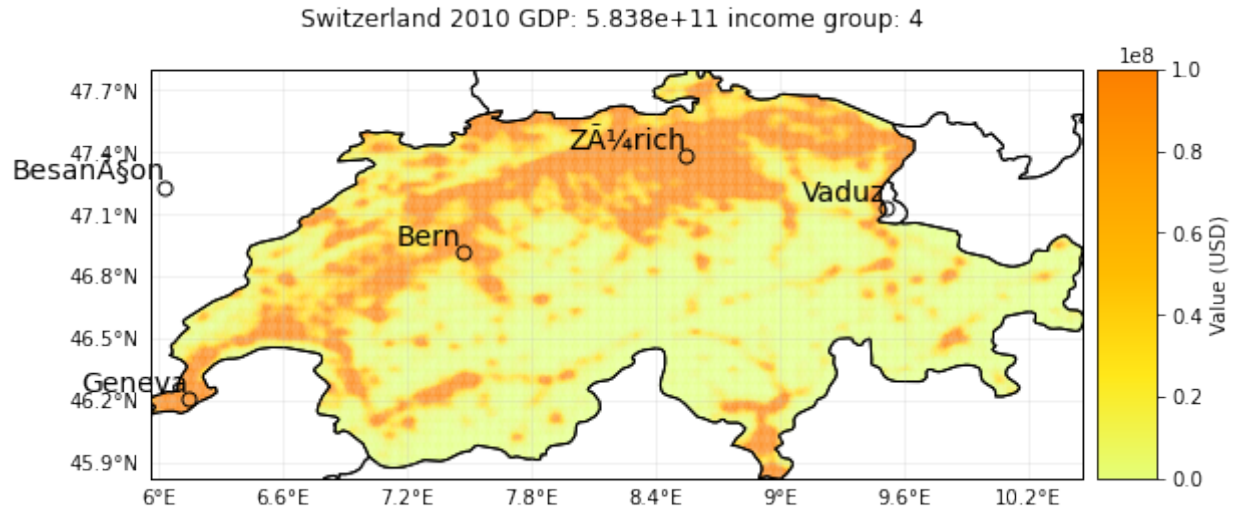
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x21408998408>

```

```

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:

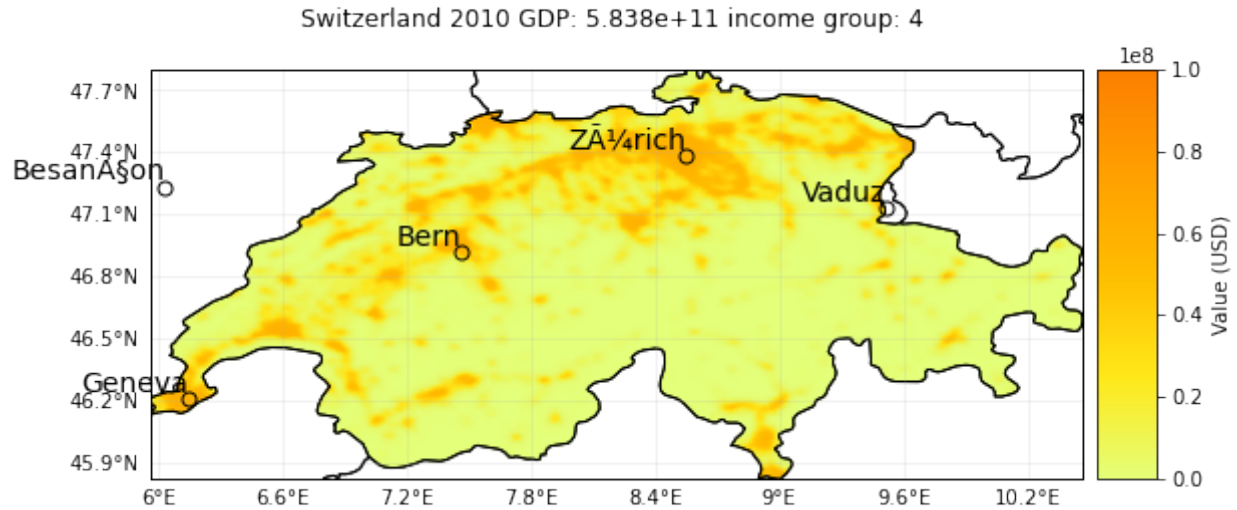
```
ch_2010_05 = BlackMarble()
ch_2010_05.set_countries(['Switzerland'], ref_year=2010, res_km=0.5)
ch_2010_05.plot_hexbin(vmax=1e8);
```

```
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()
```

```
<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`:

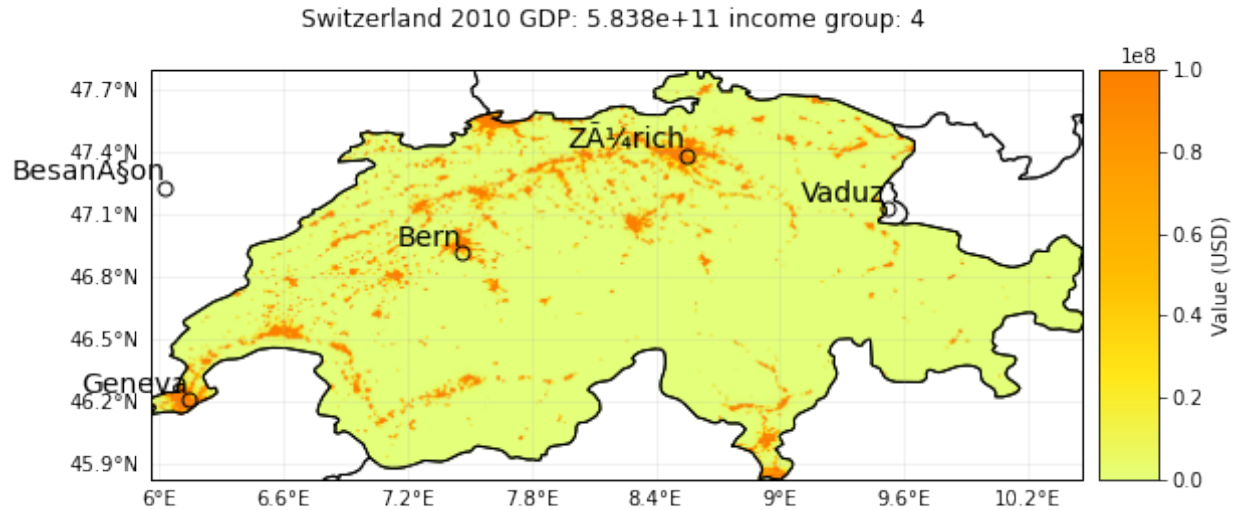
```
ch_2010_hr = BlackMarble()
ch_2010_hr.set_countries(['Switzerland'], ref_year=2010, from_hr=True)
ch_2010_hr.plot_hexbin(vmax=1e8);
```

```
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 to accommodate all axes decorations.
fig.tight_layout()
```

```
<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:

```
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);
```

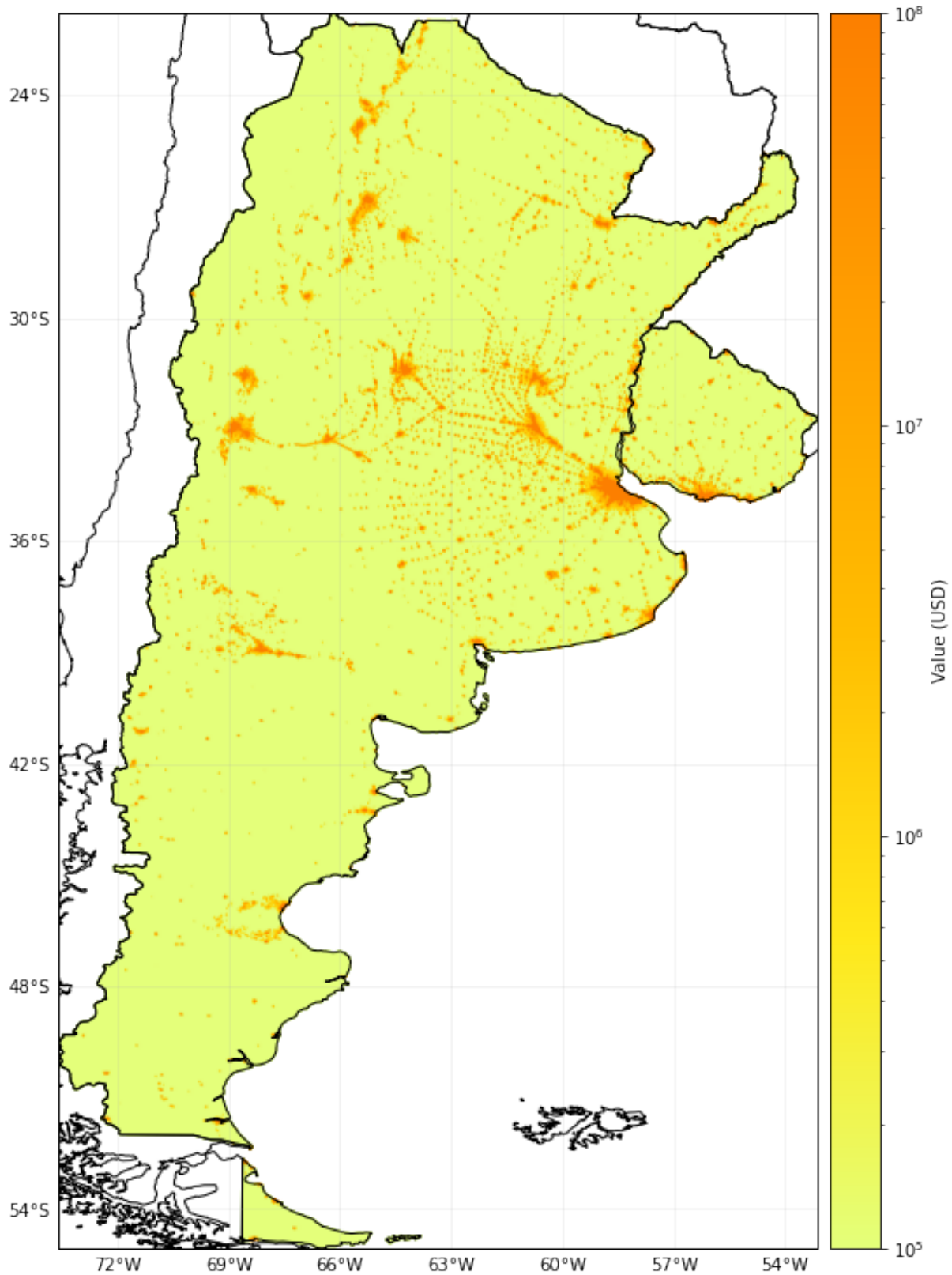
```
2021-02-15 10:36:27,233 - climada.util.finance - INFO - GDP ARG 2013: 5.520e+11.
2021-02-15 10:36:27,336 - climada.util.finance - INFO - Income group ARG 2013: 3.
2021-02-15 10:36:27,797 - climada.util.finance - INFO - GDP URY 2013: 5.753e+10.
2021-02-15 10:36:27,897 - climada.util.finance - INFO - 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()
```

```
<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))
```

Argentina 2013 GDP: 5.520e+11 income group: 3
 Uruguay 2013 GDP: 5.753e+10 income group: 4



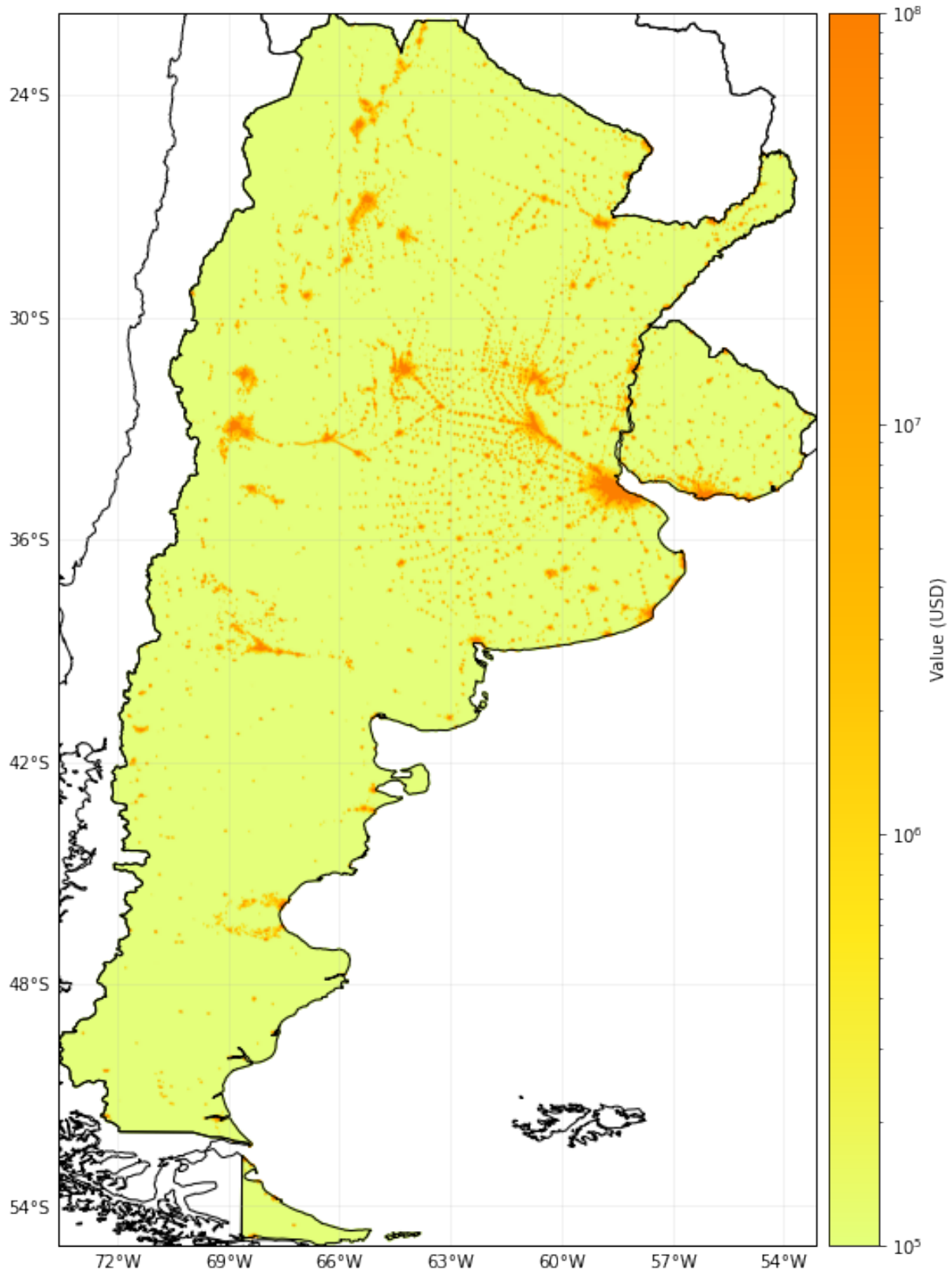
The user can apply her own gdp and income group values by passing a dictionary called *gdp*, with keys the countries iso-codes and values gdp values, and one called *inc_grp*, with keys the countries iso-codes and values income groups (from 1 to 4).

```
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.
```

```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x21450396f48>
```

Argentina 2013 GDP: 5.000e+11 income group: 3
 Uruguay 2013 GDP: 5.000e+10 income group: 4



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:

```
country_name = {'Germany': ['Berlin', 'Brandenburg', 'Bayern', 'Sachsen', 'Thüringen',
    ↳ 'Sachsen-Anhalt'],
                'Czechia': ['Prague', 'Karlovarský', 'Ústecký', 'Liberecký',
    ↳ 'Středočeský', 'Plzeňský', 'Jihočeský']}

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);
```

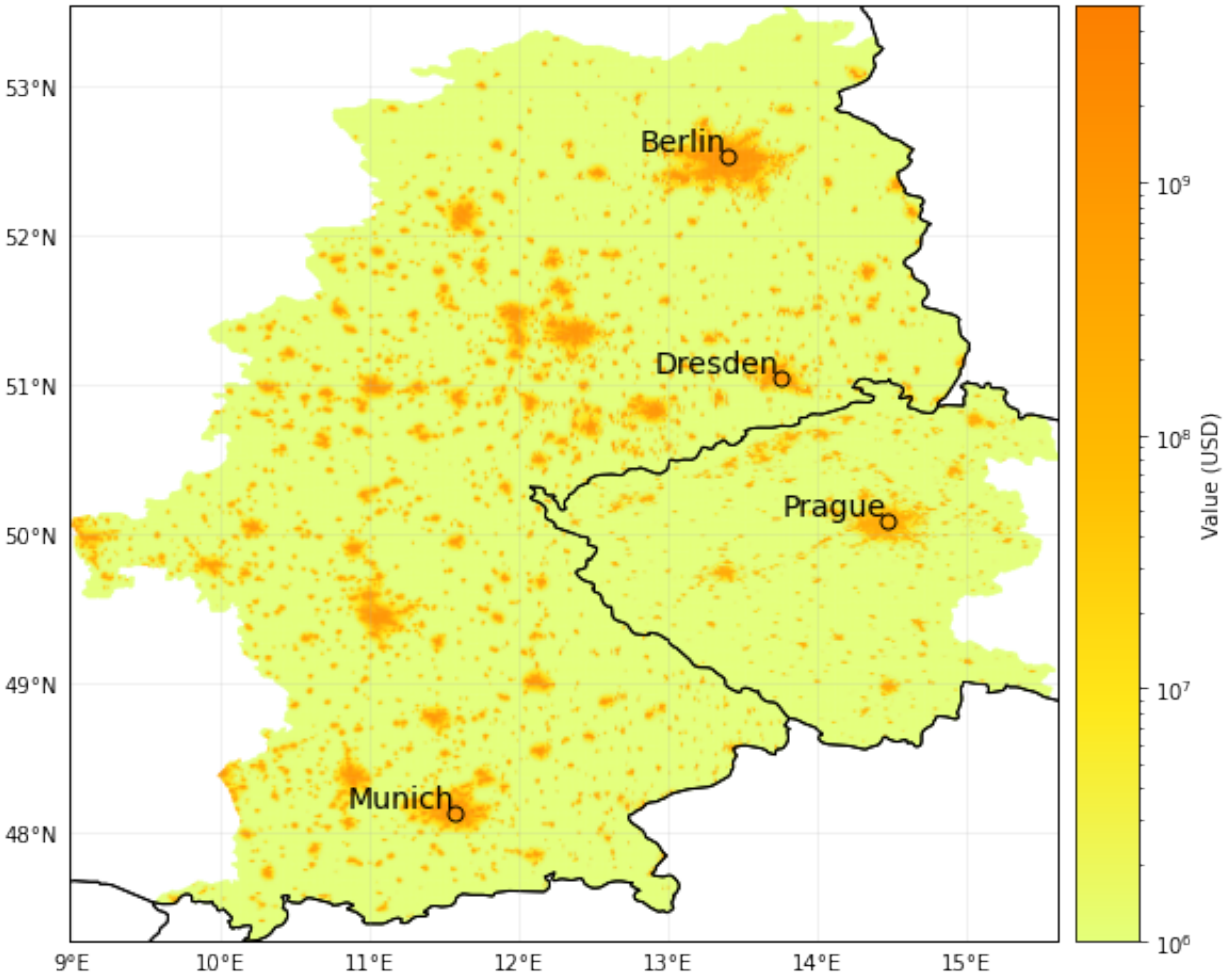
```
2021-02-15 10:39:42,488 - climada.util.finance - INFO - GDP DEU 2012: 3.527e+12.
2021-02-15 10:39:42,580 - climada.util.finance - INFO - Income group DEU 2012: 4.
2021-02-15 10:39:43,080 - climada.util.finance - INFO - GDP CZE 2012: 2.089e+11.
2021-02-15 10:39:43,174 - climada.util.finance - INFO - 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()
```

```
<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))
```

Germany 2012 GDP: 3.527e+12 income group: 4
 Czechia 2012 GDP: 2.089e+11 income group: 4



As above, if the region's name is misspelled, a list with all possible regions' names for that country is provided:

```
# 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-Württemberg',
↳ 'Brandenburg', 'Mecklenburg-Vorpommern', 'Bremen', 'Hamburg', 'Hessen', 'Thüringen
↳ ', '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)
```

(continues on next page)

(continued from previous page)

```

~\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
    96     cntry_info, cntry_admin1 = country_iso_geom(countries, shp_file,
--> 97                                     admin_key_dict[admin_
↪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.

```

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.

```

```

norm=colors.LogNorm(vmin=1.0e3, vmax=1.0e6)
fg.plot_hexbin(norm=norm);

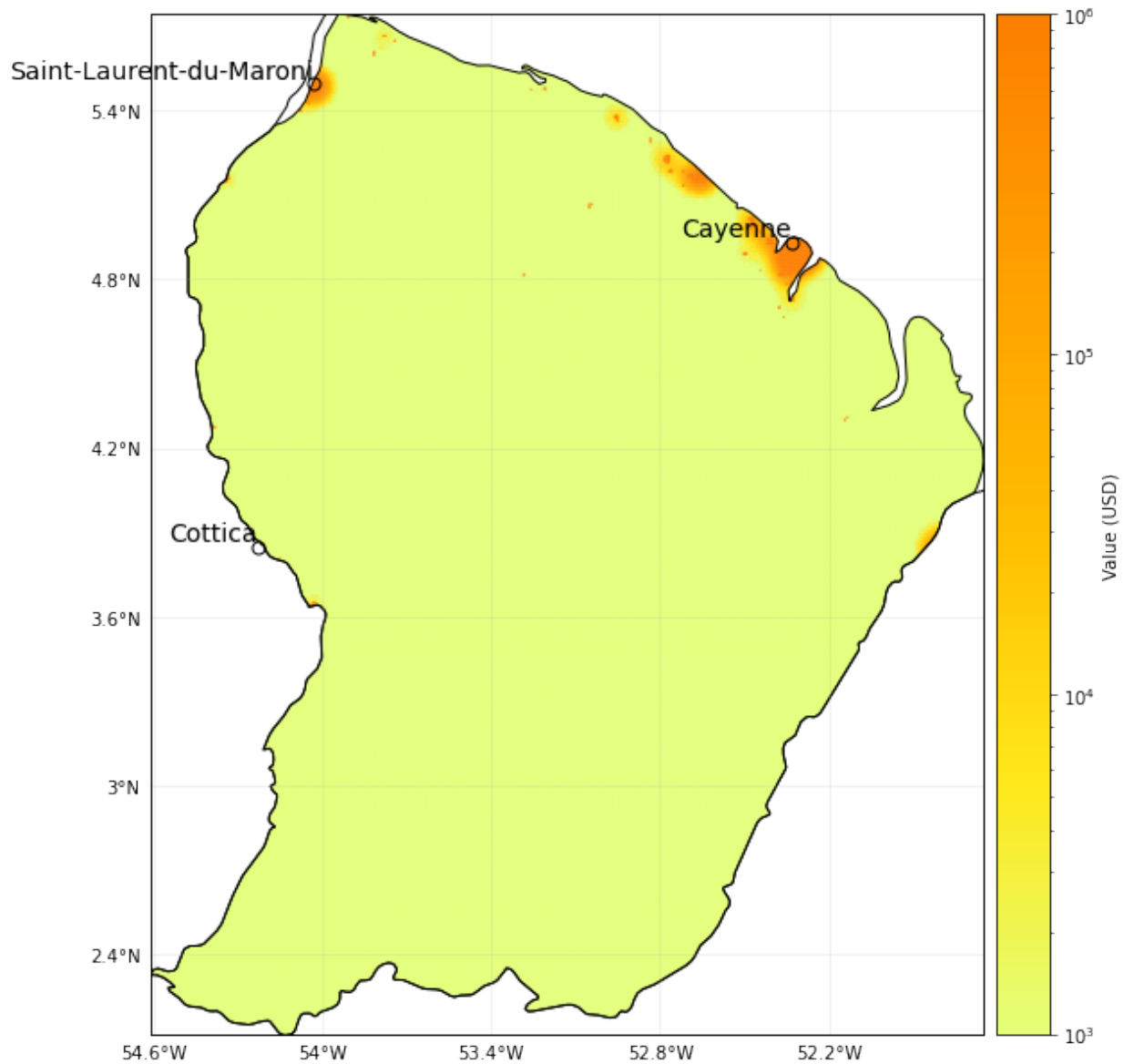
```

```

<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x21481092488>

```

French Guiana 2016 GDP: 5.000e+09 income group: 4



6.6 Change exponents of the polynomial transformation used for nightlight 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]`)

```
ch_pol_val = BlackMarble()
ch_pol_val.set_countries(['Switzerland'])
ch_pol_val.plot_hexbin(vmax=1e8)

ch_pol_val1 = BlackMarble()
ch_pol_val1.set_countries(['Switzerland'], poly_val=[0, 0, 0, 0, 1])
```

(continues on next page)

(continued from previous page)

```
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()
```

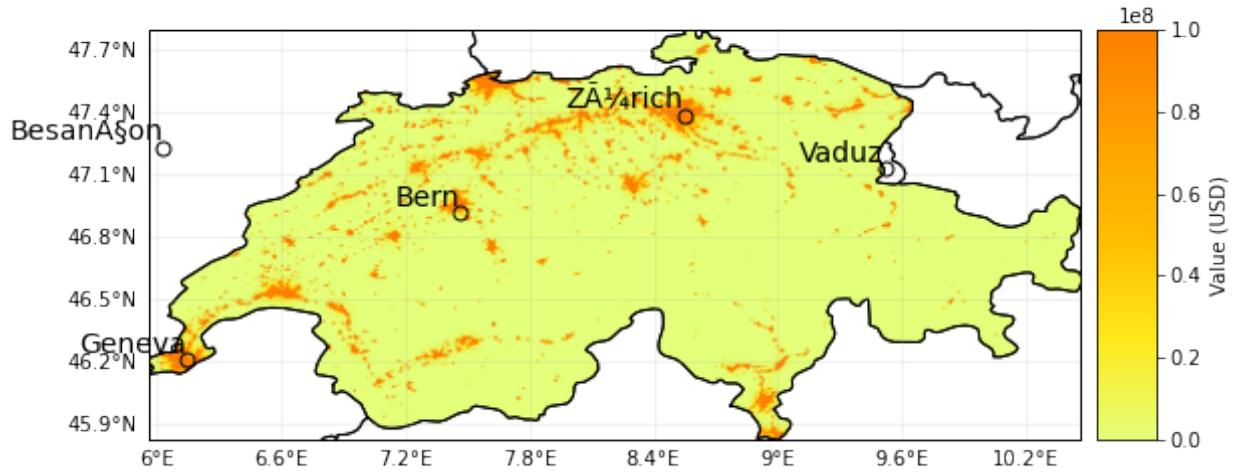
```
2021-02-15 10:47:24,209 - climada.util.finance - INFO - GDP CHE 2016: 6.713e+11.
2021-02-15 10:47:24,375 - climada.util.finance - INFO - Income group CHE 2016: 4.
2021-02-15 10:47:24,375 - climada.entity.exposures.black_marble - INFO - Nightlights_
↳from NASA's earth observatory for year 2016.
2021-02-15 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()
```

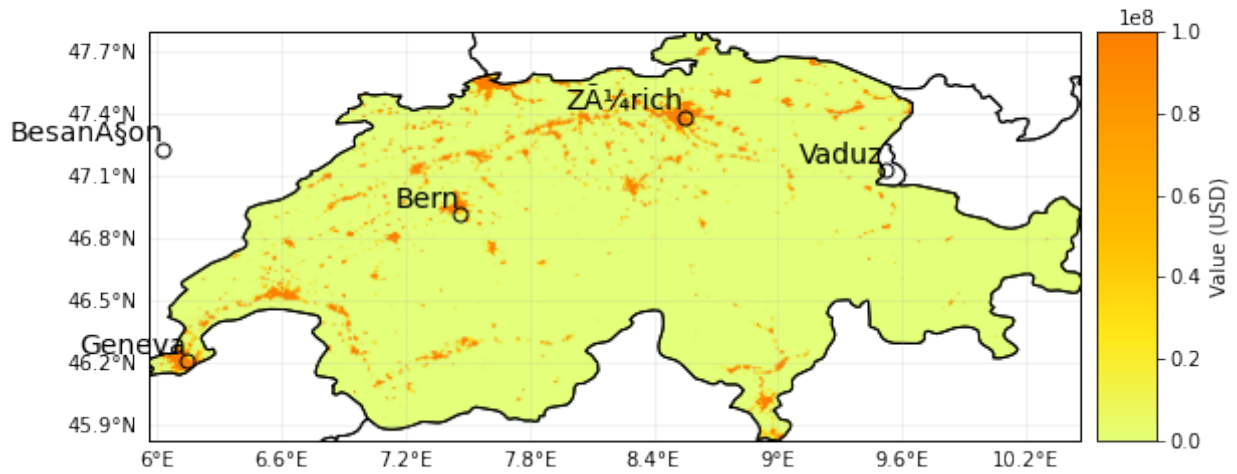
```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x2148f8f2088>
```

```
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



Switzerland 2016 GDP: 6.713e+11 income group: 4



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: OpenStreetMap for Multi-Faceted Climate Risk Assessments. Environ. Res. Commun. 6 015005 doi: 10.1088/2515-7620/ad15ab*

7.1 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 within 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 download 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_data_loader` 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.

7.2 Downloading, clipping and parsing OSM data

7.2.1 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](#).

```
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

```
# (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.

```
# check available critical infrastructure types:
osm_flex.config.DICT_CIS_OSM.keys()
```

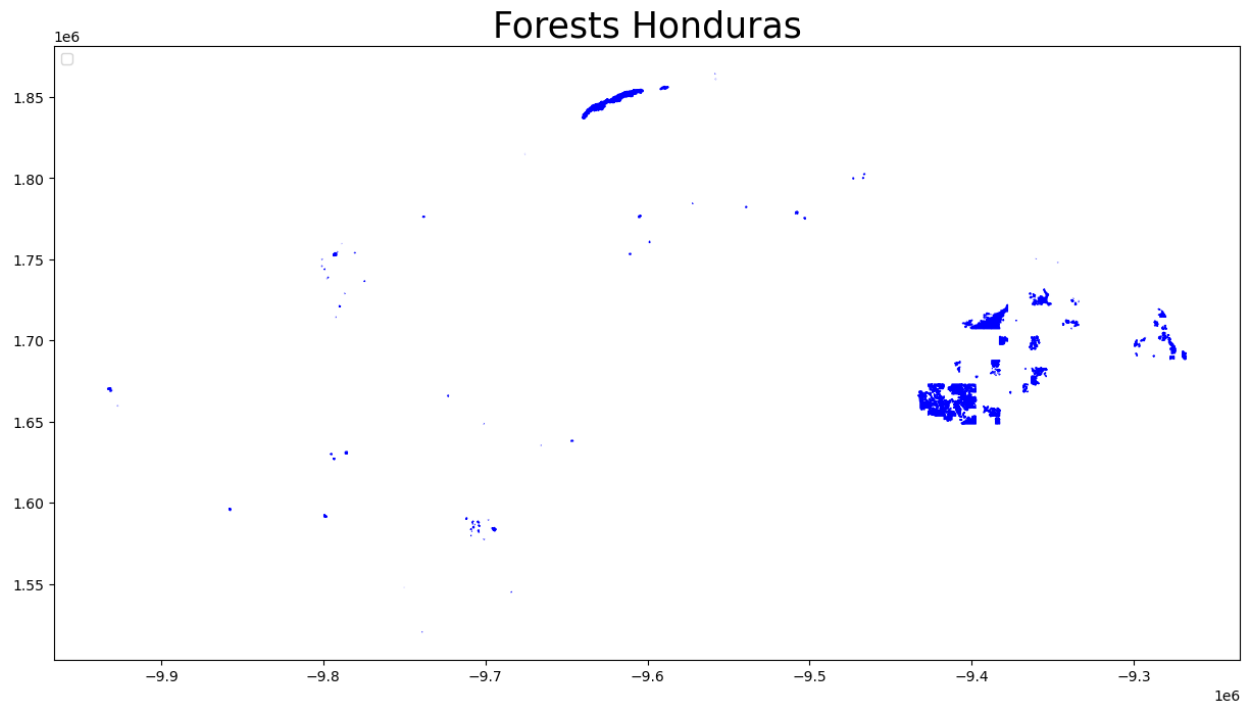
```
dict_keys(['education', 'healthcare', 'water', 'telecom', 'road', 'main_road', 'rail',
→ 'air', 'gas', 'oil', 'power', 'wastewater', 'food', 'buildings'])
```

```
# lets extract all roads from the Honduras file, via the wrapper
gdf_roads_hnd = osm_flex.extract.extract_cis(
    path_hnd_dump,
    'road')
```


(continued from previous page)

```
ax.legend(handles=handles, loc='upper left')
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 the 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

```
# Define a bbox around roughly Honduras, San Salvador & Nicaragua
bbox_customreg = [-90.043806, 10.939216, -83.116254, 15.956278]

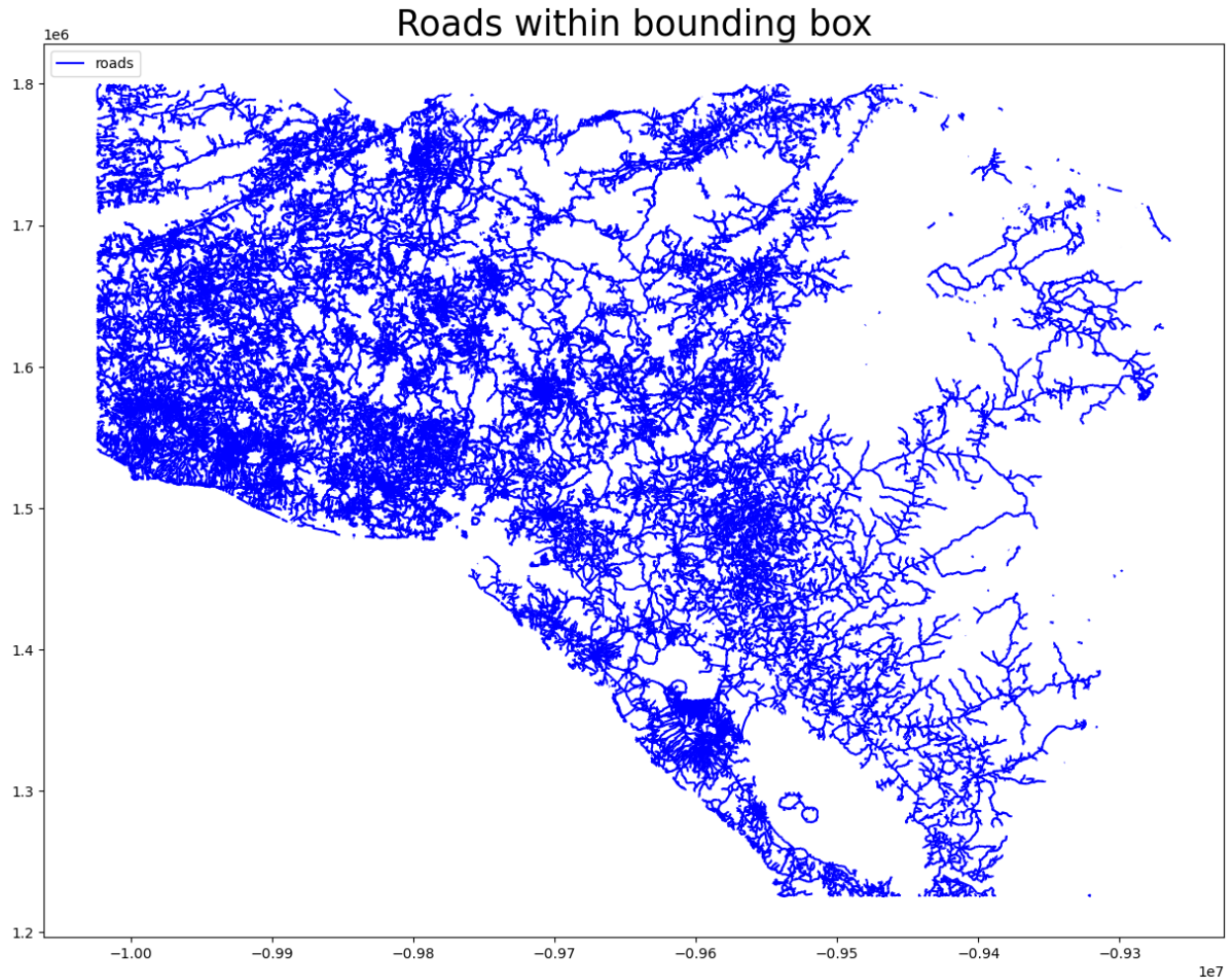
# load the central-america parent file from which we want to clip
path_ca_dump = osm_flex.download.get_region_geofabrik('central-america')
# new save path
path_custom_dump = os.path.join(osm_flex.config.OSM_DATA_DIR, 'custom_region.osm.pbf')

# 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.

```
# lets extract all roads from the Honduras file, via the wrapper
gdf_roads_customreg = osm_flex.extract.extract_cis(path_custom_dump, 'road')
```

Clip data within a custom (multi-)polygon See the tutorial provided by `osm-flex`.

7.2.2 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.

Example: Buildings & Churches in the city of Zurich

```
# reload required modules
import shapely
from climada_petals.entity.exposures.osm_data_loader import OSMApiQuery
from climada import CONFIG

DATA_DIR = CONFIG.exposures.openstreetmap.local_data.dir()
```

```
# 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)
```

(continues on next page)

(continued from previous page)

```

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:

```
gdf_zrh_churches = zrh_churchquery_bbox.get_data_overpass()
```

```
gdf_zrh_buildings = zrh_buildingquery_poly.get_data_overpass()
```

```

2023-12-07 13:31:50,483 - climada_petals.entity.exposures.osm_data_loader - 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.

```
gdf_zrh_churches.head()
```

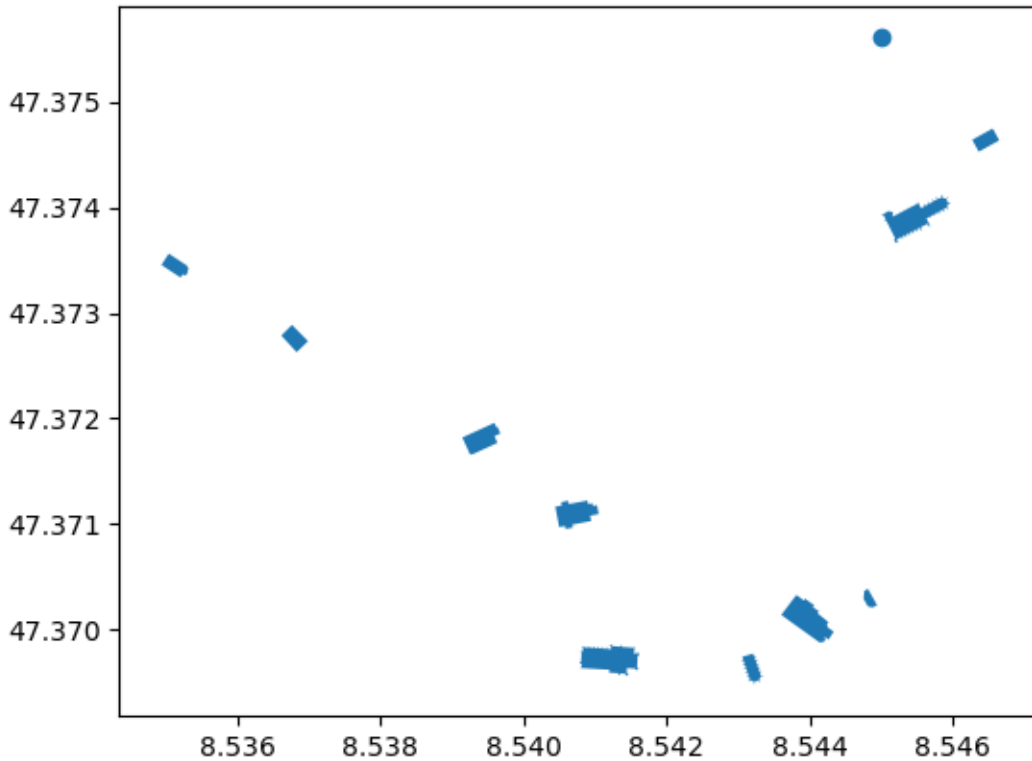
```

      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', ...

```

```
gdf_zrh_churches.plot();
```



```
gdf_zrh_buildings.head()
```

```

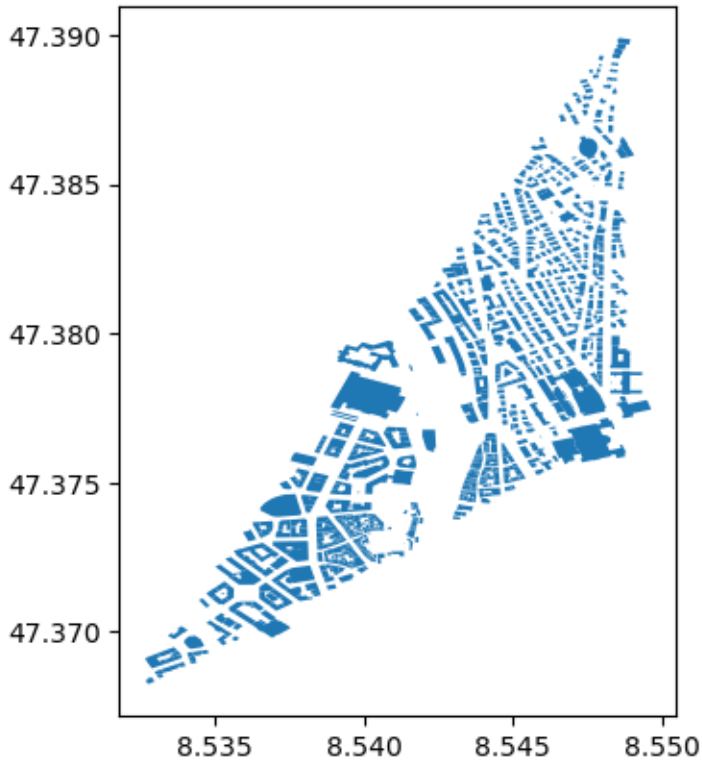
  osm_id      geometry \
0    4633  POLYGON ((8.54832 47.37961, 8.54831 47.37936, ...
1    7565      GEOMETRYCOLLECTION EMPTY
2   405794  POLYGON ((8.54124 47.37587, 8.54119 47.37583, ...
3  4235320  POLYGON ((8.54372 47.37844, 8.54377 47.37804, ...
4  4281153  POLYGON ((8.53952 47.37959, 8.53952 47.37960, ...

      tags
0  {'addr:city': 'Zürich', 'addr:house...
1  {'building': 'public', 'building:le...
2  {'addr:city': 'Zürich', 'addr:coun...
3  {'building': 'office', 'type': 'm...
4  {'addr:house...': '2', 'addr:stree...

```

```
gdf_zrh_buildings.plot()
```

```
<Axes: >
```



7.3 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.

7.3.1 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.

```

from climada.entity.exposures import Exposures
import climada.util.lines_polys_handler as u_lp

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_
↳choosing 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.

```

7.3.2 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 valueate 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 obtained via `openstreetmap`:

```

from climada.entity.exposures.litpop import LitPop
import geopandas as gpd
import numpy as np

```

(continues on next page)

(continued from previous page)

```

# 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
    """
    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

```

```

# 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)...

```

(continues on next page)

(continued from previous page)

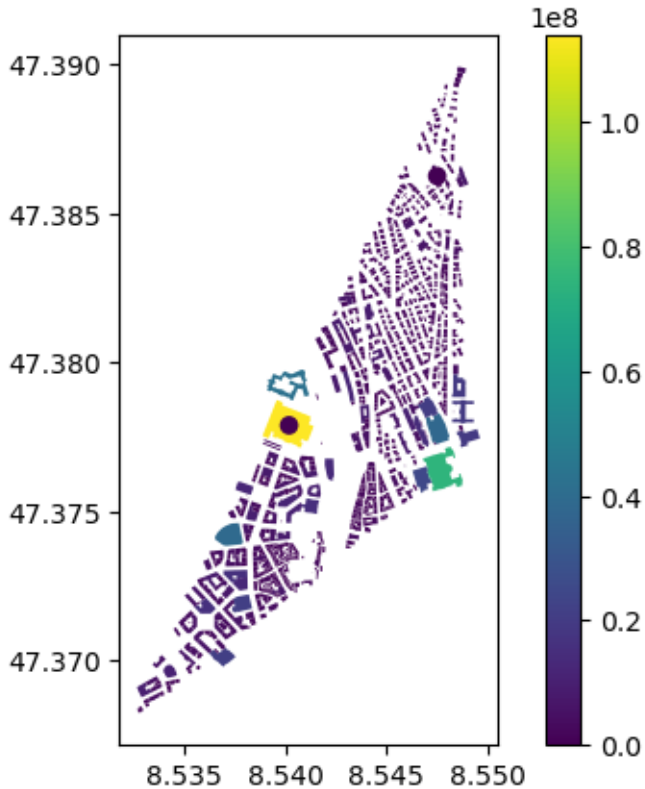
```
2023-11-27 10:40:59,072 - climada.util.finance - INFO - GDP CHE 2014: 7.265e+11.
2023-11-27 10:40:59,733 - climada.util.finance - INFO - GDP CHE 2018: 7.256e+11.
2023-11-27 10:40:59,776 - climada.entity.exposures.base - INFO - Hazard type not set.
↳in impf_
2023-11-27 10:40:59,777 - climada.entity.exposures.base - 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
```

```
# Distribute building values based on building footprint area and total LitPop value.
↳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()
```

```
<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’

```
# Load required packages:
```

```
# One command does it all (getting LitPop Exp, re-assigning values, converting back_
→ into exposure format)
```

7.3.3 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(...)
```

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* intensity is equal to the change in crop yield in a given year from the long-term average. It 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 = $0.05 * 250,000ha * 6t/(ha * y) = 75,000t/y$.

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: $75,000t/y * (-0.20) * 1y = -15,000t$.

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 $0.5^\circ \times 0.5^\circ$ 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-analysis 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)
```

(continues on next page)

(continued from previous page)

```
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)

```
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'

yerrange_haz = (2001, 2005) # yerrange for hazard (demo data only available from_
↳2001 to 2005)
yerrange_hist_mean = (2001, 2005) # yerrange for reference historical mean (demo_
↳data only available from 2001 to 2005)
haz = RelativeCropyield()
haz.set_from_isimip_netcdf(input_dir=INPUT_DIR, yerrange=yerrange_haz, ag_model=
↳'lpjml',
                           cl_model='ips1-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(yerrange_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"""

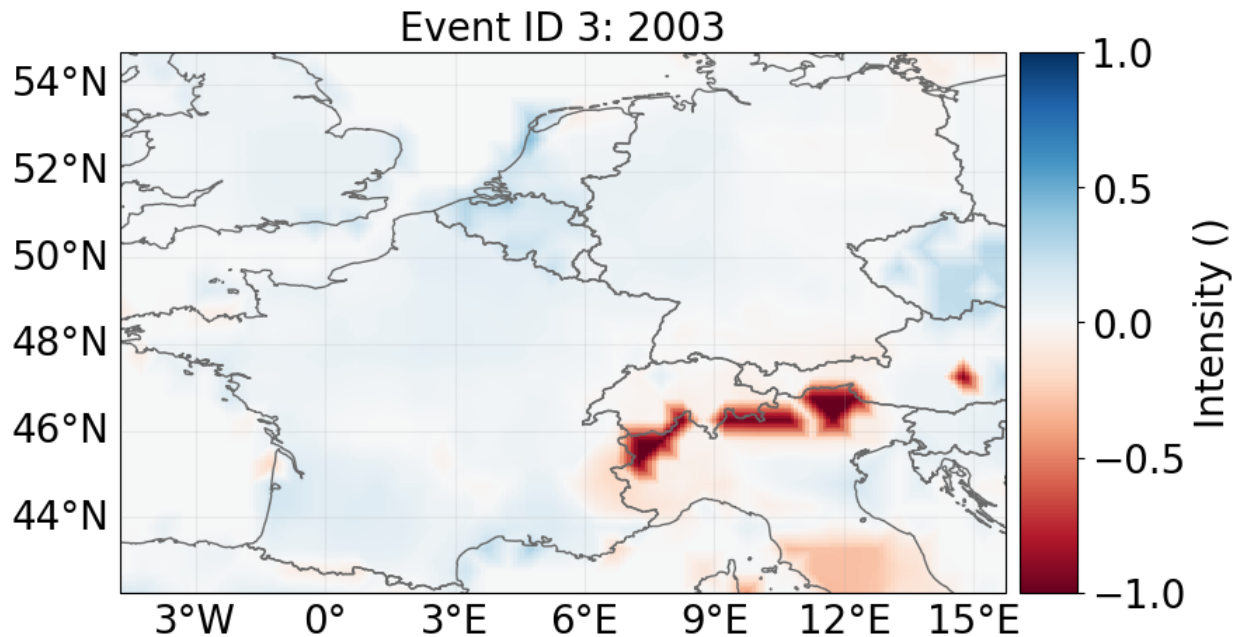
print("After calling set_rel_yield_to_int(), intensity is '%s' with unit '%s'.\n"
↳%(haz.intensity_def, haz.units))

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 ''.
```

'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.

```
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
```

(continues on next page)

(continued from previous page)

```

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(filenamees,
     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']

File c:\users\me\git\climada_petals\climada_petals\hazard\relative_cropyield.py:791,

```

(continues on next page)

(continued from previous page)

```

->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*: tonnes, $1t = 1000kg$

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"
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 simulation 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):

```

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)

```

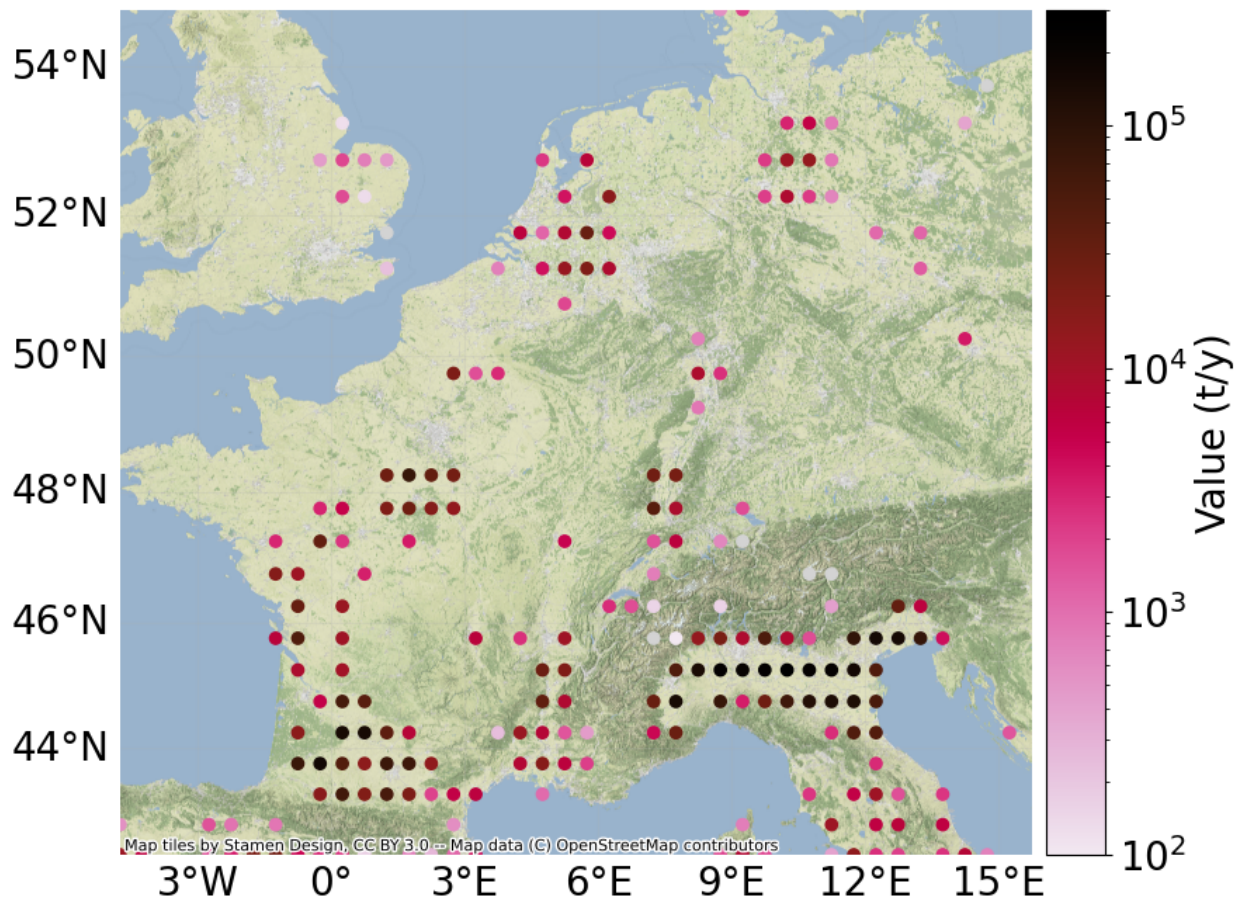
(continues on next page)

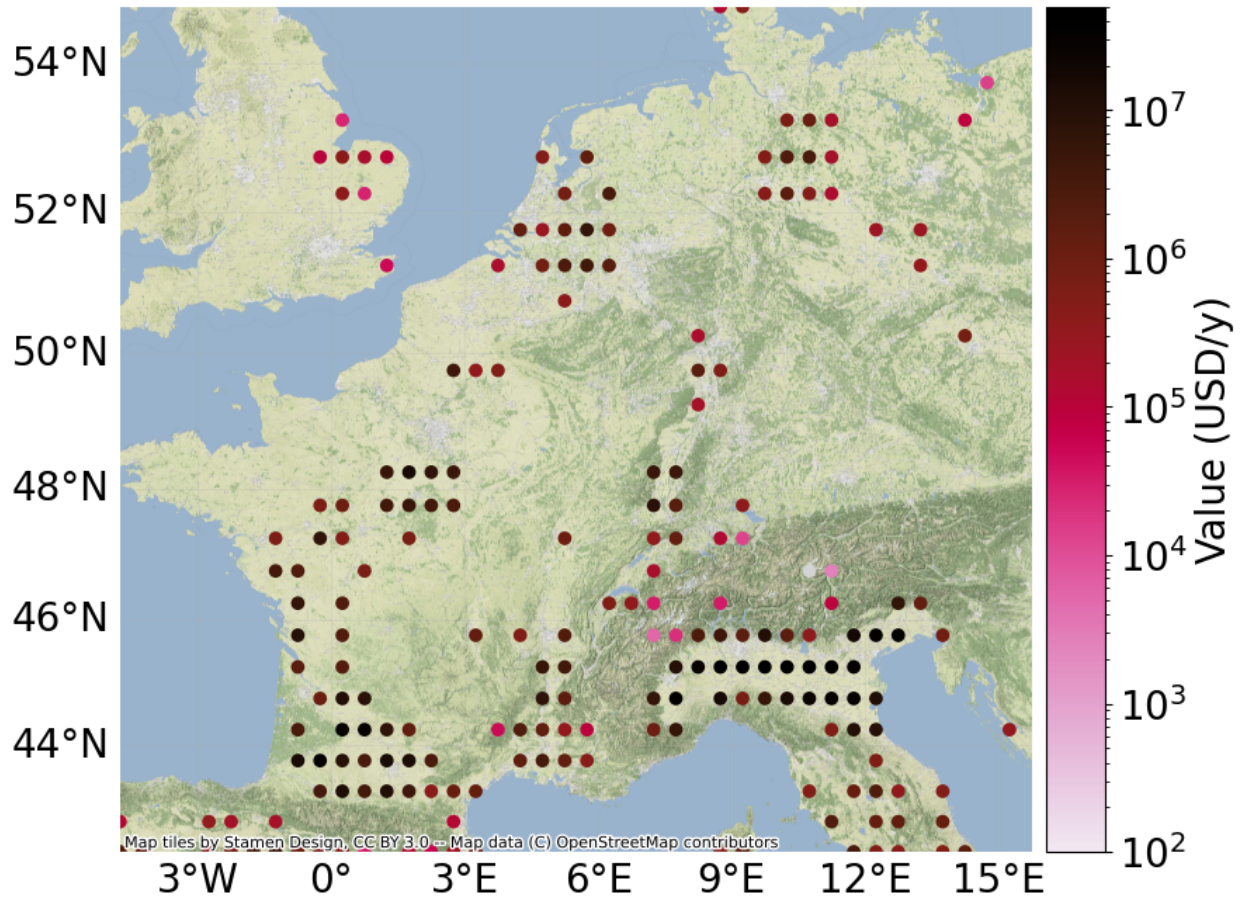
(continued from previous page)

```
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
→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(
```

```
<GeoAxes: >
```





8.4.2 Initiating an exposure set from several model runs:

Requires data download from <https://esg.pik-potsdam.de/search/isimip/>:

- `gpic_gfdl-esm2m_ewembi_historical_2005soc_co2_yield-ric-firr_global_annual_1861_2005.nc`
- `gpic_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 combinations 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).

- 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 average 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()]

```

(continues on next page)

(continued from previous page)

```

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\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=
```

(continues on next page)

(continued from previous page)

```

↪ 'lpjml',
                                cl_model='ipsl-cm5a-1r', 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
↪ 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))

```


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), TCS, `Impacts`, 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](https://doi.org/10.3929/ethz-b-000548850)

```
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'
```


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.

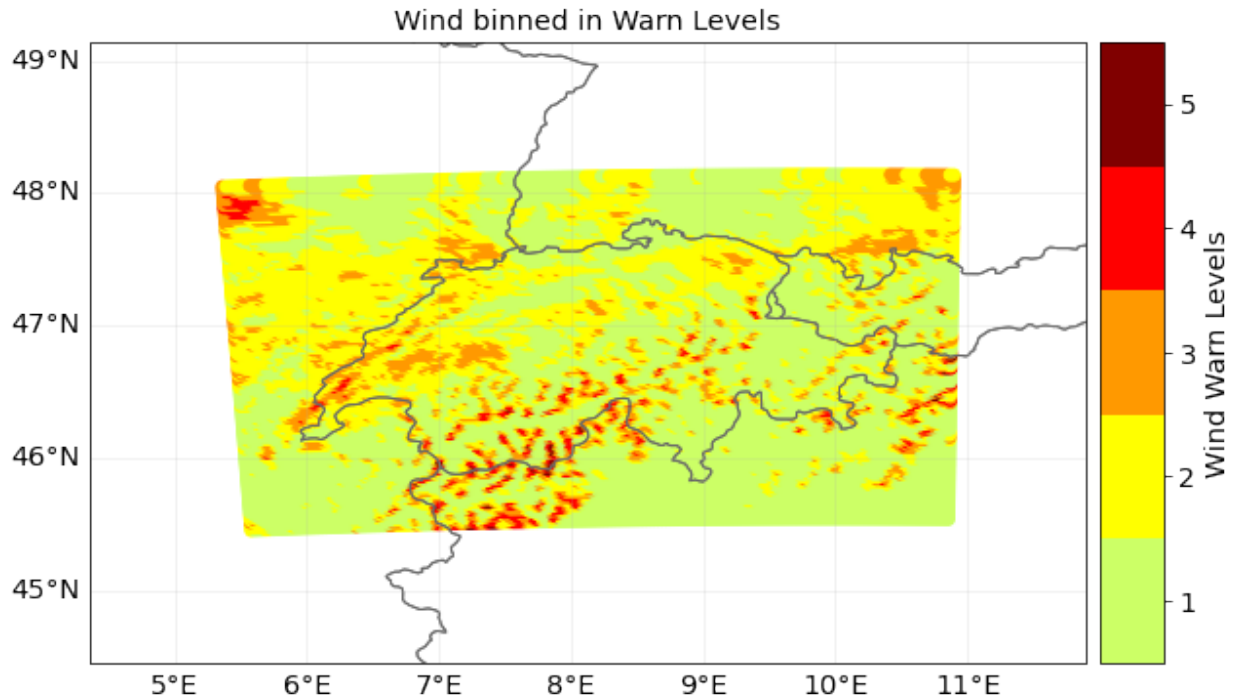
```
# Load MeteoSwiss storm example via client (COSMO2-E model)
client = Client()
dataset = client.get_dataset_info(name="cosmo2_2019121400")
path_files = client.download_dataset(dataset)[1][0]
ncdf = xr.open_dataset(path_files)
wind_matrix = ncdf.VMAX_10M.values[0, :, :] # take one ensemble member of 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.

```
# Define warn levels for this example
warn_levels = np.array([0.0, 19.44, 25.0, 30.55, 38.88, 300.0])

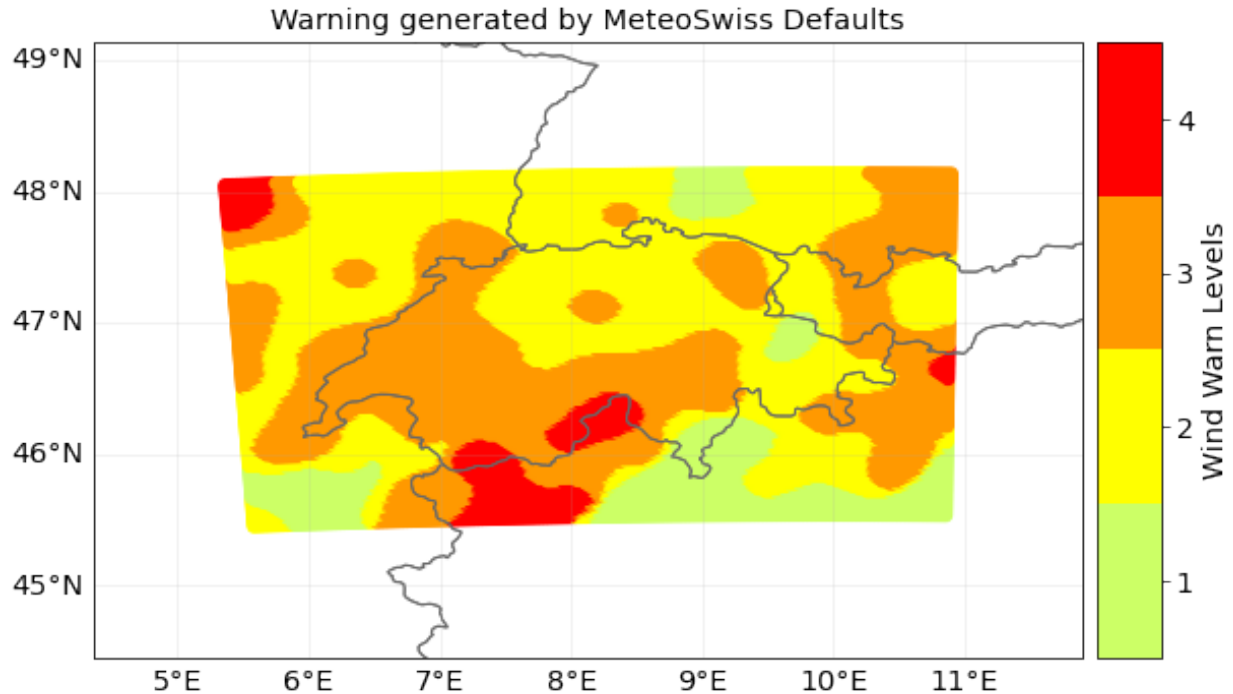
# Define warning parameters, such that the data is only binned in warn levels
↳(without operations)
warn_params_only_binning = Warn.WarnParameters(warn_levels, operations=[])
binned_map = Warn.from_map(wind_matrix, coord, warn_params_only_binning)
binned_map.plot_warning_meteoswiss_style(var_name='Wind 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.

```
# 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.

```
# 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');

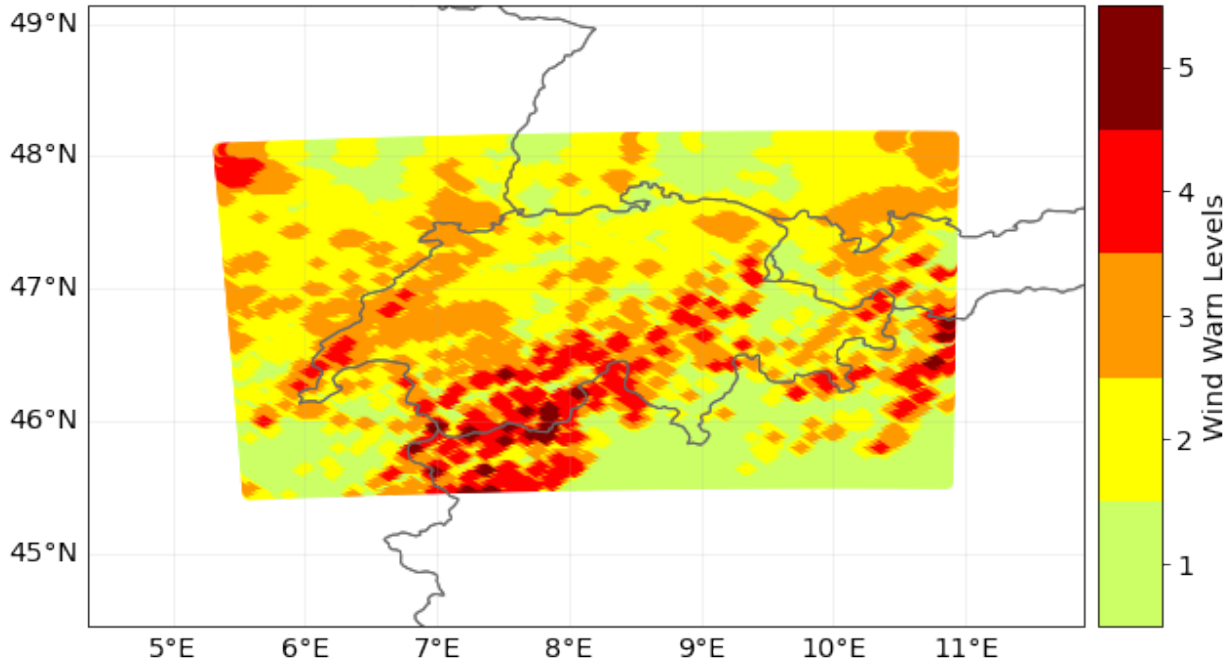
# first erosion, then dilation, then median filtering
warn_params = Warn.WarnParameters(warn_levels, operations=[(Operation.erosion, 1),
                                                           (Operation.dilation, 3),
                                                           (Operation.median_
↳ 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 '
                                                    'Erosion, Dilation, and Median Filtering');
```

(continues on next page)

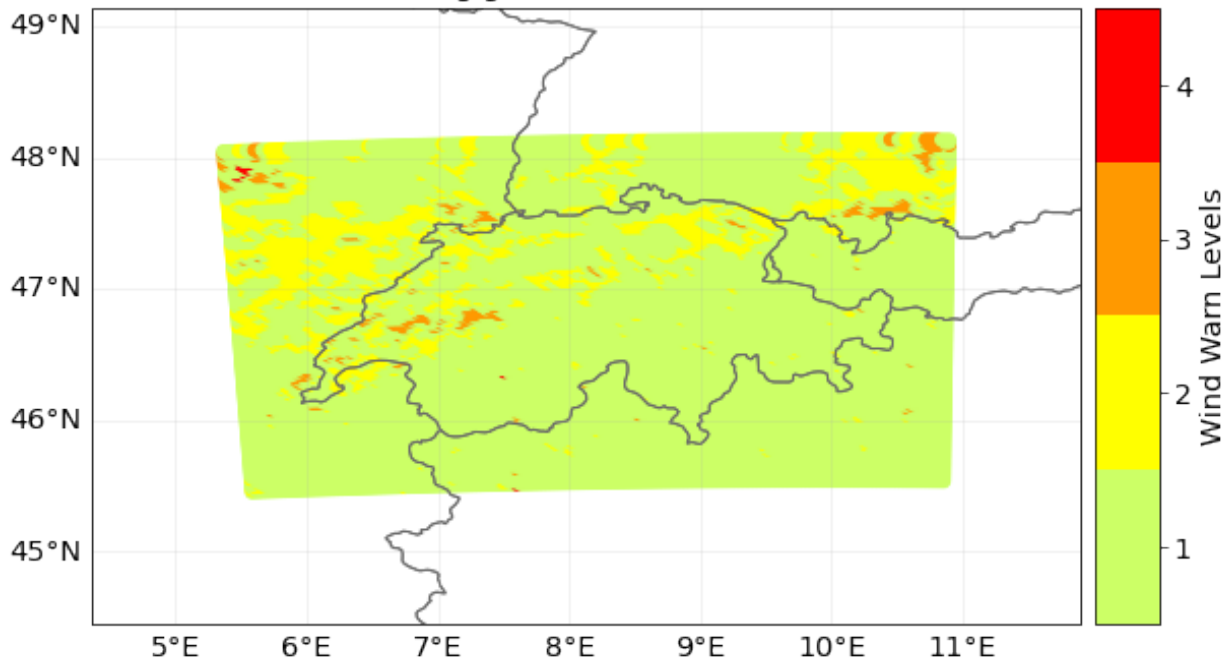
(continued from previous page)

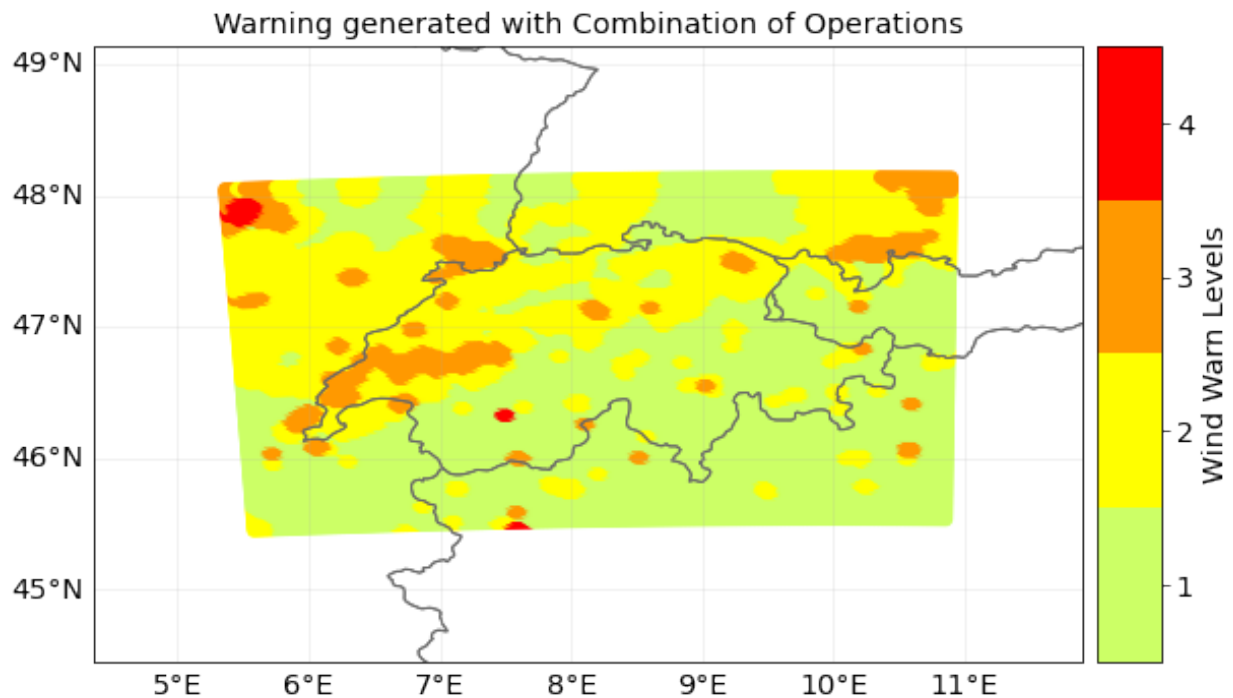
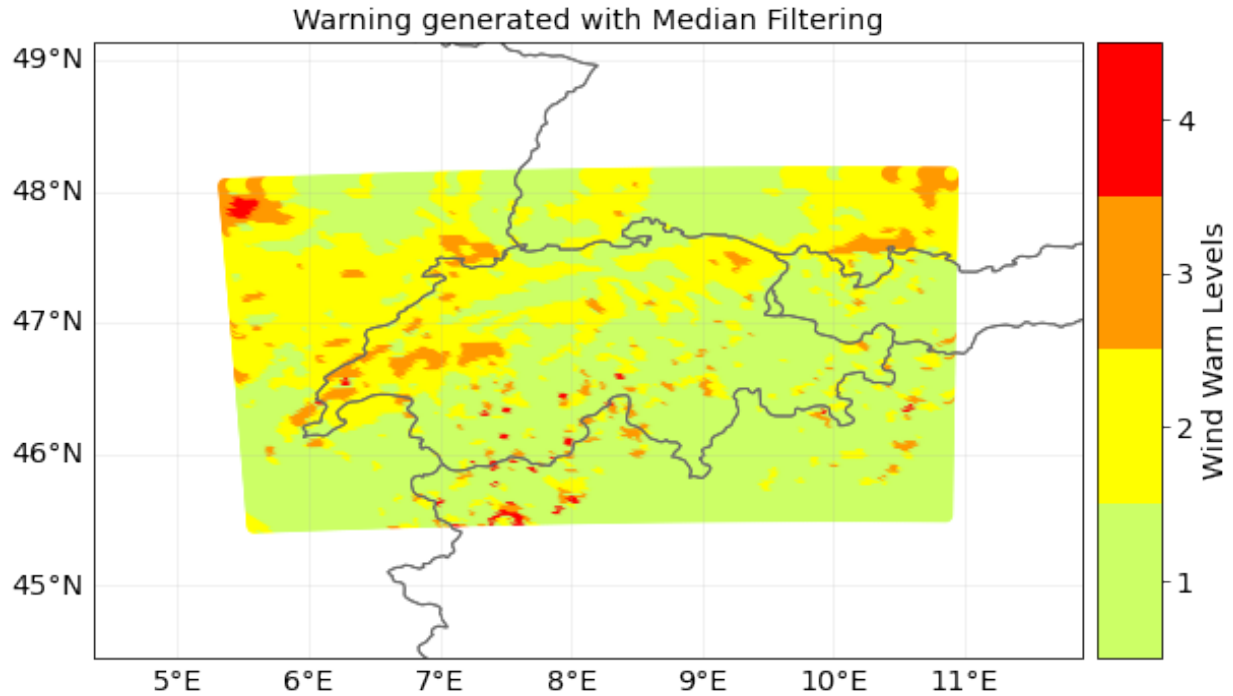
```
↳Operations');  
'Combination of
```

Warning generated with Dilation



Warning generated with Erosion





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.

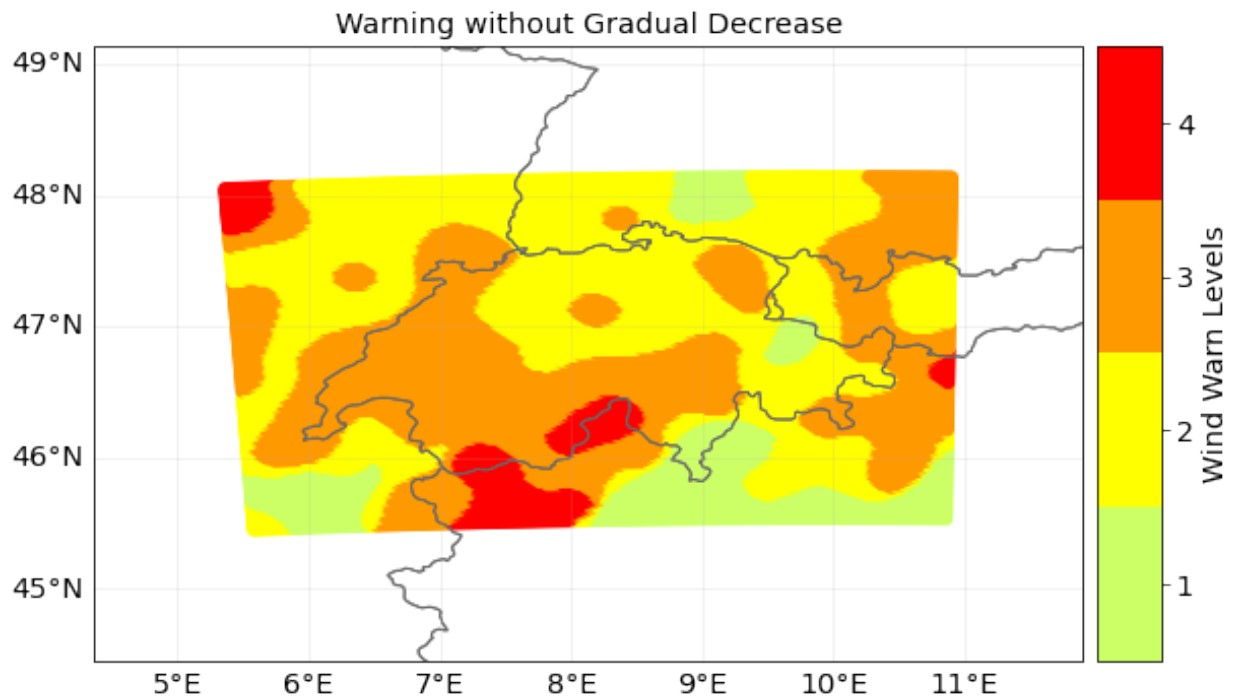
```
# Warning without and with gradual decrease of regions of higher level to lower_
↳ levels to see difference to default parameters plot
warn_params = Warn.WarnParameters(warn_levels)
```

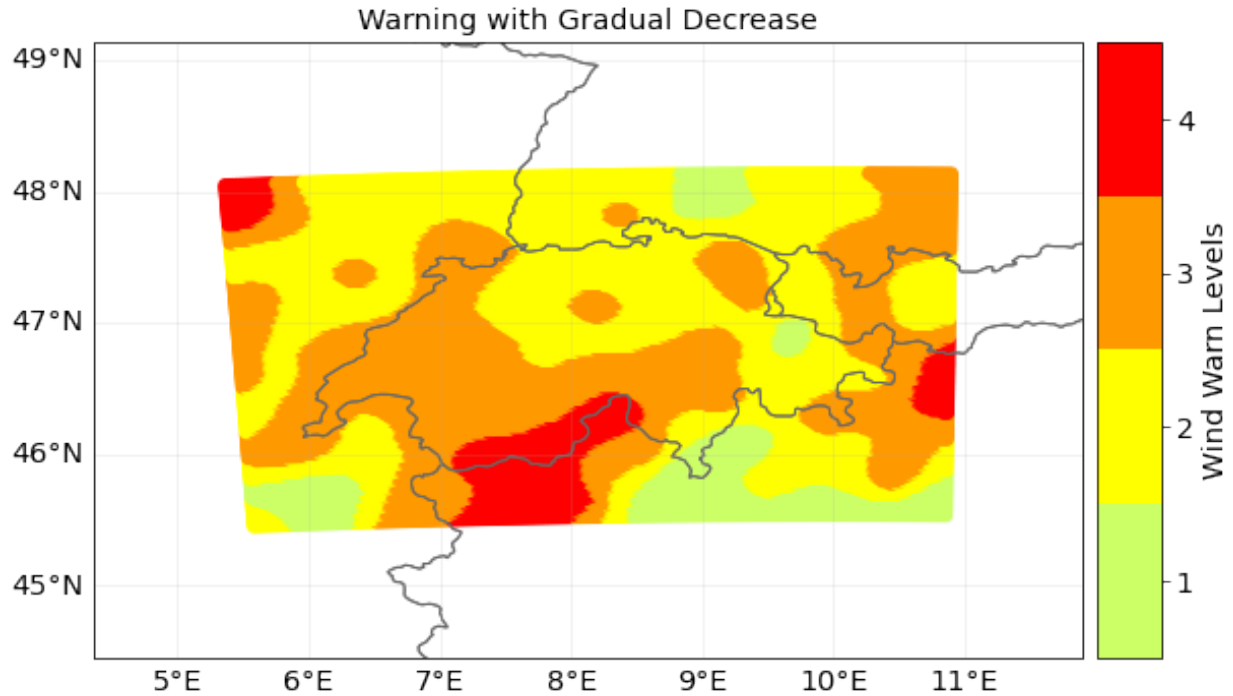
(continues on next page)

(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');
```

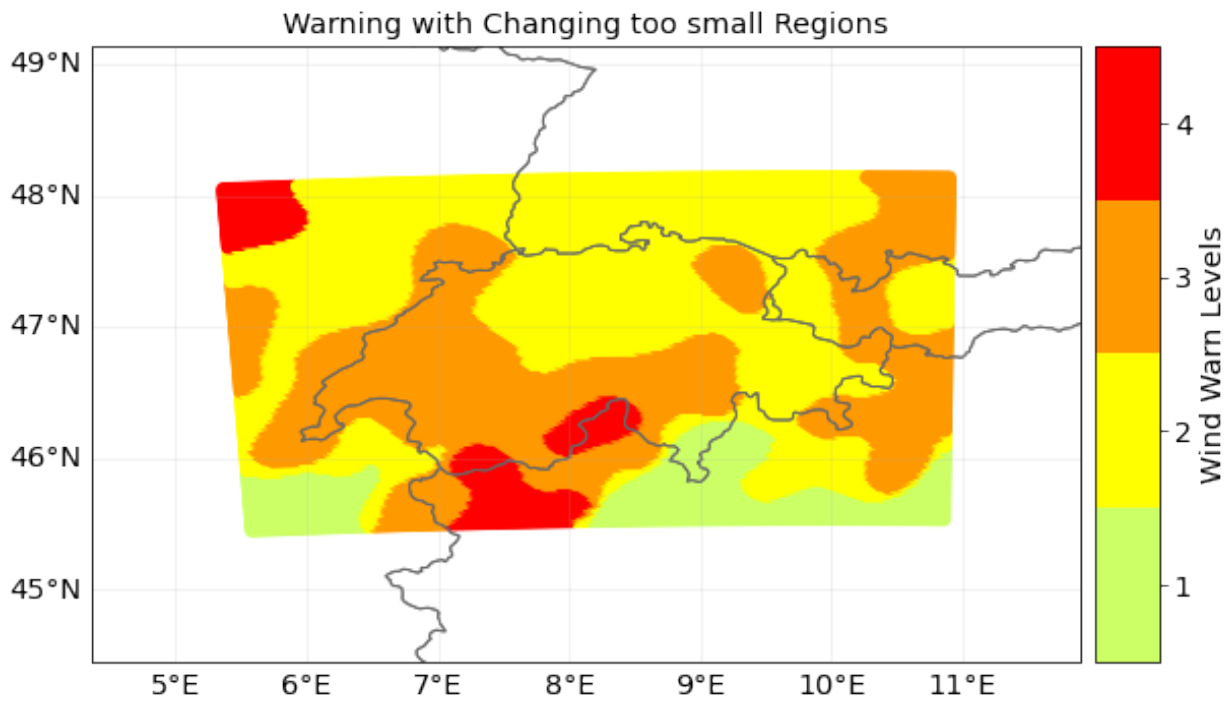
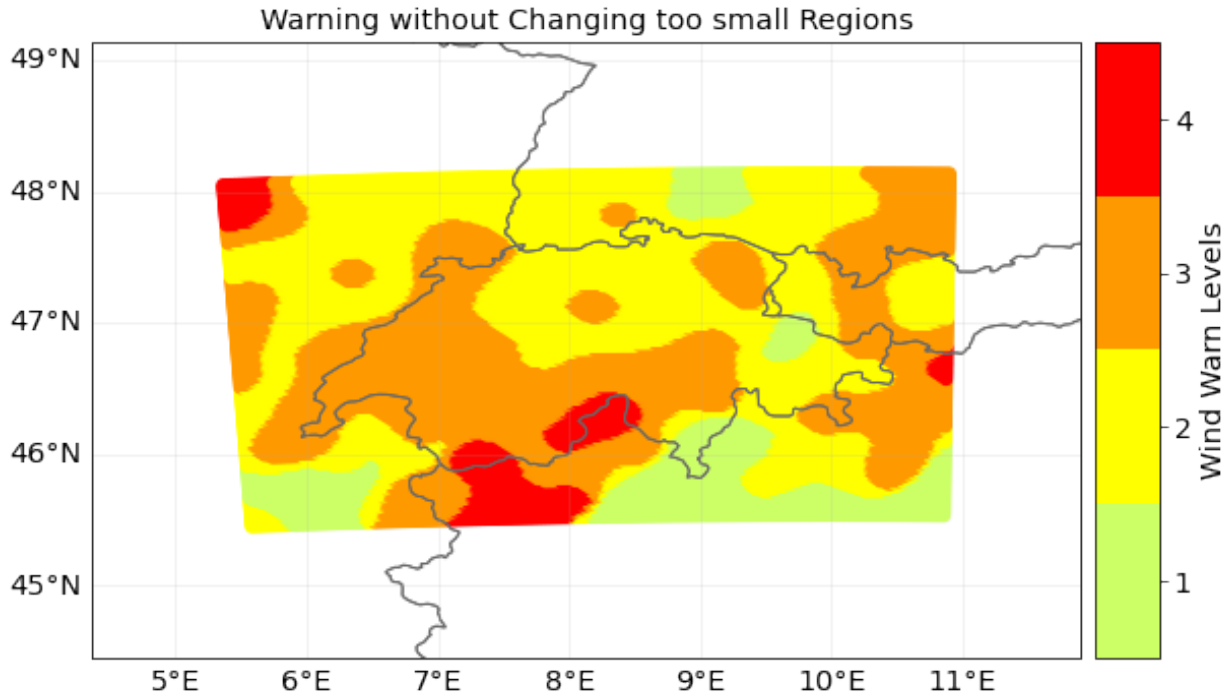




Further, small regions of a warn level formed by the algorithm can be changed to the surrounding warn level. Therefore, the parameter `change_sm` needs to be set to the number of coordinate below which a region is regarded as too small.

```
# 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');
```



HAZARD EXAMPLE - TC HAITI

This example shows how the Warn module can be used to generate warnings of natural catastrophes, e.g., tropical cyclons.

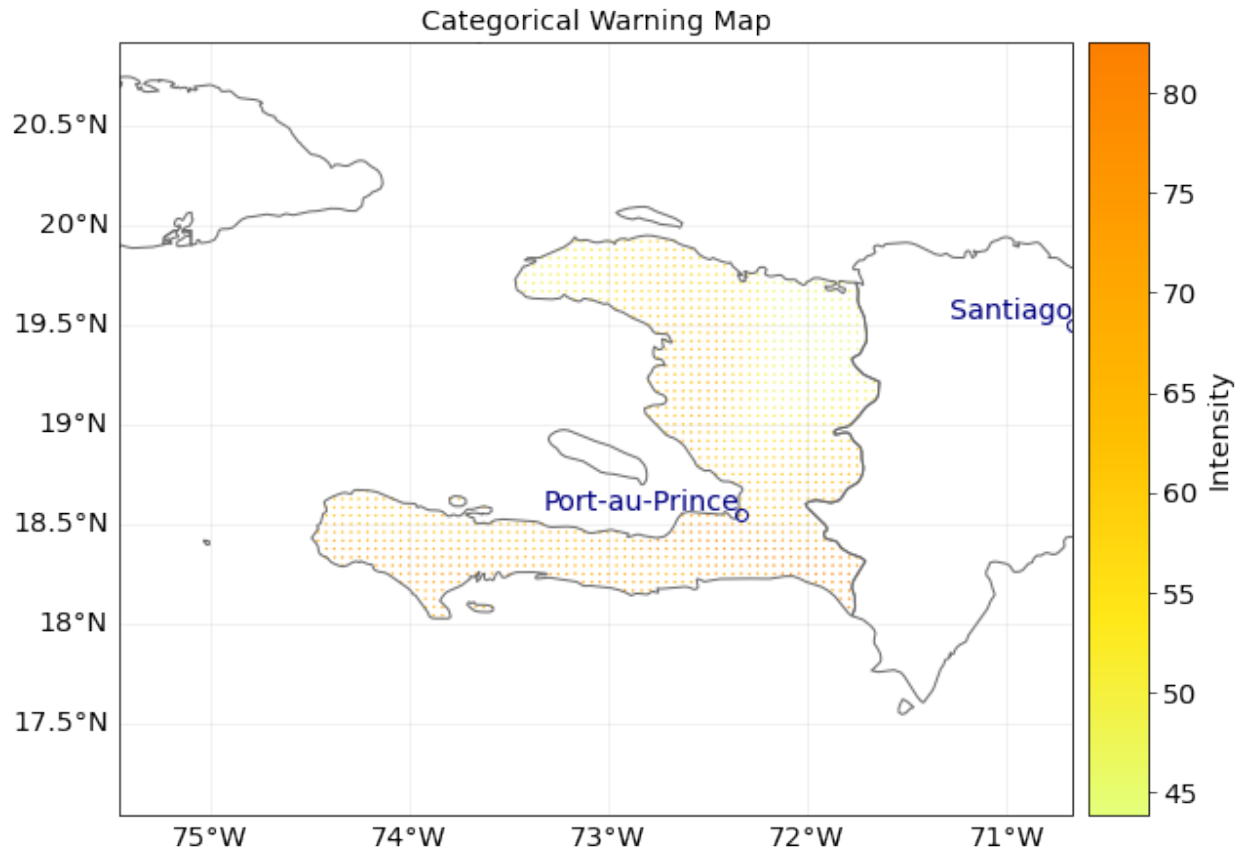
```
tc_dataset_infos = client.list_dataset_infos(data_type='tropical_cyclone')
client.get_property_values(tc_dataset_infos, known_property_values = {'country_name':
↪ 'Haiti'});
```

```
# 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()

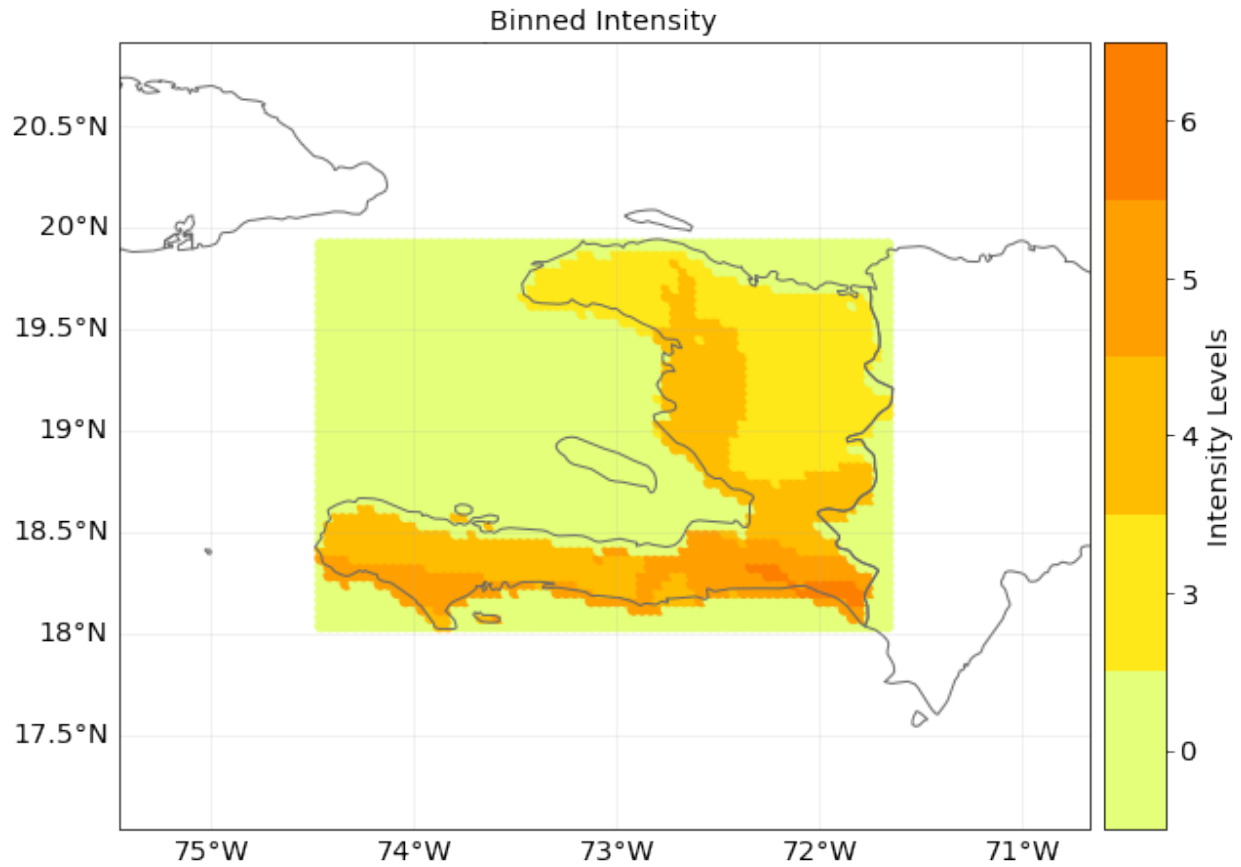
geo_bin_from_array(tc_haiti.intensity.todense().transpose(), coord_haiti, 'Intensity',
↪ 'Categorical Warning Map', **plotting_parameters);
```

```
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
```

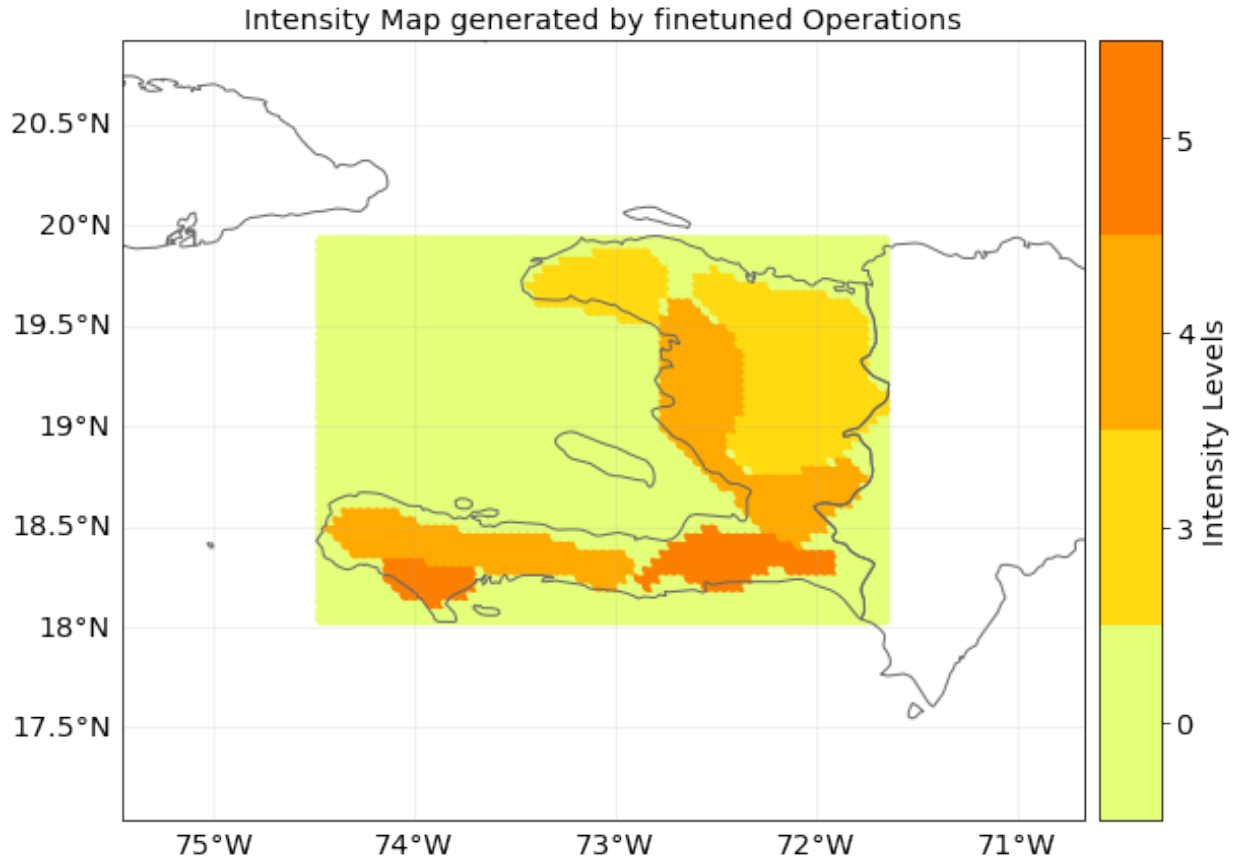


```
# Plot binned intensities
# warn Levels
warn_levels = [0, 20, 30, 40, 60, 70, 80, 1000]
grid, coord_haiti = Warn.zeropadding(tc_haiti.centroids.lat, tc_haiti.centroids.lon, ↵
↵tc_haiti
                                .intensity.todense())

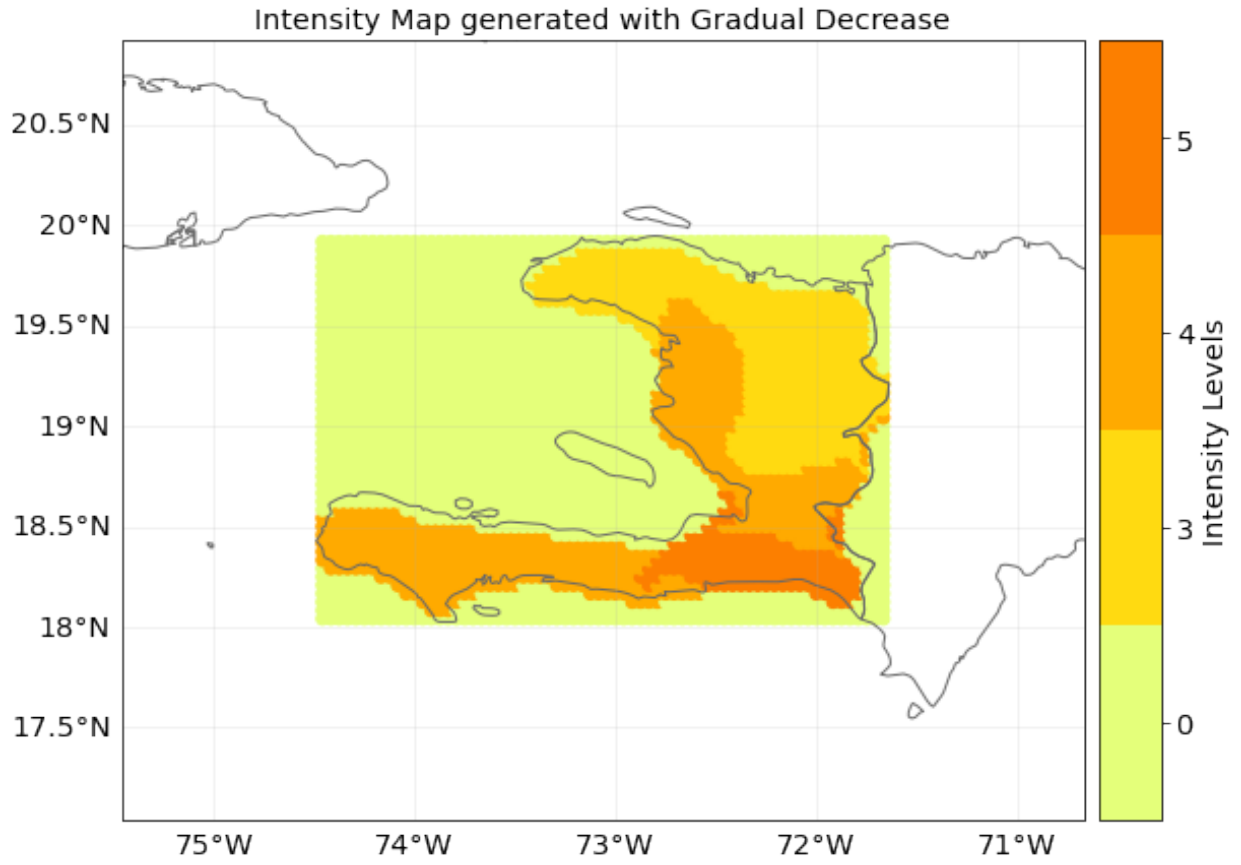
# no filtering operations applied - only binning as reference
warn_params_only_binning = Warn.WarnParameters(warn_levels, operations=[])
binned_only = Warn.from_map(grid, coord_haiti, warn_params_only_binning)
binned_only.plot_warning(var_name='Intensity Levels', title='Binned Intensity', ↵
↵**plotting_parameters);
```



```
# 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);
```



```
# 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);
```



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.

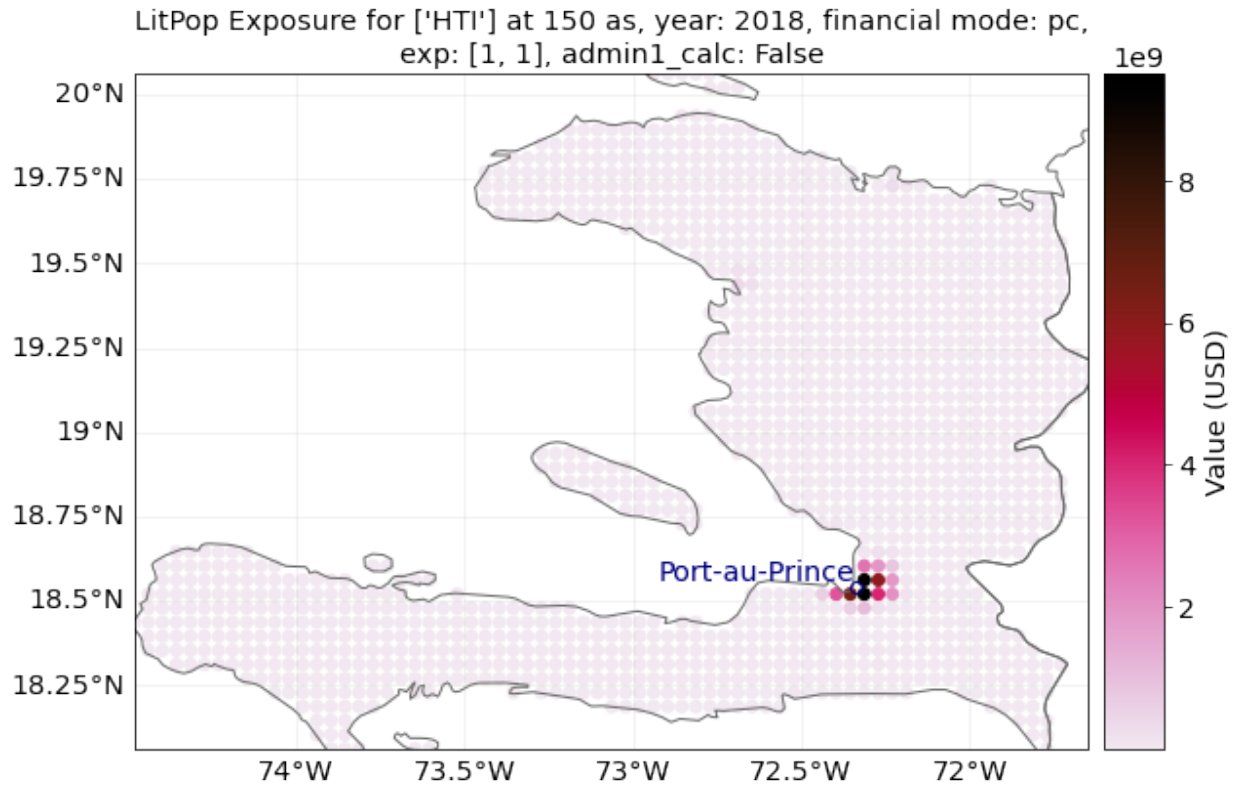
```
# Get data
exp_haiti = client.get_litpop(country="Haiti");
```

```
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
```

```
impf = ImpfTropCyclone().from_emanuel_usa()
impf_set = ImpactFuncSet()
impf_set.append(impf)
```

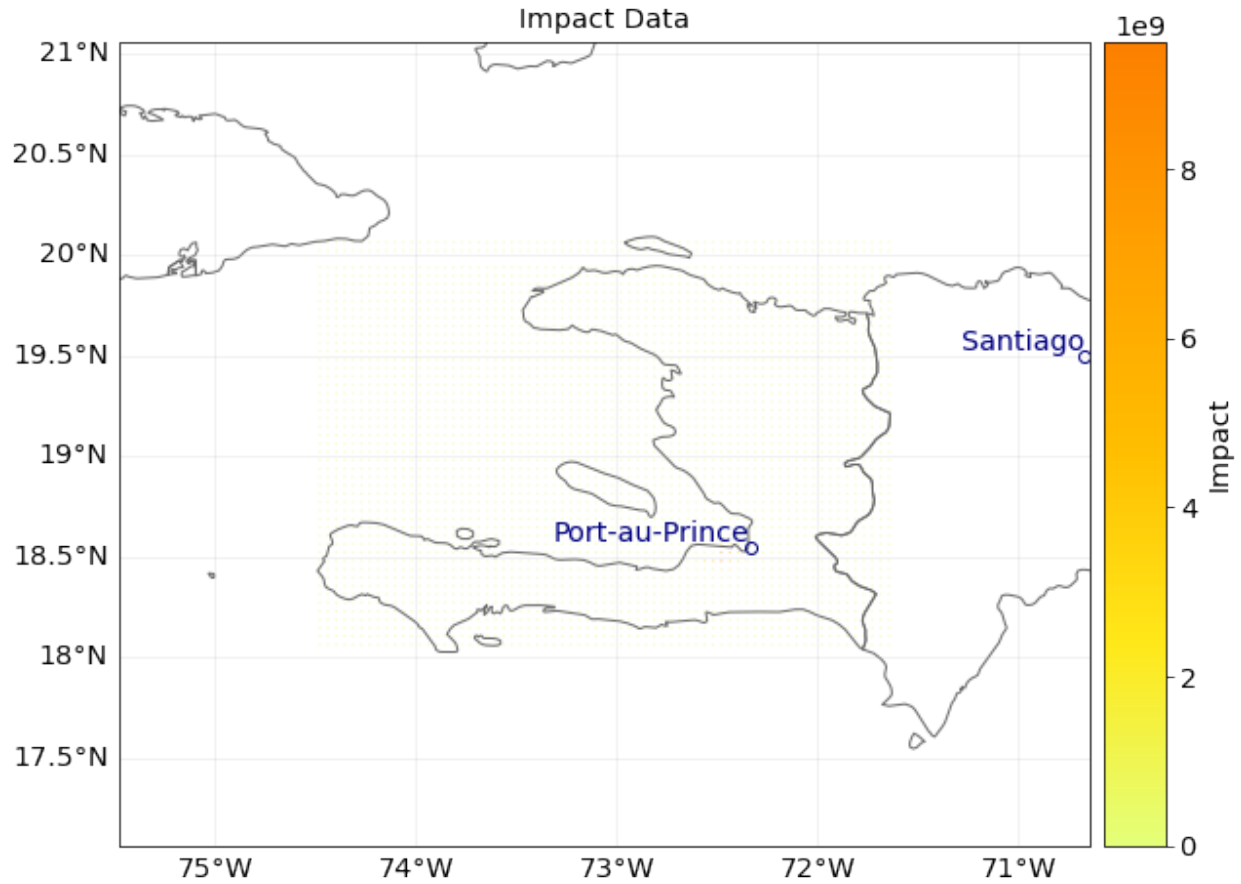
```
# Plot exposures
exp_haiti.plot_scatter();
```

```
/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)
```

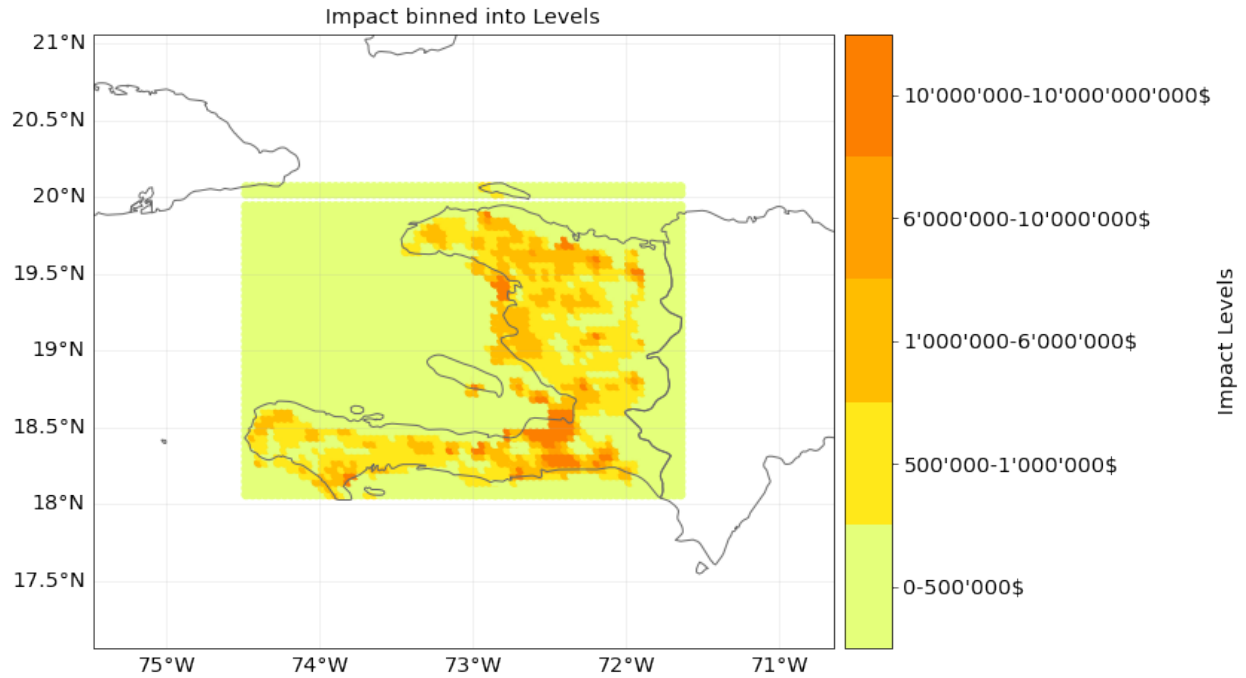


```
# Project values to rectangle
lat, lon, values = exp_haiti.gdf.latitude.to_numpy(), exp_haiti.gdf.longitude.to_
↳numpy(), exp_haiti.gdf.value.to_numpy()
grid, coord_impf = Warn.zeropadding(lat, lon, values)
```

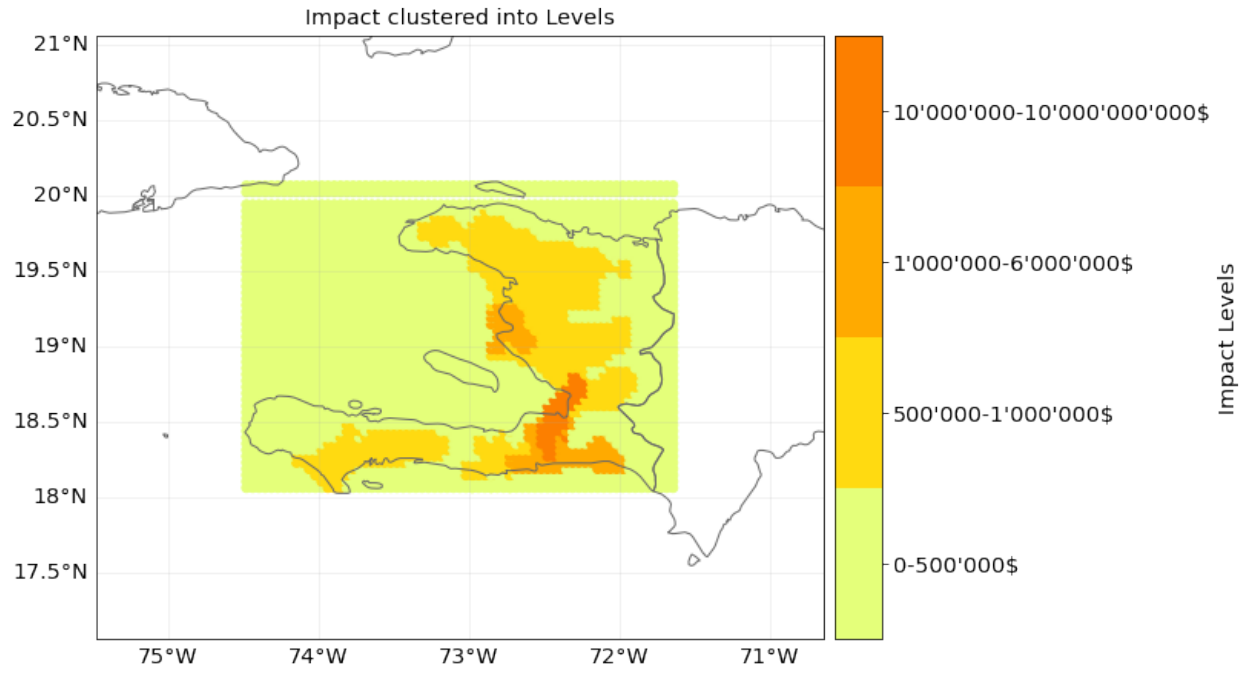
```
# Plot raw data
geo_bin_from_array(grid.flatten(), coord_impf, 'Impact', 'Impact Data', **plotting_
↳parameters);
```



```
# 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);
```



```
# 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);
```



CLIMADA

CLIMADA (CLIMate ADaptation) is a free and open-source software framework for climate risk assessment and adaptation option appraisal. Designed by a large scientific community, it helps researchers, policymakers, and businesses analyse the impacts of natural hazards and explore adaptation strategies.

As of today, CLIMADA provides global coverage of major climate-related extreme-weather hazards at high resolution (4x4km) via a [data API](#). For select hazards, historic and probabilistic events sets, for past, present and future climate exist at distinct time horizons. You will find a repository containing scientific peer-reviewed articles that explain software components implemented in CLIMADA [here](#).

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`, `OpenStreetMap`, ...). Most development is done here. The petals builds upon the core and does not work as a stand-alone.

It is recommended for new users to begin with the core (1) and the [tutorials](#) therein.

This is the Python version of CLIMADA - please see [here](#) for backward compatibility with the MATLAB version.

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
```

It is **highly recommended** to install CLIMADA into a **separate** Conda environment. See the [installation guide](#) for further information.

Follow the [tutorials](#) in a Jupyter Notebook to see what can be done with CLIMADA and how.

13.2 Documentation

The online documentation is available on [Read the Docs](#). The documentation of each release version of CLIMADA can be accessed separately through the drop-down menu at the bottom of the left sidebar. Additionally, the version 'stable' refers to the most recent release (installed via `conda`), and 'latest' refers to the latest unstable development version (the `develop` branch).

CLIMADA python:

- [online](#) (recommended)

- PDF file
- core Tutorials on GitHub

CLIMADA petals:

- online (recommended)
- PDF file
- petals Tutorials on GitHub

The documentation can also be [built locally](#).

13.3 Citing CLIMADA

See the [Citation Guide](#).

Please use the following logo if you are presenting results obtained with or through CLIMADA:

13.4 Contributing

We welcome any contribution to this repository, be it bugfixes and other code changes and additions, documentation improvements, or tutorial updates.

If you would like to contribute, please refer to the CLIMADA Project's [Developer Guide](#).

13.5 Versioning

We use [SemVer](#) for versioning. For the versions available, see the [releases on this repository](#).

13.6 License

Copyright (C) 2017 ETH Zurich, CLIMADA contributors listed in AUTHORS.

CLIMADA is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License Version 3, 29 June 2007 as published by the Free Software Foundation, <https://www.gnu.org/licenses/gpl-3.0.html>

CLIMADA is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details: <https://www.gnu.org/licenses/gpl-3.0.html>

CHANGELOG

14.1 6.2.0

Release date: 2026-06-15

14.1.1 Dependency Changes

Added:

- `pymoo`

14.1.2 Added

- Module to compute multi-country optimised CATBonds. #186

14.2 6.1.0

Release date: 2025-09-30

14.2.1 Dependency Changes

Added:

- `lxml >=5` (was implicitly part of the dependency tree before)

Updated:

- `boario >=0.5,<0.6 → >=0.6.2` # Previous versions are not compatible anymore with the refactor.

14.2.2 Added

- Module `climada_petals.hazard.coastal_flood` with new Hazard class `CoastalFlood` and associated method `from_aqueduct_tif` to read coastal flood data directly from the Aqueduct tif files #100

14.2.3 Changed

- Module `climada_petals.hazard.river_flood`, class `RiverFlood`: New method `from_aqueduct_tif` to read river flood data directly from the online Aqueduct tif files, and renamed methods `from_nc` to `from_isimip_nc` and `set_from_nc` to `set_from_isimip_nc` [#108](https://github.com/CLIMADA-project/climada_petals/pull/108)
- Complete refactor of the `supplychain` module. See #159 for details.

14.2.4 Fixed

- Add `lxml` as explicit dependency #168

14.2.5 Removed

- Removed code of the copernicus interface module #142, including code added in #150, #151 and #156. The module has been moved to a separate repository.

14.3 6.0.1

Release date: 2025-03-03

14.3.1 Dependency Changes

- `climada >=5.0 → >=6.0`

14.4 6.0.0

Release date: 2025-03-03

14.4.1 Dependency Changes

Added:

- `boario >=0.5,<0.6`
- `meson >=1.4,<1.5`

Updated:

- `cdsapi >=0.6 → >=0.7`
- `esmpy !=8.4.* → >=8.4.2`
- `scikit-image >=0.22 → >=0.25`

Removed:

- `gfortran`: this conda package conflicts with, e.g., `esmf`, at least on Windows. For using the `climada_petals.hazard.tc_surge_clawpack` module fortran must be installed natively on the local machine.

14.4.2 Added

- Included `boario` in the `supplychain` module #81
- Added a Copernicus download function `downloader.py` as part of the `climada.hazard.copernicus_interface` module under construction #150
- Update name of IBTrACS file to version 4.1 in `tc_rainfield` and `tc_surge_bathub` tests. #152

14.5 5.0.0

Release date: 2024-07-19

14.5.1 Dependency Changes

Added:

- `cdsapi >=0.7`
- `importlib-metadata <8.0`
- `meson-python >=0.15,<0.16`
- `rioxarray >=0.13`
- `ruamel.yaml >=0.18`
- `seaborn >=0.13`
- `xesmf >=0.8`

14.5.2 Changed

- Adaptations to refactoring of the `climada.hazard.Centroids` class, to be compatible with `climada>=5.0` #122
- Always assign `csr_matrix` to `Hazard.intensity` #129 #131

14.5.3 Fixed

- Fix `climada.hazard.tc_rainfield` for TC tracks crossing the antimeridian #105
- Update the table of content for the tutorials #125
- Store all-zero fraction matrices in `LowFlow` and `WildFire` hazards #129 #131
- `eeioa` approach from the supply chain module. See [associated discussion](#).

14.6 4.1.0

Release date: 2024-02-19

14.6.1 Dependency Changes

Updated:

- `climada >=4.0 → ==4.1`

Added:

- `overpy >=0.7`
- `osm-flex >=1.1.1`
- `pymrio >=0.5`

14.6.2 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

14.7 4.0.2

Release date: 2023-09-27

14.7.1 Dependency Changes

- `pandas >=1.5,<2.0 → >=1.5` (compatibility with pandas 2.x)

14.7.2 Changed

- improved integration tests for notebooks and external data apis

14.7.3 Fixed

- implicit casting from `DataArray` to `int` in reading mehtods made explicit #95

14.8 4.0.1

Release date: 2023-09-06

14.8.1 Fixed

- `TCForecast` now skips “untrackable” TCs when reading multi-message `.bufr` files #91

14.9 4.0.0

Release date: 2023-09-01

14.9.1 Dependency Changes

Upgraded:

- `shapely 1.8 -> 2.0` (#80)

14.9.2 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

14.9.3 Removed

- `tag` attribute from hazard classes #88

14.10 v3.3.2

Release date: 2023-08-25

14.10.1 Description

Patch release

14.11 v3.3.1

Release date: 2023-08-24

14.11.1 Description

Rearranged file-system structure: data subdirectory of `climada_petals`.

14.12 v3.3.0

Release date: 2023-05-08

14.12.1 Description

Release aligned with `climada (core) 3.3`.

14.12.2 Added

- Changelog and PR description template based on the Climada Core repository #72

14.12.3 Changed

- Rework docs and switch to Book theme #63

14.12.4 Fixed

- fix issue #69 Warn.zeropadding for islands `https://github.com/CLIMADA-project/climada_petals/pull/70`_

CLIMADA LIST OF AUTHORS

- Gabriela Aznar-Siguan
- David N. Bresch
- Samuel Eberenz
- Jan Hartman
- Marine Perus
- Thomas Röösl
- 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 Stalhandske
- Lukas Riedel
- Samuel Juhel
- Jere Lehtomaa

- Valentin Gebhard
- Dahyann Araya

PYTHON MODULE INDEX

C

76
climada_petals.engine.cat_bonds.helpers_tutorial_cat_bonds, 67
9
climada_petals.engine.cat_bonds.premium_class, 55
9
climada_petals.engine.cat_bonds.sng_bond_simulation, 61
11
climada_petals.engine.cat_bonds.subarea_calculations, 61
15
climada_petals.engine.cat_bonds.subareas, 61
13
climada_petals.engine.cat_bonds.utils_cat_bonds, 61
17
climada_petals.engine.supplychain, 17
climada_petals.engine.warn, 26
climada_petals.entity.exposures.black_marble, 31
31
climada_petals.entity.exposures.crop_production, 33
33
climada_petals.entity.exposures.gdp_asset, 40
40
climada_petals.entity.exposures.osm_data_loader, 41
41
climada_petals.entity.exposures.spam_agrar, 41
41
climada_petals.entity.impact_funcs.drought, 43
43
climada_petals.entity.impact_funcs.relative_cropyield, 44
44
climada_petals.entity.impact_funcs.river_flood, 45
45
climada_petals.entity.impact_funcs.wildfire, 46
46
climada_petals.hazard.drought, 71
climada_petals.hazard.emulator.const, 47
climada_petals.hazard.emulator.emulator, 48
climada_petals.hazard.emulator.geo, 51
climada_petals.hazard.emulator.random, 52
climada_petals.hazard.emulator.stats, 53
climada_petals.hazard.landslide, 71
climada_petals.hazard.low_flow, 73
climada_petals.hazard.relative_cropyield, 76
climada_petals.hazard.rf_glofas.rf_glofas, 67
climada_petals.hazard.rf_glofas.river_flood_computation, 55
climada_petals.hazard.rf_glofas.setup, 68
climada_petals.hazard.rf_glofas.transform_ops, 61
climada_petals.hazard.river_flood, 78
climada_petals.hazard.tc_rainfield, 81
climada_petals.hazard.tc_surge_bathtub, 85
climada_petals.hazard.tc_tracks_forecast, 86
climada_petals.hazard.wildfire, 88
climada_petals.util.config, 92
climada_petals.util.constants, 92