
CLIMADA documentation

Release 3.0.1

CLIMADA contributors

Oct 08, 2021

CONTENTS

1	Getting started with CLIMADA	3
2	Introduction	5
3	Installation	7
4	Using Climada on the Euler Cluster (ETH internal)	11
5	Tutorials	19
6	Developer guide	213
7	Software documentation	271
8	License	413
	Python Module Index	415
	Index	417

This is the documentation for version v3.0.1. In [CLIMADA-project](#) you will find CLIMADA's contributors, repository and scientific publications.

GETTING STARTED WITH CLIMADA

This is a short summary of the guides to help you find the information that you need to get started. To learn more about CLIMADA, have a look at the [introduction](#). You can also have a look at the paper [repository](#) to get an overview of research projects.

1.1 Installation

The first step to getting started is installing CLIMADA. To do so you will need: 1. To get the latest release from the git repository [CLIMADA releases](#) or clone the project with git if you are interested in contributing to the development. 2. To build a conda environment with the dependencies needed by CLIMADA.

For details see the [Installation Guide](#).

If you need to run a model on a computational cluster, have a look at [this guide](#) to install CLIMADA and run your jobs.

1.2 Programming in Python

It is best to have some basic knowledge of Python programming before starting with CLIMADA. But if you need a quick introduction or reminder, have a look at the short [Python Tutorial](#). Also have a look at the python [Python Dos and Don't](#) guide and at the [Python Performance Guide](#) for best practice tips.

1.3 Tutorials

A good way to start using CLIMADA is to have a look at the [Tutorials](#). The [Main Tutorial](#) will introduce you the structure of CLIMADA and how to calculate your first impacts, as well as your first appraisal of adaptation options. You can then look at the specific tutorials for each module (for example if you are interested in a specific hazard, like [Tropical Cyclones](#), or in learning to [estimate the value of asset exposure](#),...).

1.4 Contributing

If you would like to participate in the development of CLIMADA, carefully read the [Git and Development Guide](#). Before making a new feature, discuss with one of the repository admins (Now Chahan, Emmanuel and David). Every new feature or enhancement should be done on a separate branch, which will be merged in the develop branch after being reviewed (see [Checklist](#)). Finally, the develop branch is merged in the main branch in each CLIMADA release. Each new feature should come with a tutorial and with [Unit and Integration Tests](#).

1.5 Other Questions

If you have any other questions, you might find some information in the [Miscellaneous guide](#). If you cannot find you answer in the guides, you can open an [issue](#) for somebody to help you.

INTRODUCTION

CLIMADA implements a fully probabilistic risk assessment model. According to the IPCC [1], natural risks emerge through the interplay of climate and weather-related hazards, the exposure of goods or people to this hazard, and the specific vulnerability of exposed people, infrastructure and environment. The unit chosen to measure risk has to be the most relevant one in a specific decision problem, not necessarily monetary units. Wildfire hazard might be measured by burned area, exposure by population or replacement value of homes and hence risk might be expressed as number of affected people in the context of evacuation, or repair cost of buildings in the context of property insurance.

Risk has been defined by the International Organization for Standardization as the “effect of uncertainty on objectives” as the potential for consequences when something of value is at stake and the outcome is uncertain, recognizing the diversity of values. Risk can then be quantified as the combination of the probability of a consequence and its magnitude:

$$risk = probability \times severity$$

In the simplest case, \times stands for a multiplication, but more generally, it represents a convolution of the respective distributions of probability and severity. We approximate the *severity* as follows:

$$severity = F(hazard\ intensity, exposure, vulnerability) = exposure * f_{imp}(hazard\ intensity)$$

where f_{imp} is the impact function which parametrizes to what extent an exposure will be affected by a specific hazard. While ‘vulnerability function’ is broadly used in the modelers community, we refer to it as ‘impact function’ to explicitly include the option of opportunities (i.e. negative damages). Using this approach, CLIMADA constitutes a platform to analyse risks of different hazard types in a globally consistent fashion at different resolution levels, at scales from multiple kilometres down to meters, depending on the purpose.

2.1 References

[1] IPCC: Climate Change 2014: Impacts, Adaptation and Vulnerability. Part A: Global and Sectoral Aspects. Contribution of Working Group II to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, edited by C. B. Field, V. R. Barros, D. J. Dokken, K. J. Mach, M. D. Mastrandrea, T. E. Bilir, M. Chatterjee, K. L. Ebi, Y. O. Estrada, R. C. Genova, B. Girma, E. S. Kissel, A. N. Levy, S. MacCracken, P. R. Mastrandrea, and L. L. White, Cambridge University Press, United Kingdom and New York, NY, USA., 2014.

INSTALLATION

Please execute the instructions of the following text boxes in a Terminal or Anaconda Prompt.

3.1 Download CLIMADA and CLIMADA_petals

```
cd $[installation directory]

git clone https://github.com/CLIMADA-project/climada_python.git
cd climada_python
git checkout develop
cd ..

git clone https://github.com/CLIMADA-project/climada_petals.git
cd climada_petals
git checkout develop
cd ..
```

3.2 Install environment with Anaconda or Miniconda

3.2.1 Conda

It's highly recommended to use [Conda](#) for the installation. Without Conda the installation of the dependencies can be cumbersome. If it is not already installed, download the latest version of [Anaconda](#) or [Miniconda](#) and execute it.

3.2.2 Install dependencies

Open a command prompt (Windows) or shell (Mac, Linux)

```
cd $[installation directory]
conda env create -n climada_env -f climada_petals/requirements/env_climada.yml
conda activate climada_env
```

3.2.3 Install climada core and petals

```
cd ${installation directory}
pip install -e climada_python
pip install -e climada_petals
```

3.2.4 Test installation

Before leaving the *Environments* section of Anaconda, make sure that the CLIMADA environment, *climada_env* is selected. Go to the *Home* section of Anaconda and install and launch Spyder (or your preferred editor). Open the file containing all the installation tests, `tests_install.py` in `climada_python-x.y.z` folder and execute it. If the installation has been successful, an OK will appear at the end (the execution should last less than 2 min).

3.2.5 Run tutorials

In the *Home* section of Anaconda, with *CLIMADA_env* selected, install and launch *jupyter notebook*. A browser window will show up. Navigate to your `CLIMADA_python-x.y.z` repository and open `doc/tutorial/1_main_CLIMADA.ipynb`. This is the tutorial which will guide you through all CLIMADA's functionalities. Execute each code cell to see the results, you might also edit the code cells before executing. See [tutorials](#) for more information.

3.2.6 Workaround if Spyder installation in Anaconda fails for *climada_env*.

In a terminal window, enter:

```
conda activate climada_env
conda install spyder-kernels
python -c "import sys; print(sys.executable)"
--> returns a path, like /Users/XXX/opt/anaconda3/envs/climada_env/bin/python
```

Start Anaconda, create a new `python_env` environment (just click create and enter the name, then press Create) and install latest Spyder in there (currently 4.1.5), start this Spyder, then, after Spyder has started, navigate to Preferences > Python Interpreter > Use the following interpreter and paste the path from above (e.g. `/Users/XXX/opt/anaconda3/envs/CLIMADA_env/bin/python`) into the text box. Start a new IPython console and run `tests_install.py`.

3.3 FAQs

- `ModuleNotFoundError`; CLIMADA libraries are not found. Try to include *climada_python-x.y.z* path in the environment *climada_env* path as suggested in Section 2 of [Install environment with Anaconda]. If it does not work you can always include the path manually before executing your code:

```
import sys
sys.path.append('path/to/climada_python-x.y.z')
```

- `ModuleNotFoundError`; some python library is not found. It might happen that the pip dependencies of *env_climada.yml* (the ones specified after `pip:`) have not been installed in the environment *climada_env*. You can then install them manually one by one as follows:

```
conda activate climada_env  
pip install library_name
```

where `library_name` is the missing library.

Another reason may be a recent update of the operating system (macOS). In this case removing and reinstalling Anaconda will be required.

- Conda permission error (operation not permitted) in macOS Mojave: try the solutions suggested here <https://github.com/conda/conda/issues/8440>
- No 'impf_TC' column in GeoDataFrame: This may happen when a demo file from CLIMADA was not updated after the change in the impact function naming pattern from 'if_' to 'impf_' ([climada v2.2.0](#)). To solve it, run `python -c 'import climada; climada.setup_climada_data(reload=True)` in a terminal.
- How to change the log level: By default the logging level is set to 'INFO', which is quite verbose. This can be changed
 - programmatically, in a script or interactive python environment (Spyder, Jupyter, IPython) by executing e.g., `from climada.util.config import LOGGER; from logging import WARNING; LOGGER.setLevel(WARNING)`,
 - through configuration, by editing the config file `CLIMADA/conf/defaults.conf` and setting the value of the `global.log_level` property.

USING CLIMADA ON THE EULER CLUSTER (ETH INTERNAL)

4.1 Content

1. *Access to Euler*
2. *Installation and working directories*
3. *Pre-installed version of Climada*
 1. *Load dependencies*
 2. *Check installation*
 3. *Adjust the Climada configuration*
 4. *Run a job*
4. *Working with Git Branches*
 1. *Load dependencies*
 2. *Create installation environment*
 3. *Check out sources*
 4. *Pip install Climada*
 5. *Check installation*
 6. *Adjust the Climada configuration*
 7. *Run a job*
5. *Fallback: Conda installation*
 1. *Conda installation*
 2. *Check out sources*
 3. *Climada environemnt*
 4. *Adjust the Climada configuration*
 5. *Climada sripts*
 6. *Run a job*
6. *Conda Deinstallation*
 1. *Conda*
 2. *Climada*

4.1.1 Access to Euler

See https://scicomp.ethz.ch/wiki/Getting_started_with_clusters for details on how to register at and get started with Euler.

For all steps below, first enter the Cluster via SSH.

Installation- and working directories

Please, get familiar with the various Euler storage options: https://scicomp.ethz.ch/wiki/Storage_systems. As a general rule: use `/cluster/project` for installation and `/cluster/work` for data processing.

For ETH WCR group members, the suggested installation and working directories are `/cluster/project/climate/$USER` and `/cluster/work/climate/$USER` respectively. You may have to create the installation directory:

```
mkdir -p /cluster/project/climate/$USER \
        /cluster/work/climate/$USER
```

4.1.2 Pre-installed version of Climada

Climada is pre-installed and available in the default pip environment of Euler.

4.2 1. Load dependencies

```
env2lmod
module load gcc/6.3.0 python/3.8.5 gdal/3.1.2 geos/3.8.1 proj/7.2.1 libspatialindex/1.8.
↪ 5 hdf5/1.10.1 netcdf/4.4.1.1 eccodes/2.21.0 zlib/1.2.9
```

You need to execute these two lines every time you login to Euler before Climada can be used. To save yourself from doing it manually, one can append these lines to the `~/.bashrc` script, which is automatically executed upon logging in to Euler.

4.3 2. Check installation

```
python -c 'import climada; print(climada.__file__)'
```

should output something like this:

```
/cluster/apps/nss/gcc-6.3.0/python/3.8.5/x86_64/lib64/python3.8/site-packages/climada/___
↪ init__.py
```


4.4 3. Adjust the Climada configuration

Edit a configuration file according to your needs (see [Guide_Configuration](#)). Create a climada.conf file e.g., in /cluster/home/\$USER/.config with the following content:

```
{
  "local_data": {
    "system": "/cluster/work/climate/USERNAME/climada/data",
    "demo": "/cluster/project/climate/USERNAME/climada_python/data/demo",
    "save_dir": "/cluster/work/climate/USERNAME/climada/results"
  }
}
```

(Replace USERNAME with your nethz-id.)

4.5 4. Run a job

Please see the Wiki: https://scicomp.ethz.ch/wiki/Using_the_batch_system for an overview on how to use bsub.

```
cd /cluster/work/climate/$USER # change to the working directory
bsub [bsub-options*] python climada_job_script.py # submit the job
```

4.5.1 Working with Git branches

If the Climada version of the default installation is not according to your needs, you can install Climada from a local Git repository.

4.6 1. Load dependencies

See *Load dependencies* above.

4.7 2. Create installation environment

```
python -m venv --system-site-packages /cluster/project/climate/$USER/climada_venv
```

4.8 3. Checkout sources

```
cd /cluster/project/climate/$USER
git clone https://github.com/CLIMADA-project/climada_python.git
cd climada_python
git checkout develop # i.e., your branch of interest
```

4.9 4. Pip install Climada

```
source /cluster/project/climate/$USER/clinada_venv/bin/activate
pip install -e /cluster/project/climate/$USER/clinada_python
```

4.10 5. Check installation

```
cd /cluster/work/climate/$USER
python -c 'import clinada; print(clinada.__file__)'
```

should output exactly this (with explicit \$USER):

```
/cluster/project/climate/$USER/clinada_python/clinada/__init__.py
```

4.11 6. Adjust the Climada configuration

See *Adjust the Climada configuration* above.

4.12 7. Run a job

See *Run a job* above.

4.12.1 Fallback: Conda installation

If Climada cannot be installed through pip because of changed dependency requirements, there is still the possibility to install Climada through the Conda environment. > **WARNING:** This approach is highly discouraged, as it imposes a heavy and mostly unnecessary burden on the file system of the cluster.

4.13 1. Conda Installation

Download or update to the latest version of [Miniconda](#). Installation is done by execution of the following steps:

```
cd /cluster/project/climate/USERNAME
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
miniconda3/bin/conda init
rm Miniconda3-latest-Linux-x86_64.sh
```

During the installation process of Miniconda, you are prompted to set the working directory according to your choice. Set it to `/cluster/project/climate/USERNAME/miniconda3`. Once the installation has finished, log out of Euler and in again. The command prompt should be preceded by `(base)`, indicating that the installation was a success and that you login in into conda's base environment by default.

4.14 2. Checkout sources

See *Checkout sources* above.

4.15 3. Climada Environment

Create the conda environment:

```
cd /cluster/project/climate/USERNAME/climada_python
conda env create -f requirements/env_climada.yml --name climada_env
conda env update -n climada_env -f requirements/env_developer.yml

conda activate climada_env
conda install conda-build
conda develop .
```

4. Adjust the Climada configuration

See *Adjust the Climada configuration* above.

4.16 5. Climada Scripts

Create a bash script for executing python scripts in the climada environment, `climadajob.sh`:

```
#!/bin/bash
PYTHON_SCRIPT=$1
shift
. ~/.bashrc
conda activate climada_env
python $PYTHON_SCRIPT $@
echo $PYTHON_SCRIPT completed
```

Make it executable:

```
chmod +x climadajob.sh
```

Create a python script that executes climada code, e.g., `climada_smoke_test.py`:

```
import sys
from climada import CONFIG, SYSTEM_DIR
from climada.util.test.test_finance import TestNetpresValue
TestNetpresValue().test_net_pres_val_pass()
print(SYSTEM_DIR)
print(CONFIG.local_data.save_dir.str())
print("the script ran with arguments", sys.argv)
```

4.17 6. Run a Job

Please see the Wiki: https://scicomp.ethz.ch/wiki/Using_the_batch_system.

With the scripts from above you can submit the python script as a job like this:

```
bsub [options] /path/to/climadajob.sh /path/to/climada_smoke_test.py arg1 arg2
```

After the job has finished the lsf output file should look something like this:

```
Sender: LSF System <lsfadmin@eu-ms-010-32>
Subject: Job 161617875: <./climada_job.sh climada_smoke_test.py arg1 arg2> in cluster
-><euler> Done

Job <./climada_job.sh climada_smoke_test.py arg1 arg2> was submitted from host <eu-login-
->41> by user <USERNAME> in cluster <euler> at Thu Jan 28 14:10:15 2021
Job was executed on host(s) <eu-ms-010-32>, in queue <normal.4h>, as user <USERNAME> in
->cluster <euler> at Thu Jan 28 14:10:42 2021
</cluster/home/USERNAME> was used as the home directory.
</cluster/work/climate/USERNAME> was used as the working directory.
Started at Thu Jan 28 14:10:42 2021
Terminated at Thu Jan 28 14:10:53 2021
Results reported at Thu Jan 28 14:10:53 2021

Your job looked like:

-----
# LSBATCH: User input
./climada_job.sh climada_smoke_test.py arg1 arg2
-----

Successfully completed.

Resource usage summary:

CPU time :                2.99 sec.
Max Memory :              367 MB
Average Memory :          5.00 MB
Total Requested Memory : 1024.00 MB
Delta Memory :            657.00 MB
Max Swap :                -
Max Processes :           5
Max Threads :             6
Run time :                22 sec.
Turnaround time :         38 sec.

The output (if any) follows:

/cluster/project/climate/USERNAME/miniconda3/envs/climada/lib/python3.7/site-packages/
->pandas_datareader/compat/__init__.py:7: FutureWarning: pandas.util.testing is
->deprecated. Use the functions in the public API at pandas.testing instead.
    from pandas.util.testing import assert_frame_equal
/cluster/work/climate/USERNAME/climada/data
```

(continues on next page)

(continued from previous page)

```
/cluster/work/climate/USERNAME/clinada/results  
the script ran with arguments ['/path/to/clinada_smoke_test.py', 'arg1' 'arg2']  
python_script.sh completed
```

Conda Deinstallation

4.18 1. Conda

Remove the miniconda3 directory from the installation directory:

```
rm -rf /cluster/project/climate/USERNAME/miniconda3/
```

Delete the conda related parts from `/cluster/home/USERNAME/.bashrc`, i.e., everything between

```
# >>> conda initialize >>>  
and  
# <<< conda initialize <<<
```

4.19 2. Climada

Remove the climada sources and config file:

```
rm -rf /cluster/project/climate/USERNAME/clinada_python  
rm -f /cluster/home/USERNAME/clinada.conf /cluster/home/USERNAME/*/clinada.conf
```


TUTORIALS

5.1 CLIMADA overview

5.1.1 Contents

- *Introduction*
 - *What is CLIMADA?*
 - *This tutorial*
 - *Resources beyond this tutorial*
- *CLIMADA features*
 - *CLIMADA classes*
- *Tutorial: an example risk assessment*
 - *Hazard*
 - * *Storm tracks*
 - * *Centroids*
 - * *Hazard footprint*
 - *Entity*
 - * *Exposures*
 - * *Impact functions*
 - * *Adaptation measures*
 - * *Discount rates*
 - *Engine*
 - * *Impact*
 - * *Adaptation options appraisal*

5.1.2 Introduction

What is CLIMADA?

CLIMADA is a fully probabilistic climate risk assessment tool. It provides a framework for users to combine exposure, hazard and vulnerability or impact data to calculate risk. Users can create probabilistic impact data from event sets, look at how climate change affects these impacts, and see how effectively adaptation measures can change them. CLIMADA also allows for studies of individual events, historical event sets and forecasts.

The model is a highly customisable, meaning that users can work with out-of-the-box data provided for different hazards, population and economic exposure, or can provide their own data for part or all of the analysis. The pre-packaged data make CLIMADA particularly useful for users who focus on just one element of risk, since CLIMADA can ‘fill in the gaps’ for hazard, exposure or vulnerability in the rest of the analysis.

The model core is designed to give as much flexibility as possible when describing the elements of risk, meaning that CLIMADA isn’t limited to particular hazards, exposure types or impacts. We love to see the model applied to new problems and contexts.

CLIMADA provides classes, methods and data for exposure, hazard and impact functions (also called vulnerability functions), plus a financial model and a framework to analyse adaptation measures. Additional classes and data for common uses, such as economic exposures or tropical storms. Tutorials for every class are available: see the [CLIMADA features](#) section below.

This tutorial

This tutorial is for people new to CLIMADA who want to get a high level understanding of the model and work through an example risk analysis. It will list the current features of the model, and go through a complete CLIMADA analysis to give an idea of how the model works. Other tutorials go into more detail about different model components and individual hazards.

Resources beyond this tutorial

- [Installation guide](#) - go here if you’ve not installed the model yet
- [CLIMADA Read the Docs home page](#) - for all other documentation
- [List of CLIMADA’s features and associated tutorials](#)
- [CLIMADA GitHub develop branch documentation](#) for the very latest versions of code and documentation
- [CLIMADA paper GitHub repository](#) - for publications using CLIMADA

5.1.3 CLIMADA features

A risk analysis with CLIMADA can include

1. the statistical risk to your exposure from a set of events,
2. how it changes under climate change, and
3. a cost-benefit analysis of adaptation measures.

CLIMADA is flexible: the “statistical risk” above could be describing the annual expected insured flood losses to a property portfolio, the number of people displaced by an ensemble of typhoon forecasts, the annual disruption to a railway network from landslides, or changes to crop yields.

Users from risk-analysis backgrounds will be familiar with describing the impact of events by combining exposure, hazard and an impact function (or vulnerability curve) that combines the two to describe a hazard's effects. A CLIMADA analysis uses the same approach but wraps the exposures and their impact functions into a single `Entity` class, along with discount rates and adaptation options (see the below tutorials for more on CLIMADA's financial model).

CLIMADA's `Impact` object is used to analyse events and event sets, whether this is the impact of a single wildfire, or the global economic risk from tropical cyclones in 2100.

CLIMADA classes

This is a full directory of tutorials for CLIMADA's classes to use as a reference. You don't need to read all this to do this tutorial, but it may be useful to refer back to.

- **Hazard**: a class that stores sets of geographic hazard footprints, (e.g. for wind speed, water depth and fraction, drought index), and metadata including event frequency. Several predefined extensions to create particular hazards from particular datasets and models are included with CLIMADA:
- **Tropical cyclone wind**: global hazard sets for tropical cyclone events, constructing statistical wind fields from storm tracks. Subclasses include methods and data to calculate historical wind footprints, create forecast ensembles from ECMWF tracks, and create climatological event sets for different climate scenarios.
- **Storm surge**: (under development) global surge hazard for tropical storms. Runs the GeoClaw surge model to create and plot hazard from tropical storm tracks
- **European windstorms**: includes methods to read and plot footprints from the Copernicus WISC dataset and for DWD and ICON forecasts.
- **River flooding**: global water depth hazard for flood, including methods to work with ISIMIP simulations.
- **Crop modelling**: combines ISIMIP crop simulations and UN Food and Agriculture Organization data. The module uses crop production as exposure, with hydrometeorological 'hazard' increasing or decreasing production.
- **Wildfire (global)**: tutorial under development
- **Drought (global)**: tutorial under development
- **Entity**: this is a container that groups CLIMADA's socio-economic models. It's where the Exposures and Impact Functions are stored, which can then be combined with a hazard for a risk analysis (using the Engine's `Impact` class). It is also where Discount Rates and Measure Sets are stored, which are used in adaptation cost-benefit analyses (using the Engine's `CostBenefit` class):
- **Exposures**: geolocated exposures. Each exposure is associated with a value (which can be a dollar value, population, crop yield, etc), information to associate it with impact functions for the relevant hazard(s) (in the Entity's `ImpactFuncSet`), a geometry, and other optional properties such as deductables and cover. Exposures can be loaded from a file, specified by the user, or created from regional economic models accessible within CLIMADA, for example:
 - **LitPop**: regional economic model using nightlight and population maps together with several economic indicators
 - **BlackMarble**: regional economic model from nightlight intensities and economic indicators (GDP, income group). Largely succeeded by LitPop.
 - **OpenStreetMap**: methods to create exposures from data available through the OpenStreetMap API
- **ImpactFuncSet**: functions to describe the impacts that hazards have on exposures, expressed in terms of e.g. the % dollar value of a building lost as a function of water depth, or the mortality rate for over-70s as a function of temperature. CLIMADA provides some common impact functions, or they can be user-specified. The following is an incomplete list:
 - **ImpactFunc**: a basic adjustable impact function, specified by the user

- IFropCyclone: impact functions for tropical cyclone winds
- IFRiverFlood: impact functions for river floods
- IFStormEurope: impact functions for European windstorms
- **DiscRates**: discount rates per year
- **MeasureSet**: a collection of Measure objects that together describe any adaptation measures being modelled. Adaptation measures are described by their cost, and how they modify exposure, hazard, and impact functions (and have a method to do these things). Measures also include risk transfer options.
- ****Engine****: the CLIMADA Engine contains the Impact and CostBenefit classes, which are where the main model calculations are done, combining Hazard and Entity objects.
 - **Impact**: a class that stores CLIMADA's modelled impacts and the methods to calculate them from Exposure, Impact Function and Hazard classes. The calculations include average annual impact, expected annual impact by exposure item, total impact by event, and (optionally) the impact of each event on each exposure point. Includes statistical and plotting routines for common analysis products.
 - **CostBenefit**: a class to appraise adaptation options. It uses an Entity's MeasureSet to calculate new Impacts based on their adjustments to hazard, exposure, and impact functions, and returns statistics and plotting routines to express cost-benefit comparisons.

This list will be updated periodically along with new CLIMADA releases. To see the latest, development version of all tutorials, see the [tutorials page on the CLIMADA GitHub](#).

5.1.4 Tutorial: an example risk assessment

This example will work through a risk assessment for tropical storm wind in Puerto Rico, constructing hazard, exposure and vulnerability and combining them to create an Impact object. Everything you need for this is included in the main CLIMADA installation and additional data will be downloaded by the scripts as required.

5.1.5 Hazard

Hazards are characterized by their frequency of occurrence and the geographical distribution of their intensity. The Hazard class collects events of the same hazard type (e.g. tropical cyclone, flood, drought, ...) with intensity values over the same geographic centroids. They might be historical events or synthetic.

See the [Hazard tutorial](#) to learn about the Hazard class in more detail, and the [CLIMADA features](#) section of this document to explore tutorials for different hazards, including [tropical cyclones](#), as used here.

Tropical cyclones in CLIMADA and the TropCyclone class work like any hazard, storing each event's wind speeds at the geographic centroids specified for the class. Pre-calculated hazards can be loaded from files (see the [full Hazard tutorial](#)), but they can also be modelled from a storm track using the TCTracks class, based on a storm's parameters at each time step. This is how we'll construct the hazards for our example.

So before we create the hazard, we will create our storm tracks and define the geographic centroids for the locations we want to calculate hazard at.

Storm tracks

Storm tracks are created and stored in a separate class, `TCTracks`. We use its method `read_ibtracs_netcdf` to create the tracks from the `IBTRaCS` storm tracks archive. The first time this runs on your machine it will need to download the full dataset which might take a little time. See the [full TropCyclone tutorial](#) for more detail and troubleshooting.

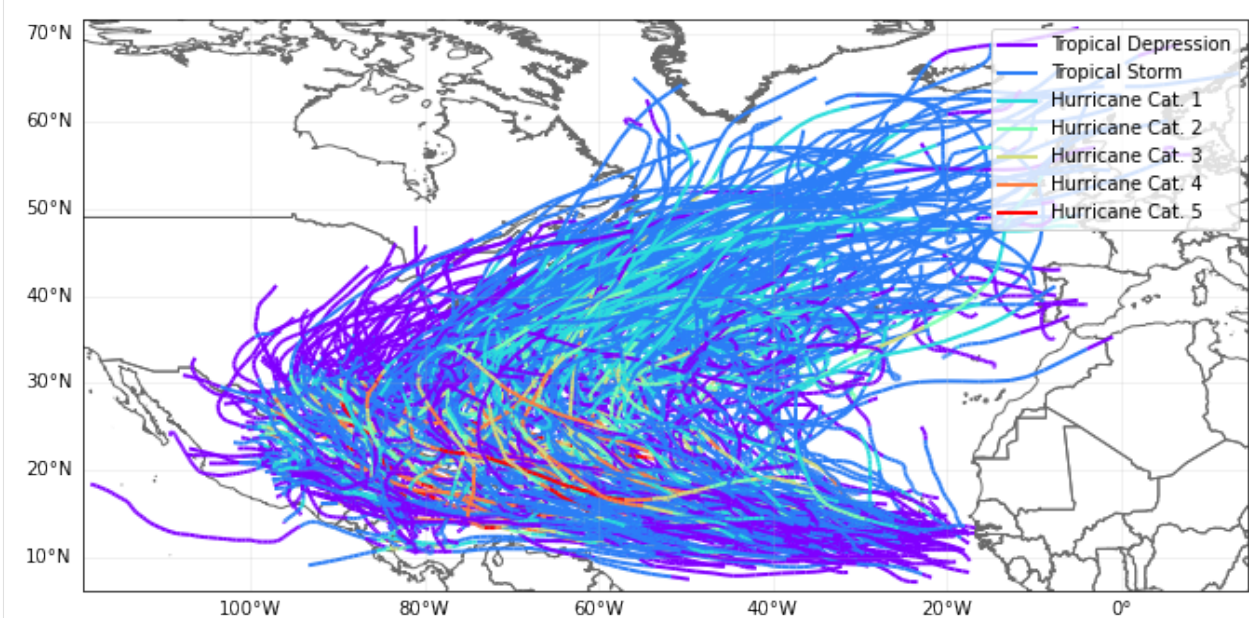
```
[1]: import numpy as np
from climada.hazard import TCTracks

tracks = TCTracks()
tracks.read_ibtracs_netcdf(provider='usa', basin='NA')

2021-04-23 11:47:38,480 - climada.hazard.tc_tracks - WARNING - 57 storm events are_
↳discarded because no valid wind/pressure values have been found: 1980199N31284,
↳1980200N25270, 1980204N23287, 1980226N15339, 1980238N16328, ...
2021-04-23 11:47:39,308 - climada.hazard.tc_tracks - INFO - Progress: 10%
2021-04-23 11:47:40,110 - climada.hazard.tc_tracks - INFO - Progress: 20%
2021-04-23 11:47:40,888 - climada.hazard.tc_tracks - INFO - Progress: 30%
2021-04-23 11:47:41,716 - climada.hazard.tc_tracks - INFO - Progress: 40%
2021-04-23 11:47:42,679 - climada.hazard.tc_tracks - INFO - Progress: 50%
2021-04-23 11:47:44,076 - climada.hazard.tc_tracks - INFO - Progress: 60%
2021-04-23 11:47:45,066 - climada.hazard.tc_tracks - INFO - Progress: 70%
2021-04-23 11:47:45,889 - climada.hazard.tc_tracks - INFO - Progress: 81%
2021-04-23 11:47:46,700 - climada.hazard.tc_tracks - INFO - Progress: 91%
2021-04-23 11:47:47,410 - climada.hazard.tc_tracks - INFO - Progress: 100%
```

This will load all historical tracks in the North Atlantic into the `tracks` object (since we set `basin='NA'`). The `TCTracks.plot` method will plot the downloaded tracks, though there are too many for the plot to be very useful:

```
[2]: tracks.plot()
[2]: <GeoAxesSubplot:>
```



It's also worth adding additional time steps to the tracks (though this can be memory intensive!). Most tracks are reported at 3-hourly intervals (plus a frame at landfall). Event footprints are calculated as the maximum wind from

any time step. For a fast-moving storm these combined three-hourly footprints give quite a rough event footprint, and it's worth adding extra frames to smooth the footprint artificially (try running this notebook with and without this interpolation to see the effect):

```
[3]: tracks.equal_timestep(time_step_h=0.5)
```

```
2021-04-23 11:48:22,720 - climada.hazard.tc_tracks - INFO - Interpolating 543 tracks to 0.5h time steps.
```

Now, irresponsibly for a risk analysis, we're only going to use these historical events: they're enough to demonstrate CLIMADA in action. A proper risk analysis would expand it to include enough events for a statistically robust climatology. See the [full TropCyclone tutorial](#) for CLIMADA's stochastic event generation.

Centroids

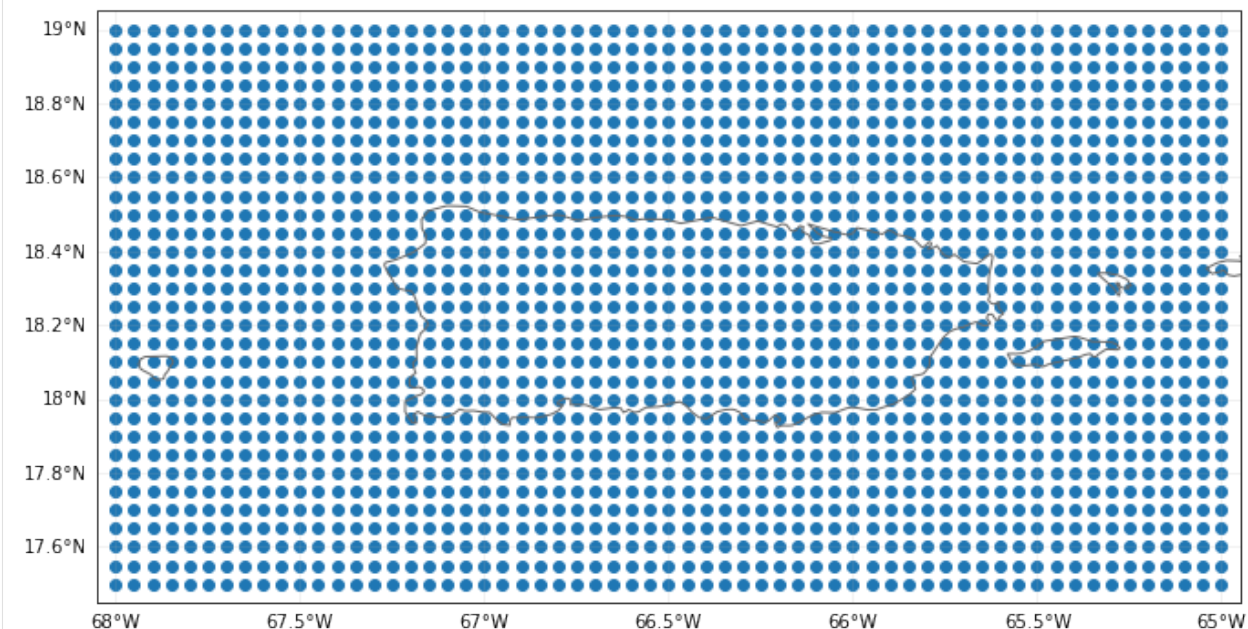
A hazard's centroids can be any set of locations where we want the hazard to be evaluated. This could be the same as the locations of your exposure, though commonly it is on a regular lat-lon grid (with hazard being imputed to exposure between grid points).

Here we'll set the centroids as a 0.1 degree grid covering Puerto Rico. Centroids are defined by a `Centroids` class, which has the `set_raster_from_pnt_bounds` method for generating regular grids and a `plot` method to inspect the centroids.

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

```
min_lat, max_lat, min_lon, max_lon = 17.5, 19.0, -68.0, -65.0
cent = Centroids()
cent.set_raster_from_pnt_bounds((min_lon, min_lat, max_lon, max_lat), res=0.05)
cent.check()
cent.plot()
```

```
[4]: <GeoAxesSubplot:>
```



Almost every class in CLIMADA has a `check()` method, as used above. This verifies that the necessary data for an

objet is correctly provided and logs the optional variables that are not present. It is always worth running it after filling an instance of an object.

Hazard footprint

Now we're ready to create our hazard object. This will be a `TropCyclone` class, which inherits from the `Hazard` class, and has the `from_tracks` constructor method to create a hazard from a `TCTracks` object at given centroids.

```
[5]: from climada.hazard import TropCyclone
```

```
haz = TropCyclone.from_tracks(tracks, centroids=cent)
haz.check()
```

```
2021-04-23 11:48:42,119 - climada.hazard.centroids.centr - INFO - Convert centroids to
↳ GeoSeries of Point shapes.
```

```
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
return _prepare_from_string(" ".join(pjargs))
```

```
2021-04-23 11:48:42,890 - climada.util.coordinates - INFO - dist_to_coast: UTM 32619 (1/
↳ 2)
```

```
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
return _prepare_from_string(" ".join(pjargs))
```

```
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
return _prepare_from_string(" ".join(pjargs))
```

```
2021-04-23 11:48:43,554 - climada.util.coordinates - INFO - dist_to_coast: UTM 32620 (2/
↳ 2)
```

```
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
return _prepare_from_string(" ".join(pjargs))
```

```
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
return _prepare_from_string(" ".join(pjargs))
```



```

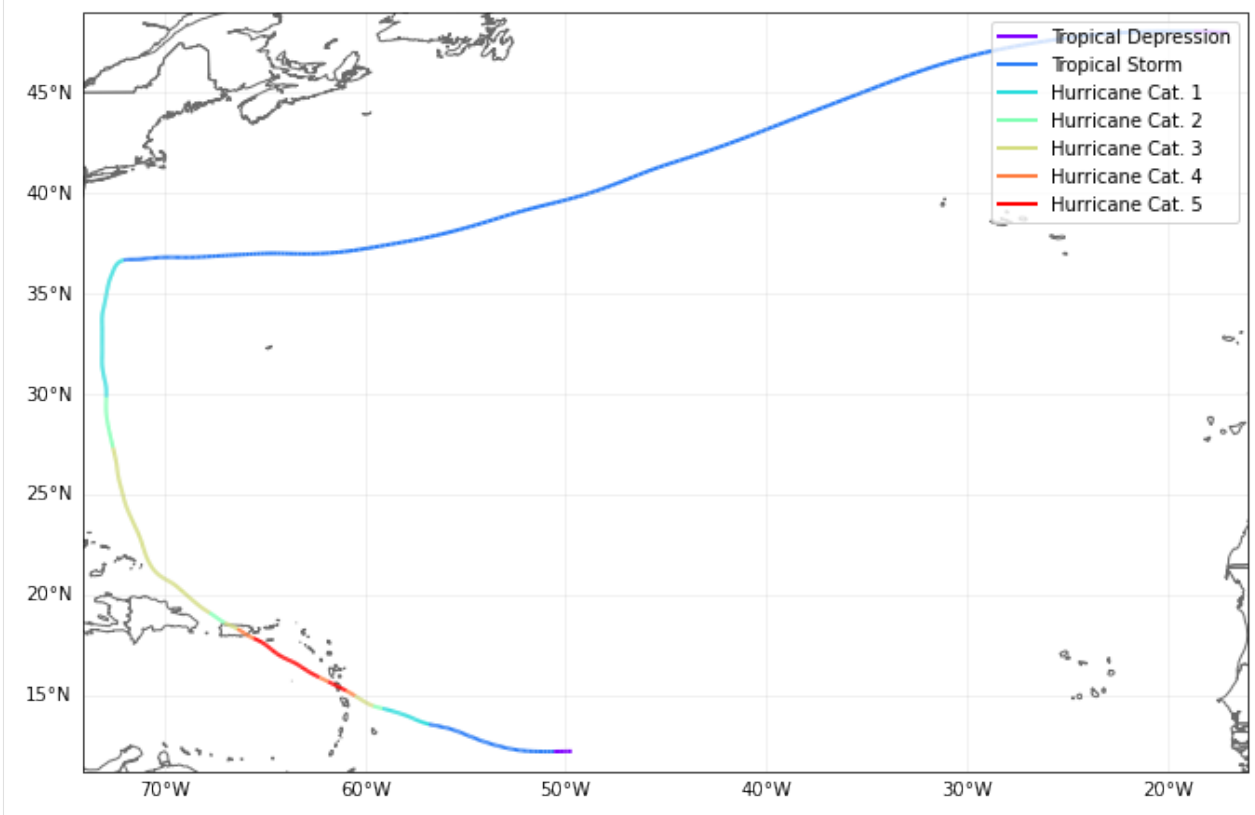
2021-04-23 11:48:43,995 - climada.hazard.trop_cyclone - INFO - Mapping 543 tracks to
↳1891 coastal centroids.
2021-04-23 11:48:46,182 - climada.hazard.trop_cyclone - INFO - Progress: 10%
2021-04-23 11:48:47,872 - climada.hazard.trop_cyclone - INFO - Progress: 20%
2021-04-23 11:48:49,206 - climada.hazard.trop_cyclone - INFO - Progress: 30%
2021-04-23 11:48:51,636 - climada.hazard.trop_cyclone - INFO - Progress: 40%
2021-04-23 11:48:52,977 - climada.hazard.trop_cyclone - INFO - Progress: 50%
2021-04-23 11:48:54,699 - climada.hazard.trop_cyclone - INFO - Progress: 60%
2021-04-23 11:48:56,543 - climada.hazard.trop_cyclone - INFO - Progress: 70%
2021-04-23 11:48:58,320 - climada.hazard.trop_cyclone - INFO - Progress: 81%
2021-04-23 11:49:00,273 - climada.hazard.trop_cyclone - INFO - Progress: 91%
2021-04-23 11:49:02,168 - climada.hazard.trop_cyclone - INFO - Progress: 100%

```

In 2017 Hurricane Maria devastated Puerto Rico. In the IBTRaCs event set, it has ID `2017260N12310` (we use this rather than the name, as IBTRaCS contains three North Atlantic storms called Maria). We can plot the track:

```
[6]: tracks.subset({"sid": "2017260N12310"}).plot() # This is how we subset a TCTracks object
```

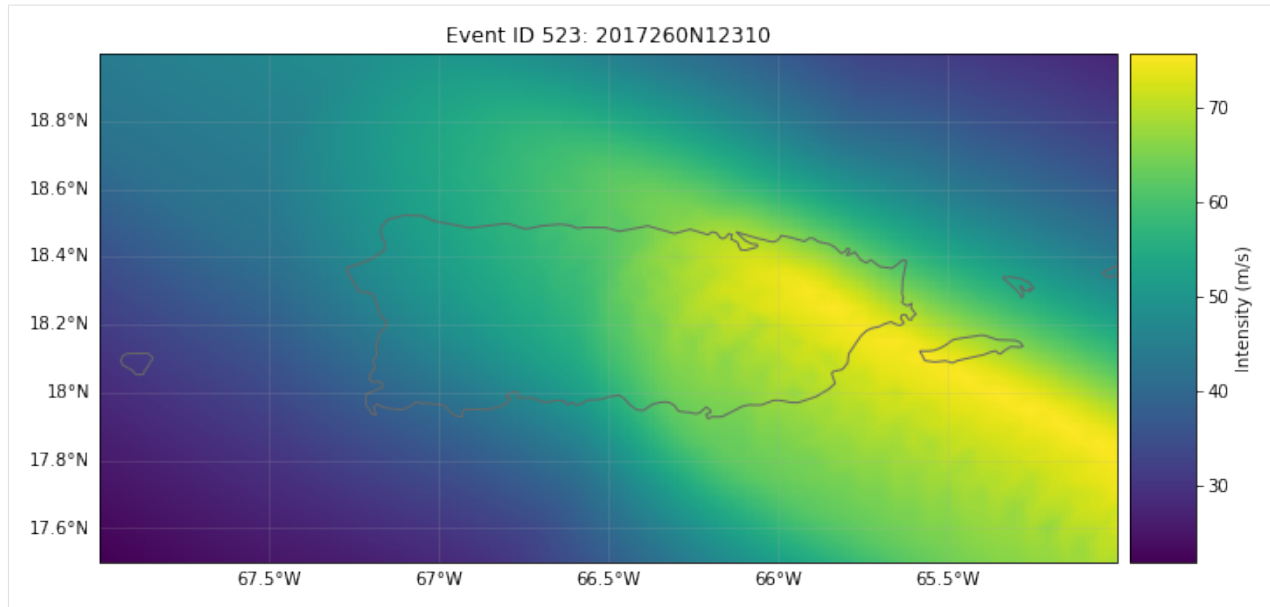
```
[6]: <GeoAxesSubplot:>
```



And plot the hazard on our centroids for Puerto Rico:

```
[7]: haz.plot_intensity(event='2017260N12310')
```

```
[7]: <GeoAxesSubplot:title={'center':'Event ID 523: 2017260N12310'}>
```



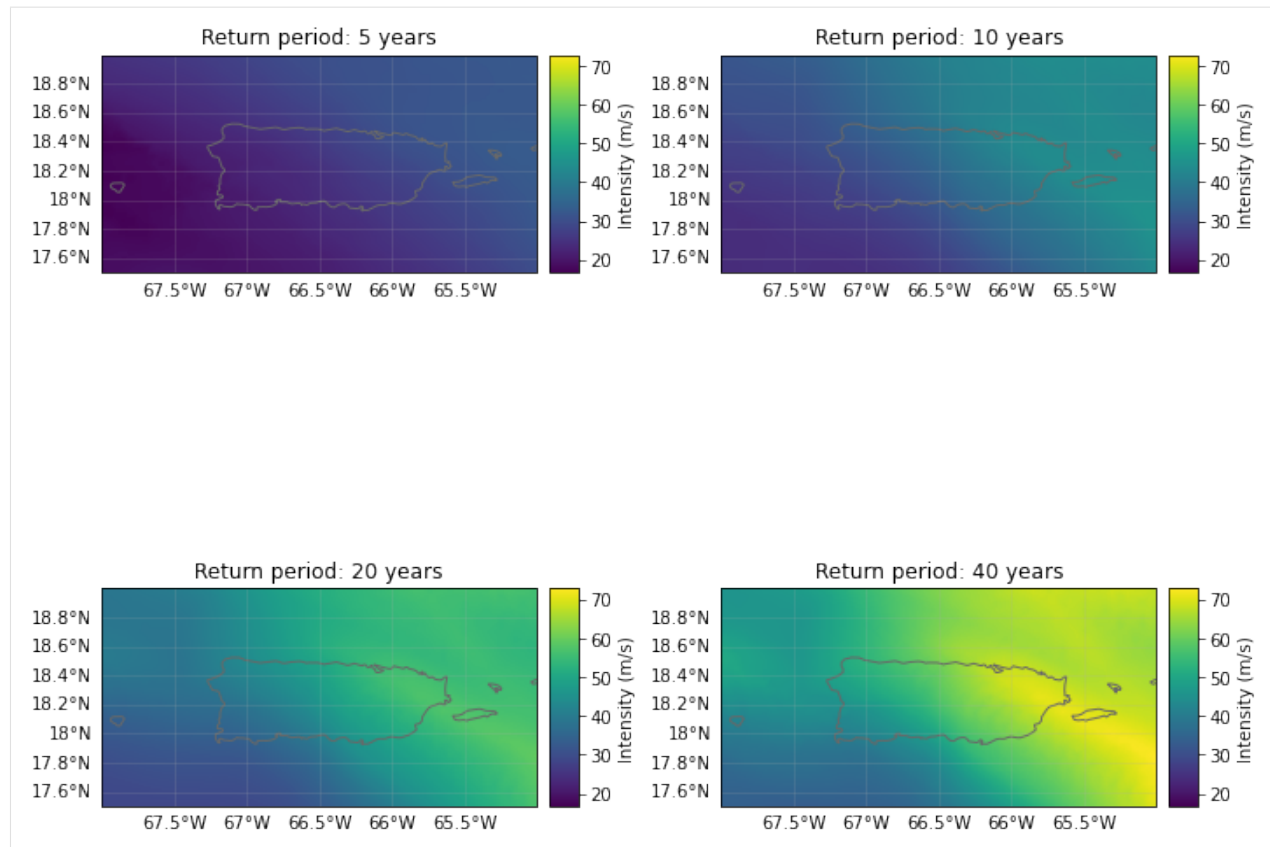
A Hazard object also lets us plot the hazard at different return periods. The IBTRaCS archive produces footprints from 1980 onwards (CLIMADA discarded earlier events) and so the historical period is short. Therefore these plots don't make sense as 'real' return periods, but we're being irresponsible and demonstrating the functionality anyway.

```
[8]: haz.plot_rp_intensity(return_periods=(5,10,20,40))
```

```
2021-04-23 11:49:08,235 - climada.hazard.base - WARNING - Return period 40.0 exceeds max.
↳ event return period.
```

```
2021-04-23 11:49:08,237 - climada.hazard.base - INFO - Computing exceedance intensitiy
↳ map for return periods: [ 5 10 20 40]
```

```
[8]: (array([[<GeoAxesSubplot:title={'center':'Return period: 5 years'}>,
              <GeoAxesSubplot:title={'center':'Return period: 10 years'}>],
              [<GeoAxesSubplot:title={'center':'Return period: 20 years'}>,
              <GeoAxesSubplot:title={'center':'Return period: 40 years'}>]],
              dtype=object),
      array([[24.67411181, 24.82109692, 24.98683063, ..., 29.29464182,
              29.67756411, 29.9150518 ],
              [31.69842124, 31.8197121 , 31.96135634, ..., 42.21941388,
              43.0336463 , 43.2747222 ],
              [38.72273067, 38.81832728, 38.93588204, ..., 55.14418595,
              56.38972848, 56.63439259],
              [45.7470401 , 45.81694246, 45.91040775, ..., 68.06895802,
              69.74581066, 69.99406298]]))
```



See the [TropCyclone tutorial](#) for full details of the TropCyclone hazard class.

We can also recalculate event sets to reflect the effects of climate change. The `set_climate_scenario_knu` method applies changes in intensity and frequency projected due to climate change, as described in ‘Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios’ (Knutson *et al.* 2015). See the [tutorial](#) for details.

Exercise: Extend this notebook’s analysis to examine the effects of climate change in Puerto Rico. You’ll need to extend the historical event set with stochastic tracks to create a robust statistical storm climatology - the TCTracks class has the functionality to do this. Then you can apply the `set_climate_scenario_knu` method to the generated hazard object to create a second hazard climatology representing storm activity under climate change. See how the results change using the different hazard sets.

Next we’ll work on exposure and vulnerability, part of the Entity class.

5.1.6 Entity

The entity class is a container class that stores exposures and impact functions (vulnerability curves) needed for a risk calculation, and the discount rates and adaptation measures for an adaptation cost-benefit analysis.

As with Hazard objects, Entities can be read from files or created through code. The Excel template can be found in `climada_python/data/system/entity_template.xlsx`.

In this tutorial we will create an Exposure object using the LitPop economic exposure module, and load a pre-defined wind damage function.

First we create an empty Entity object:


```
[9]: from climada.entity import Entity

ent = Entity()

2021-04-23 11:49:13,207 - climada.entity.exposures.base - INFO - meta set to default
↳ value {}
2021-04-23 11:49:13,208 - climada.entity.exposures.base - INFO - tag set to default
↳ value File:
Description:
2021-04-23 11:49:13,209 - climada.entity.exposures.base - INFO - ref_year set to default
↳ value 2018
2021-04-23 11:49:13,210 - climada.entity.exposures.base - INFO - value_unit set to
↳ default value USD
2021-04-23 11:49:13,211 - climada.entity.exposures.base - INFO - crs set to default
↳ value: EPSG:4326

/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳ 221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now
↳ deprecated and will not be supported in the future.
self.gdf.crs = self.meta['crs']
```

Exposures

The Entity's exposures attribute contains geolocalized values of anything exposed to the hazard, whether monetary values of assets or number of human lives, for example. It is of type Exposures.

See the [Exposures tutorial](#) for more detail on the structure of the class, and how to create and import exposures. The [LitPop tutorial](#) explains how CLIMADA models economic exposures using night-time light and economic data, and is what we'll use here. To combine your exposure with OpenStreetMap's data see the [OSM tutorial](#).

LitPop is a module that allows CLIMADA to estimate exposed populations and economic assets at any point on the planet without additional information, and in a globally consistent way. Here we can create an economic Exposure dataset for Puerto Rico, add it to our Entity, and plot it:

```
[10]: from climada.entity.exposures import LitPop

exp_litpop = LitPop()
exp_litpop.set_country('Puerto Rico', res_arcsec = 120) # We'll go lower resolution than
↳ default to keep it simple
exp_litpop.set_geometry_points() # Set geodataframe geometries from lat lon data

ent.exposures = exp_litpop

exp_litpop.plot_hexbin(pop_name=True, linewidth=4, buffer=0.1)

2021-04-23 11:49:13,230 - climada.entity.exposures.base - INFO - meta set to default
↳ value {}
2021-04-23 11:49:13,231 - climada.entity.exposures.base - INFO - tag set to default
↳ value File:
Description:
2021-04-23 11:49:13,232 - climada.entity.exposures.base - INFO - ref_year set to default
↳ value 2018
2021-04-23 11:49:13,233 - climada.entity.exposures.base - INFO - value_unit set to
↳ default value USD
```

(continues on next page)

(continued from previous page)

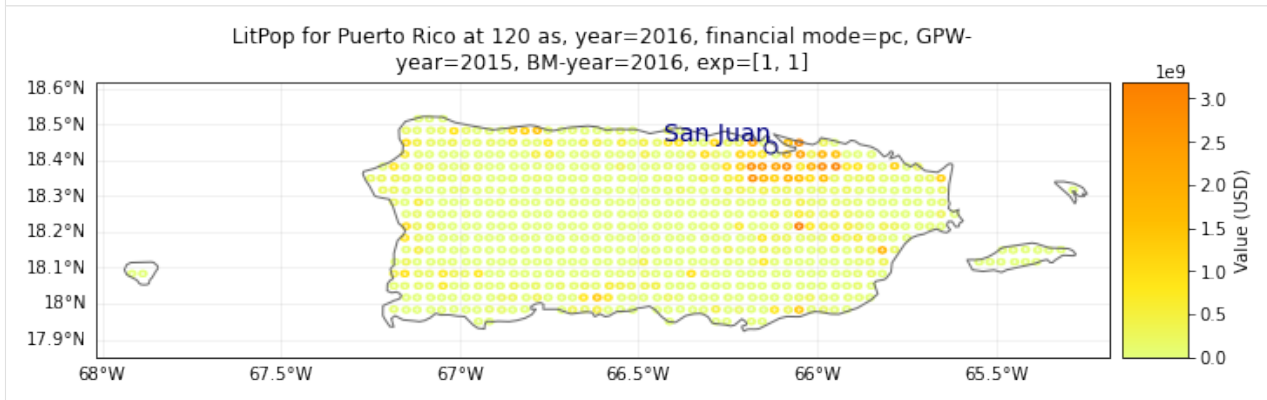
```

2021-04-23 11:49:13,234 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-04-23 11:49:14,717 - climada.entity.exposures.litpop - INFO - Generating LitPop.
↳data at a resolution of 120 arcsec.
2021-04-23 11:49:18,245 - climada.entity.exposures.gpw_import - INFO - Reference year:
↳2016. Using nearest available year for GWP population data: 2015
2021-04-23 11:49:18,246 - climada.entity.exposures.gpw_import - INFO - GPW Version v4.11
2021-04-23 11:49:36,011 - climada.util.finance - WARNING - No data available for country.
↳ Using non-financial wealth instead
2021-04-23 11:49:36,880 - climada.util.finance - INFO - GDP PRI 2016: 1.043e+11.
2021-04-23 11:49:36,884 - climada.util.finance - WARNING - No data for country, using
↳mean factor.
2021-04-23 11:49:37,023 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-23 11:49:37,024 - climada.entity.exposures.base - INFO - tag set to default.
↳value File:
Description:
2021-04-23 11:49:37,024 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-04-23 11:49:37,025 - climada.entity.exposures.base - INFO - value_unit set to
↳default value USD
2021-04-23 11:49:37,026 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-04-23 11:49:37,045 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-23 11:49:37,048 - climada.entity.exposures.litpop - INFO - Creating the LitPop.
↳exposure took 23 s
2021-04-23 11:49:37,048 - climada.entity.exposures.base - INFO - Hazard type not set in
↳impf_
2021-04-23 11:49:37,049 - climada.entity.exposures.base - INFO - category_id not set.
2021-04-23 11:49:37,049 - climada.entity.exposures.base - INFO - cover not set.
2021-04-23 11:49:37,050 - climada.entity.exposures.base - INFO - deductible not set.
2021-04-23 11:49:37,050 - climada.entity.exposures.base - INFO - geometry not set.
2021-04-23 11:49:37,051 - climada.entity.exposures.base - INFO - centr_ not set.
2021-04-23 11:49:37,062 - climada.util.coordinates - INFO - Setting geometry points.

/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now
↳deprecated and will not be supported in the future.
    self.gdf.crs = self.meta['crs']
/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳730: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now
↳deprecated and will not be supported in the future.
    exp.gdf = GeoDataFrame(
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/geopandas/
↳geodataframe.py:167: UserWarning: Pandas doesn't allow columns to be created via a new
↳attribute name - see https://pandas.pydata.org/pandas-docs/stable/indexing.html
↳#attribute-access
    super(GeoDataFrame, self).__setattr__(attr, val)
/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳190: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now
↳deprecated and will not be supported in the future.
    self.gdf = GeoDataFrame(*args, **kwargs)

```

```
[10]: <GeoAxesSubplot:title={'center': 'LitPop for Puerto Rico at 120 as, year=2016, financial_
↪mode=pc, GPW-\nyear=2015, BM-year=2016, exp=[1, 1]'}>
```



LitPop's default exposure is measured in US Dollars, with a reference year depending on the most recent data available.

Once we've created our impact function we will come back to this Exposure and give it the parameters needed to connect exposure to impacts.

Impact functions

Impact functions describe a relationship between a hazard's intensity and your exposure in terms of a percentage loss. The impact is described through two terms. The Mean Degree of Damage (MDD) gives the percentage of an exposed asset's numerical value that's affected as a function of intensity, such as the damage to a building from wind in terms of its total worth. Then the Proportion of Assets Affected (PAA) gives the fraction of exposures that are affected, such as the mortality rate in a population from a heatwave. These multiply to give the Mean Damage Ratio (MDR), the average impact to an asset.

Impact functions are stored as the Entity's `impact_funcs` attribute, in an instance of the `ImpactFuncSet` class which groups one or more `ImpactFunc` objects. They can be specified manually, read from a file, or you can use CLIMADA's pre-defined impact functions. We'll use a pre-defined function for tropical storm wind damage stored in the `IFTropCyclone` class.

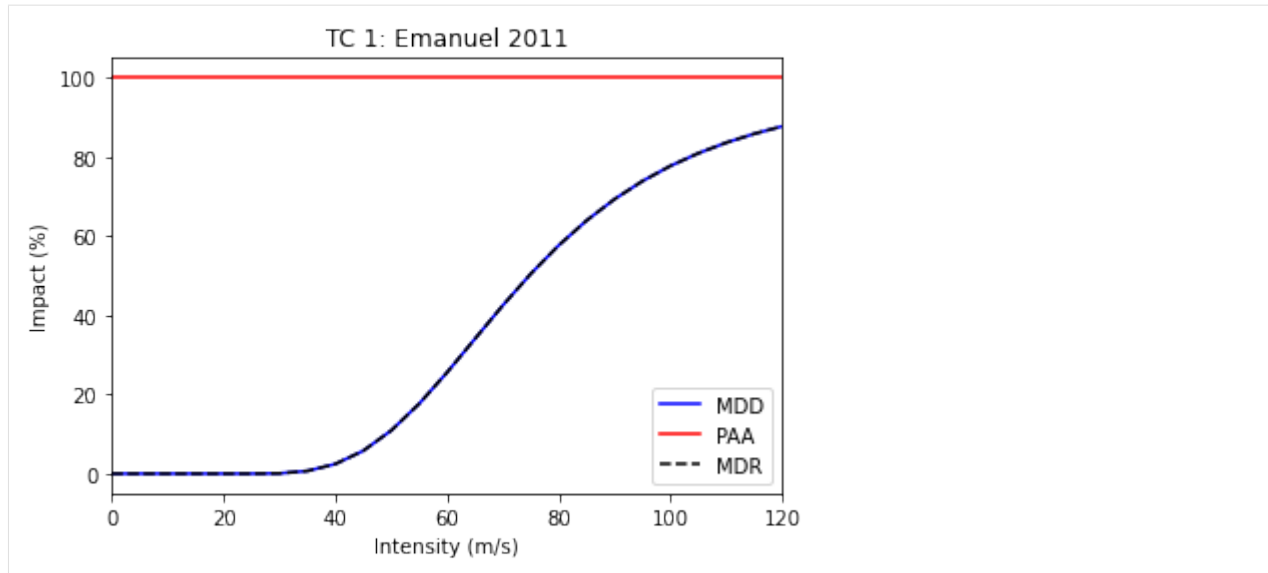
See the [Impact Functions tutorial](#) for a full guide to the class, including how data are stored and reading and writing to files.

We initialise an Impact Function with the `IFTropCyclone` class, and use its `set_emanuel_usa` method to load the Emanuel (2011) impact function. (The class also contains regional impact functions for the full globe, but we'll won't use these for now.) The class's `plot` method visualises the function, which we can see is expressed just through the Mean Degree of Damage, with all assets affected.

```
[13]: from climada.entity.impact_funcs import ImpactFuncSet, ImpfTropCyclone
```

```
imp_fun = ImpfTropCyclone()
imp_fun.set_emanuel_usa()
imp_fun.plot()
```

```
[13]: <AxesSubplot:title={'center': 'TC 1: Emanuel 2011'}, xlabel='Intensity (m/s)', ylabel=
↪'Impact (%)'>
```



The plot title also includes information about the function's ID, which were also set by the `set_emanuel_usa` method. The hazard is "TC" and the function ID is 1. Since a study might use several impact functions - for different hazards, or for different types of exposure.

We then create an `ImpactFuncSet` object to store the impact function. This is a container class, and groups a study's impact functions together. Studies will often have several impact functions, due to multiple hazards, multiple types of exposure that are impacted differently, or different adaptation scenarios. We add it to our Entity object.

```
[14]: imp_fun_set = ImpactFuncSet()
      imp_fun_set.append(imp_fun)

      ent.impact_funcs = imp_fun_set
```

Finally, we can update our LitPop exposure to point to the TC 1 impact function. This is done by adding a column to the exposure:

```
[15]: ent.exposures.gdf['impf_TC'] = 1
      ent.check()
```

```
2021-04-23 11:52:32,801 - climada.entity.exposures.base - INFO - Hazard type not set in_
↳ impf_
2021-04-23 11:52:32,802 - climada.entity.exposures.base - INFO - category_id not set.
2021-04-23 11:52:32,802 - climada.entity.exposures.base - INFO - cover not set.
2021-04-23 11:52:32,803 - climada.entity.exposures.base - INFO - deductible not set.
2021-04-23 11:52:32,804 - climada.entity.exposures.base - INFO - centr_ not set.
2021-04-23 11:52:32,805 - 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.
```

Here the `impf_TC` column tells the CLIMADA engine that for a tropical cyclone (TC) hazard, it should use the first impact function defined for TCs. We use the same impact function for all of our exposure.

This is now everything we need for a risk analysis, but while we're working on the Entity class, we can define the adaptation measures and discount rates needed for an adaptation analysis. If you're not interested in the cost-benefit analysis, you can skip ahead to the [Impact section](#)

Adaptation measures

CLIMADA's adaptation measures describe possible interventions that would change event hazards and impacts, and the cost of these interventions.

They are stored as `Measure` objects within a `MeasureSet` container class (similarly to `ImpactFuncSet` containing several `ImpactFuncs`), and are assigned to the `measures` attribute of the `Entity`.

See the [Adaptation Measures tutorial](#) on how to create, read and write measures. CLIMADA doesn't yet have pre-defined adaptation measures, mostly because they are hard to standardise.

The best way to understand an adaptation measure is by an example. Here's a possible measure for the creation of coastal mangroves (ignore the exact numbers, they are just for illustration):

```
[16]: from climada.entity import Measure, MeasureSet

meas_mangrove = Measure()
meas_mangrove.name = 'Mangrove'
meas_mangrove.haz_type = 'TC'
meas_mangrove.color_rgb = np.array([0.2, 0.2, 0.7])
meas_mangrove.cost = 5000000000
meas_mangrove.mdd_impact = (1, 0)
meas_mangrove.paa_impact = (1, -0.15)
meas_mangrove.hazard_inten_imp = (1, -10)

meas_set = MeasureSet()
meas_set.append(meas_mangrove)
meas_set.check()
```

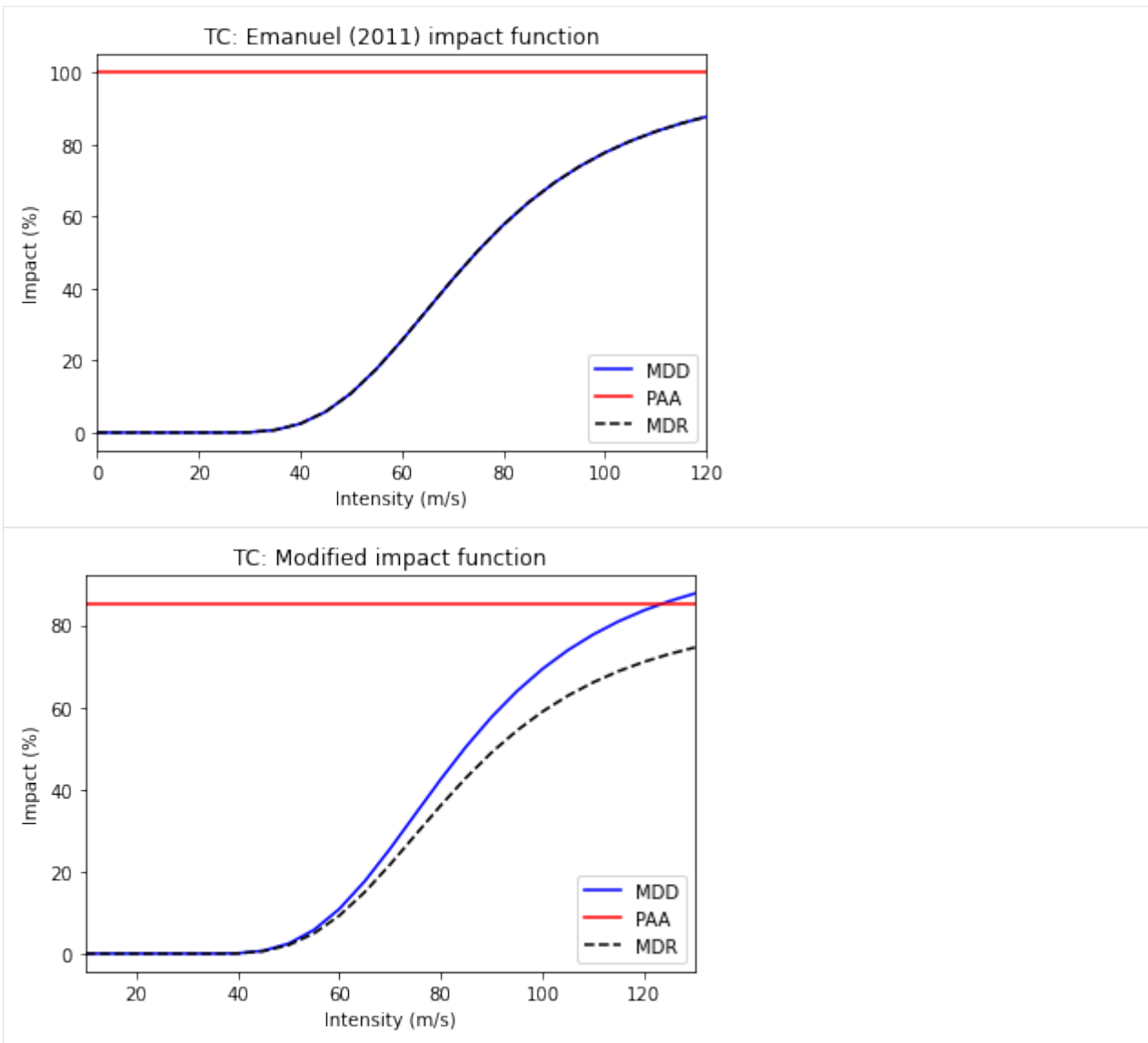
What values have we set here? - The `haz_type` gives the hazard that this measure affects. - The `cost` is a flat price that will be used in cost-benefit analyses. - The `mdd_impact`, `paa_impact`, and `hazard_inten_imp` attributes are all tuples that describes a linear transformation to event hazard, the impact function's mean damage degree and the impact function's proportion of assets affected. The tuple (a, b) describes a scalar multiplication of the function and a constant to add. So (1, 0) is unchanged, (1.1, 0) increases values by 10%, and (1, -10) decreases all values by 10.

So the Mangrove example above costs 50,000,000 USD, protects 15% of assets from any impact at all (`paa_impact = (1, -0.15)`) and decreases the (effective) hazard intensity by 10 m/s (`hazard_inten_imp = (1, -10)`).

We can apply these measures to our existing `Exposure`, `Hazard` and `Impact` functions, and plot the old and new impact functions:

```
[17]: mangrove_exp, mangrove_imp_fun_set, mangrove_haz = meas_mangrove.apply(exp_litpop, imp_
      ↪ fun_set, haz)
axes1 = imp_fun_set.plot()
axes1.set_title('TC: Emanuel (2011) impact function')
axes2 = mangrove_imp_fun_set.plot()
axes2.set_title('TC: Modified impact function')

[17]: Text(0.5, 1.0, 'TC: Modified impact function')
```



Let's define a second measure. Again, the numbers here are made up, for illustration only.

```
[18]: meas_buildings = Measure()
      meas_buildings.name = 'Building code'
      meas_buildings.haz_type = 'TC'
      meas_buildings.color_rgb = np.array([0.2, 0.7, 0.5])
      meas_buildings.cost = 100000000
      meas_buildings.hazard_freq_cutoff = 0.1

      meas_set.append(meas_buildings)
      meas_set.check()

      buildings_exp, buildings_imp_fun_set, buildings_haz = meas_buildings.apply(exp_litpop,
      ↪ imp_fun_set, haz)

      2021-04-23 11:52:38,047 - climada.entity.exposures.base - INFO - Matching 691 exposures,
      ↪ with 1891 centroids.
```

(continues on next page)

(continued from previous page)

```
2021-04-23 11:52:38,050 - climada.engine.impact - INFO - Calculating damage for 682
↳assets (>0) and 543 events.
```

This measure describes an upgrade to building codes to withstand 10-year events. The measure costs 100,000,000 USD and, through `hazard_freq_cutoff = 0.1`, removes events with calculated impacts below the 10-year return period.

The [Adaptation Measures tutorial](#) describes other parameters for describing adaptation measures, including risk transfer, assigning measures to subsets of exposure, and reassigning impact functions.

We can compare the 5- and 20-year return period hazard (remember: not a real return period due to the small event set!) compared to the adjusted hazard once low-impact events are removed.

```
[ ]: haz.plot_rp_intensity(return_periods=(5, 20))
      buildings_haz.plot_rp_intensity(return_periods=(5, 20))
```

It shows there are now very few events at the 5-year return period - the new building codes removed most of these from the event set. Finally we add the measure set to our Entity.

```
[ ]: ent.measures = meas_set
```

Discount rates

The `disc_rates` attribute is of type `DiscRates`. This class contains the discount rates for the following years and computes the net present value for given values.

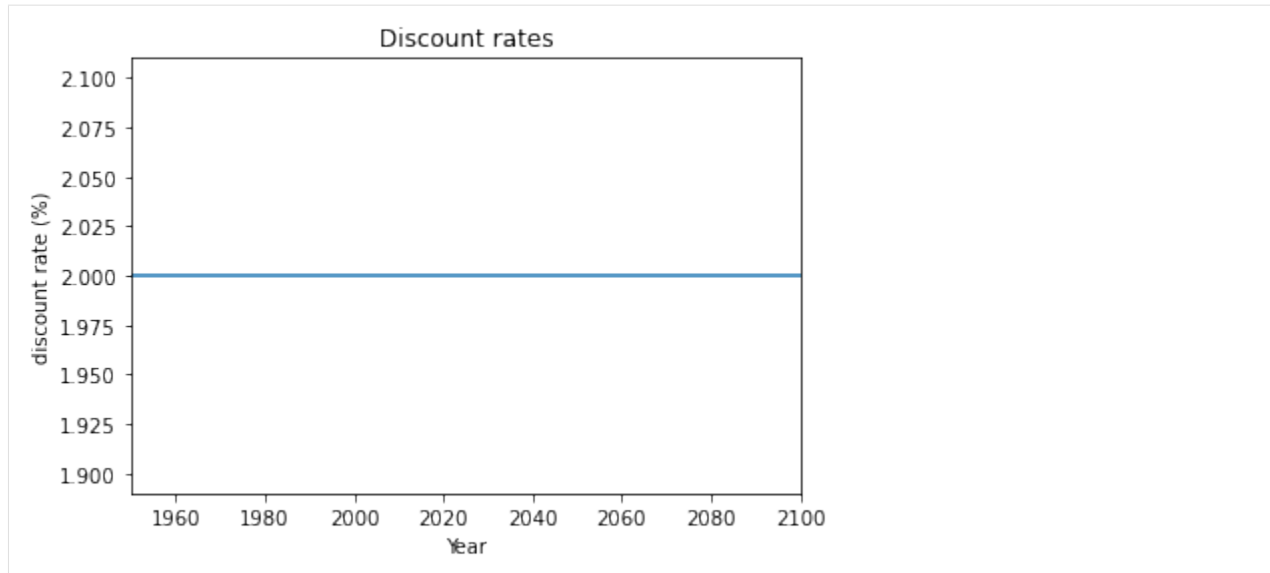
See the [Discount Rates tutorial](#) for more details about creating, reading and writing the `DiscRates` class, and how it is used in calculations.

Here we will implement a simple, flat 2% discount rate.

```
[19]: from climada.entity import DiscRates

disc = DiscRates()
disc.years = np.arange(1950, 2101)
disc.rates = np.ones(disc.years.size) * 0.02
disc.check()
disc.plot()

ent.disc_rates = disc
```



We are now ready to move to the last part of the CLIMADA model for Impact and Cost Benefit analyses.

5.1.7 Engine

The CLIMADA Engine is where the main risk calculations are done. It contains two classes, `Impact`, for risk assessments, and `CostBenefit`, to evaluate adaptation measures.

Impact

Let us compute the impact of historical tropical cyclones in Puerto Rico.

Our work above has given us everything we need for a risk analysis using the `Impact` class. By computing the impact for each historical event, the `Impact` class provides different risk measures, as the expected annual impact per exposure, the probable maximum impact for different return periods and the total average annual impact.

Note: the configurable parameter `MAX_SIZE` controls the maximum matrix size contained in a chunk. You can decrease its value if you are having memory issues when using the `Impact`'s `calc` method. A high value will make the computation fast, but increase the memory use. The configuration file is located at `climada_python/climada/conf/defaults.conf`.

CLIMADA calculates impacts by providing exposures, impact functions and hazard to an `Impact` object's `calc` method:

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

imp = Impact()
imp.calc(ent.exposures, ent.impact_funcs, haz)

2021-04-23 11:52:42,881 - climada.engine.impact - INFO - Exposures matching centroids_
↳ found in centr_TC
2021-04-23 11:52:42,883 - climada.engine.impact - INFO - Calculating damage for 682_
↳ assets (>0) and 543 events.
```

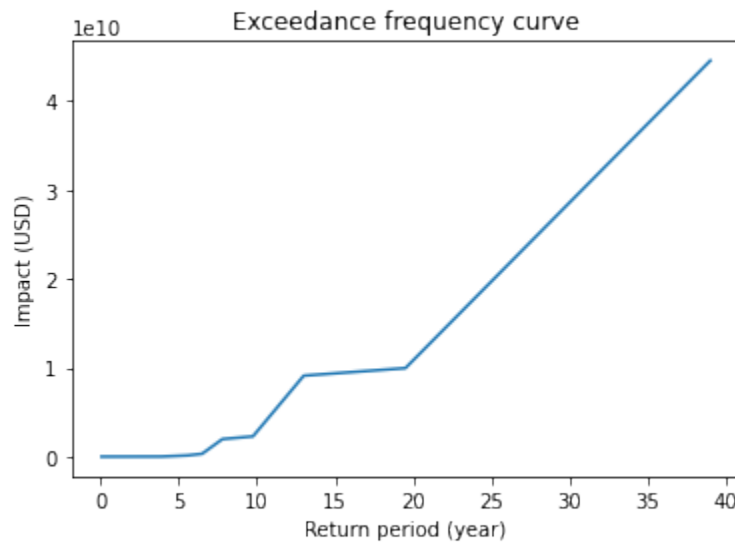
A useful parameter for the `calc` method is `save_mat`. When set to `True` (default is `False`), the `Impact` object saves the calculated impact for each event at each point of exposure, stored as a (large) sparse matrix in the `imp_mat` attribute. This allows for more detailed analysis at the event level.

The Impact class includes a number of analysis tools. We can plot an exceedance frequency curve, showing us how often different damage thresholds are reached in our source data (remember this is only 40 years of storms, so not a full climatology!)

```
[21]: freq_curve = imp.calc_freq_curve() # impact exceedance frequency curve
      freq_curve.plot();

      print('Expected average annual impact: {:.3e} USD'.format(imp.aai_agg))

Expected average annual impact: 1.754e+09 USD
```



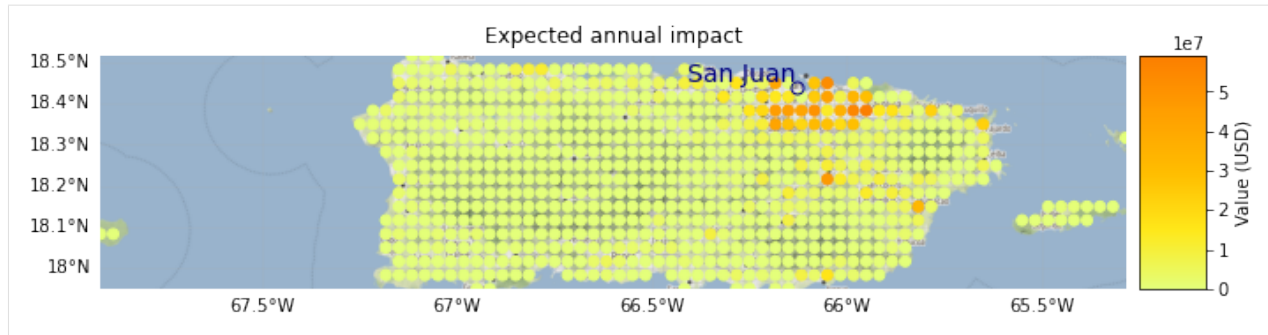
We can map the expected annual impact by exposure:

```
[22]: imp.plot_basemap_eai_exposure(buffer=0.1); # average annual impact at each exposure

2021-04-23 11:52:47,128 - climada.util.coordinates - INFO - Setting geometry points.
2021-04-23 11:52:47,249 - climada.entity.exposures.base - INFO - Setting latitude and
↳ longitude attributes.

/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳ 190: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now
↳ deprecated and will not be supported in the future.
      self.gdf = GeoDataFrame(*args, **kwargs)
/Users/zeliestalhanske/miniconda3/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(

2021-04-23 11:52:50,676 - climada.entity.exposures.base - INFO - Setting latitude and
↳ longitude attributes.
```



For additional functionality, including plotting the impacts of individual events, see the [Impact tutorial](#).

Exercise: Plot the impacts of Hurricane Maria. To do this you'll need to set `save_mat=True` in the earlier `Impact.calc()`.

We can save our variables in pickle format using the `save` function and load them with `load`. This will save your results in the folder specified in the configuration file. The default folder is a `results` folder which is created in the current path (see default configuration file `climada/conf/defaults.conf`). However, we recommend to use CLIMADA's writers in `hdf5` or `csv` whenever possible.

```
[23]: import os
from climada.util import save, load

### Uncomment this to save - saves by default to ./results/
# save('impact_puerto_rico_tc.p', imp)

### Uncomment this to read the saved data:
# abs_path = os.path.join(os.getcwd(), 'results/impact_puerto_rico_tc.p')
# data = load(abs_path)
```

`Impact` also has `write_csv()` and `write_excel()` methods to save the impact variables, and `write_sparse_csr()` to save the impact matrix (impact per event and exposure). Use the [Impact tutorial](#) to get more information about these functions and the class in general.

Adaptation options appraisal

Finally, let's look at a cost-benefit analysis. The adaptation measures defined with our `Entity` can be valued by estimating their cost-benefit ratio. This is done in the class `CostBenefit`.

Let us suppose that the socioeconomic and climatological conditions remain the same in 2040. We then compute the cost and benefit of every adaptation measure from our `Hazard` and `Entity` (and plot them) as follows:

```
[ ]: from climada.engine import CostBenefit

cost_ben = CostBenefit()
cost_ben.calc(haz, ent, future_year=2040) # prints costs and benefits
cost_ben.plot_cost_benefit() # plot cost benefit ratio and averted damage of every_
    ↪ exposure
cost_ben.plot_event_view(return_per=(10, 20, 40)) # plot averted damage of each measure_
    ↪ for every return period
```

This is just the start. Analyses improve as we add more adaptation measures into the mix.

Cost-benefit calculations can also include - climate change, by specifying the `haz_future` parameter in `CostBenefit.calc()` - changes to economic exposure over time (or to whatever exposure you're modelling) by specifying the `ent_future` parameter in `CostBenefit.calc()` - different functions to calculate risk benefits. These are specified in `CostBenefit.calc()` and by default use changes to average annual impact - linear, sublinear and superlinear evolution of impacts between the present and future, specified in the `imp_time_depen` parameter in `CostBenefit.calc()`

And once future hazards and exposures are defined, we can express changes to impacts over time as waterfall diagrams. See the `CostBenefit` class for more details.

Exercise: repeat the above analysis, creating future climate hazards (see the first exercise), and future exposures based on projected economic growth. Visualise it with the `CostBenefit.plot_waterfall()` method.

5.1.8 What next?

Thanks for following this tutorial! Take time to work on the exercises it suggested, or design your own risk analysis for your own topic. More detailed tutorials for individual classes were listed in the [Features](#) section.

Also, explore the full CLIMADA documentation and additional resources [described at the start of this document](#) to learn more about CLIMADA, its structure, its existing applications and how you can contribute.

5.2 Exposures class

5.2.1 What is an exposure?

Exposure describes the set of assets, people, livelihoods, infrastructures, etc. within an area of interest in terms of their geographic location, their value etc.; in brief - everything potentially exposed to hazards.

5.2.2 What options does CLIMADA offer for me to create an exposure?

CLIMADA has an `Exposures` class for this purpose. An `Exposures` instance can be filled with your own data, or loaded from available default sources implemented through some `Exposures`-type classes from CLIMADA. If you have your own data, they can be provided in the formats of a `pandas.DataFrame`, a `geopandas.GeoDataFrame` or simply an Excel file. If you didn't collect your own data, exposures can be generated on the fly using CLIMADA's [LitPop](#), [BlackMarble](#) or [OpenStreetMap](#) modules. See the respective tutorials to learn what exactly they contain and how to use them.

5.2.3 What does an exposure look like in CLIMADA?

An exposure is represented in the class `Exposures`, which contains a `geopandas.GeoDataFrame` that is accessible through the `Exposures.gdf` attribute. Certain columns of `gdf` *have to* be specified, while others are optional (this means that the package `climada.engine` also works without these variables set.) The full list looks like this:

Mandatory columns	Data Type	Description
latitude	float	latitude
longitude	float	longitude
value	float	a value for each exposure

Op- tional columns	Data Type	Description
impf_*	int	impact functions ids for hazard types.important attribute, since it relates the exposures to the hazard by specifying the impf_act functions.Ideally it should be set to the specific hazard (e.g. impf_TC) so that different hazards can be set in the same Exposures (e.g. impf_TC and impf_FL).If not provided, set to default impf_ with ids 1 in check().
ge- ome- try	Point	geometry of type PointMain feature of geopandas DataFrame extensionComputed in method set_geometry_points()
de- ductible	float	deductible value for each exposure. Used for insurance
cover	float	cover value for each exposure. Used for insurance
cate- gory_id	int	category id (e.g. building code) for each exposure
re- gion_id	int	region id (e.g. country ISO code) for each exposure
centr_*	int	centroids index for hazard type.There might be different hazards defined: centr_TC, centr_FL, ...Computed in method assign_centroids()

Meta- data vari- ables	Data Type	Description
crs	str or int	coordinate reference system, see GeoDataFrame.crs
tag	Tag	information about the source data
ref_year	int	reference year
value_unit	str	unit of the exposures' values
meta	dict	dictionary containing corresponding raster properties (if any):width, height, crs and transform must be present at least (transform needs to contain upper left corner!).Exposures might not contain all the points of the corresponding raster.

How is this tutorial structured?

****Part 1:**** Defining exposures from your own data (DataFrame, GeoDataFrame, Excel)

****Part 2:**** Loading exposures from CLIMADA-files or generating new ones (LitPop, BlackMarble, OSM)

****Part 3:**** Visualizing exposures

****Part 4:**** Writing (=saving) exposures

****Part 5:**** What to do with large exposure data

Part 1: Defining exposures from your own data The essential structure of an exposure is similar, irrespective of the data type you choose to provide: As mentioned in the introduction, the key variables to be provided are `latitudes`, `longitudes` and `values` of your exposed assets. While not mandatory, but very useful to provide for the impact calculation at later stages: the impact function id (see `impf_*` in the table above). The following examples will walk you through how

to specify those four variables, and demonstrate the use of a few more optional parameters on the go.

Exposures from a pandas DataFrame

In case you are unfamiliar with the data structure, check out the [pandas DataFrame documentation](#).

```
[1]: import numpy as np
from pandas import DataFrame
from climada.entity import Exposures

# Fill a pandas DataFrame with the 3 mandatory variables (latitude, longitude, value)
# for a number of assets (10'000).
# We will do this with random dummy data for purely illustrative reasons:
exp_df = DataFrame()
n_exp = 100*100
# provide value
exp_df['value'] = np.arange(n_exp)
# provide latitude and longitude
lat, lon = np.mgrid[15 : 35 : complex(0, np.sqrt(n_exp)), 20 : 40 : complex(0, np.sqrt(n_exp))]
exp_df['latitude'] = lat.flatten()
exp_df['longitude'] = lon.flatten()
```

```
[2]: # For each exposure entry, specify which impact function should be taken for which
# hazard type.
# In this case, we only specify the IDs for tropical cyclone (TC); here, each exposure
# entry will be treated with
# the same impact function: the one that has ID '1':
# Of course, this will only be relevant at later steps during impact calculations.
exp_df['impf_TC'] = np.ones(n_exp, int)
```

```
[3]: # Let's have a look at the pandas DataFrame
print('\x1b[1;03;30;30m' + 'exp_df is a DataFrame:', str(type(exp_df)) + '\x1b[0m')
print('\x1b[1;03;30;30m' + 'exp_df looks like:' + '\x1b[0m')
print(exp_df.head())
```

```
exp_df is a DataFrame: <class 'pandas.core.frame.DataFrame'>
exp_df looks like:
```

	value	latitude	longitude	impf_TC
0	0	15.0	20.000000	1
1	1	15.0	20.202020	1
2	2	15.0	20.404040	1
3	3	15.0	20.606061	1
4	4	15.0	20.808081	1

```
[4]: # Generate Exposures from the pandas DataFrame. This step converts the DataFrame into
# a CLIMADA Exposures instance!
exp = Exposures(exp_df)
print('\n\x1b[1;03;30;30m' + 'exp has the type:', str(type(exp)))
print('and contains a GeoDataFrame exp.gdf:', str(type(exp.gdf)) + '\n\n\x1b[0m')

# set geometry attribute (shapely Points) from GeoDataFrame from latitude and longitude
exp.set_geometry_points()
print('\n' + '\x1b[1;03;30;30m' + 'check method logs:' + '\x1b[0m')
```

(continues on next page)

(continued from previous page)

```
# always apply the check() method in the end. It puts metadata that has not been
↳ assigned,
# and points out missing mandatory data
exp.check()

2021-06-04 17:07:21,752 - climada.entity.exposures.base - INFO - meta set to default
↳ value {}
2021-06-04 17:07:21,753 - climada.entity.exposures.base - INFO - tag set to default
↳ value File:
Description:
2021-06-04 17:07:21,754 - climada.entity.exposures.base - INFO - ref_year set to default
↳ value 2018
2021-06-04 17:07:21,754 - climada.entity.exposures.base - INFO - value_unit set to
↳ default value USD
2021-06-04 17:07:21,755 - climada.entity.exposures.base - INFO - crs set to default
↳ value: EPSG:4326
```

```
exp has the type: <class 'climada.entity.exposures.base.Exposures'>
and contains a GeoDataFrame exp.gdf: <class 'geopandas.geodataframe.GeoDataFrame'>
```

```
2021-06-04 17:07:21,773 - climada.util.coordinates - INFO - Setting geometry points.
/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳ 221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now
↳ deprecated and will not be supported in the future.
self.gdf.crs = self.meta['crs']
```

check method logs:

```
2021-06-04 17:07:22,063 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-04 17:07:22,064 - climada.entity.exposures.base - INFO - cover not set.
2021-06-04 17:07:22,064 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-04 17:07:22,065 - climada.entity.exposures.base - INFO - region_id not set.
2021-06-04 17:07:22,065 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
[5]: # let's have a look at the Exposures instance we created!
print('\n' + '\x1b[1;03;30;30m' + 'exp looks like:' + '\x1b[0m')
print(exp)
```

exp looks like:

tag: File:

Description:

ref_year: 2018

value_unit: USD

meta: {'crs': 'EPSG:4326'}

crs: EPSG:4326

data:

	value	latitude	longitude	impf_TC	geometry
0	0	15.0	20.000000	1	POINT (20.00000 15.00000)
1	1	15.0	20.202020	1	POINT (20.20202 15.00000)
2	2	15.0	20.404040	1	POINT (20.40404 15.00000)
3	3	15.0	20.606061	1	POINT (20.60606 15.00000)
4	4	15.0	20.808081	1	POINT (20.80808 15.00000)

(continues on next page)

(continued from previous page)

```

...      ...      ...      ...      ...
9995    9995      35.0  39.191919      1  POINT (39.19192 35.000000)
9996    9996      35.0  39.393939      1  POINT (39.39394 35.000000)
9997    9997      35.0  39.595960      1  POINT (39.59596 35.000000)
9998    9998      35.0  39.797980      1  POINT (39.79798 35.000000)
9999    9999      35.0  40.000000      1  POINT (40.00000 35.000000)

[10000 rows x 5 columns]

```

Exposures from a geopandas GeoDataFrame

In case you are unfamiliar with with data structure, check out the [geopandas GeoDataFrame documentation](#). The main difference to the example above (pandas DataFrame) is that, while previously, we provided latitudes and longitudes which were then converted to a geometry GeoSeries using the `set_geometry_points` method, GeoDataFrames already come with a defined geometry GeoSeries. In this case, we take the geometry info and use the `set_lat_lon` method to explicitly provide latitudes and longitudes. This example focuses on data with POINT geometry, but in principle, other geometry types (such as POLYGON and MULTIPOLYGON) would work as well.

```

[6]: import numpy as np
import geopandas as gpd
from climada.entity import Exposures

# Read spatial info from an external file into GeoDataFrame
world = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
print('\x1b[1;03;30;30m' + 'World is a GeoDataFrame:', str(type(world)) + '\x1b[0m')
print('\x1b[1;03;30;30m' + 'World looks like:' + '\x1b[0m')
print(world.head())

World is a GeoDataFrame: <class 'geopandas.geodataframe.GeoDataFrame'>
World looks like:
   name      geometry
0  Vatican City  POINT (12.45339 41.90328)
1   San Marino  POINT (12.44177 43.93610)
2   Vaduz      POINT (9.51667 47.13372)
3  Luxembourg  POINT (6.13000 49.61166)
4   Palikir    POINT (158.14997 6.91664)

```

```

[7]: # Generate Exposures: value, latitude and longitude for each exposure entry.
# Convert GeoDataFrame into Exposure instance
exp_gpd = Exposures(world)
print('\n' + '\x1b[1;03;30;30m' + 'exp_gpd is an Exposures:', str(type(exp_gpd)) + '\x1b[0m')
# add random values to entries
exp_gpd.gdf['value'] = np.arange(world.shape[0])
# set latitude and longitude attributes from geometry
exp_gpd.set_lat_lon()

2021-06-04 17:07:22,164 - climada.entity.exposures.base - INFO - meta set to default_
↳ value {}
2021-06-04 17:07:22,165 - climada.entity.exposures.base - INFO - tag set to default_
↳ value File:
Description:

```

(continues on next page)

(continued from previous page)

```

2021-06-04 17:07:22,165 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-06-04 17:07:22,166 - climada.entity.exposures.base - INFO - value_unit set to
↳default value USD

exp_gpd is an Exposures: <class 'climada.entity.exposures.base.Exposures'>
2021-06-04 17:07:22,169 - climada.entity.exposures.base - INFO - Setting latitude and
↳longitude attributes.

```

```

[8]: # For each exposure entry, specify which impact function should be taken for which
↳hazard type.
# In this case, we only specify the IDs for tropical cyclone (TC); here, each exposure
↳entry will be treated with
# the same impact function: the one that has ID '1':
# Of course, this will only be relevant at later steps during impact calculations.
exp_gpd.gdf['impf_TC'] = np.ones(world.shape[0], int)
print('\n' + '\x1b[1;03;30;30m' + 'check method logs:' + '\x1b[0m')

# as always, run check method to assign meta-data and check for missing mandatory
↳variables.
exp_gpd.check()

```

check method logs:

```

2021-06-04 17:07:22,193 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-04 17:07:22,194 - climada.entity.exposures.base - INFO - cover not set.
2021-06-04 17:07:22,195 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-04 17:07:22,196 - climada.entity.exposures.base - INFO - region_id not set.
2021-06-04 17:07:22,197 - climada.entity.exposures.base - INFO - centr_ not set.

```

```

[9]: # let's have a look at the Exposures instance we created!
print('\n' + '\x1b[1;03;30;30m' + 'exp_gpd looks like:' + '\x1b[0m')
print(exp_gpd)

```

exp_gpd looks like:

```

tag: File:
Description:
ref_year: 2018
value_unit: USD
meta: {'crs': <Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
}

```

(continues on next page)

(continued from previous page)

crs: epsg:4326

data:

	name	geometry	value	latitude	longitude	\
0	Vatican City	POINT (12.45339 41.90328)	0	41.903282	12.453387	
1	San Marino	POINT (12.44177 43.93610)	1	43.936096	12.441770	
2	Vaduz	POINT (9.51667 47.13372)	2	47.133724	9.516669	
3	Luxembourg	POINT (6.13000 49.61166)	3	49.611660	6.130003	
4	Palikir	POINT (158.14997 6.91664)	4	6.916644	158.149974	
..	
197	Cairo	POINT (31.24802 30.05191)	197	30.051906	31.248022	
198	Tokyo	POINT (139.74946 35.68696)	198	35.686963	139.749462	
199	Paris	POINT (2.33139 48.86864)	199	48.868639	2.331389	
200	Santiago	POINT (-70.66899 -33.44807)	200	-33.448068	-70.668987	
201	Singapore	POINT (103.85387 1.29498)	201	1.294979	103.853875	

	impf_TC
0	1
1	1
2	1
3	1
4	1
..	...
197	1
198	1
199	1
200	1
201	1

[202 rows x 6 columns]

The fact that Exposures is built around a `geopandas.GeoDataFrame` offers all the useful functionalities that come with the package. The following examples showcase only a few of those.

```
[10]: # Example 1: extract data in a region: latitudes between -5 and 5
sel_exp = exp_gpd.copy() # to keep the original exp_gpd Exposures data
sel_exp.gdf = sel_exp.gdf.cx[:, -5:5]

print('\n' + '\x1b[1;30;30m' + 'sel_exp contains a subset of the original data' + '\n'
      + '\x1b[0m')
sel_exp.gdf.head()
```

sel_exp contains a subset of the original data

	name	geometry	value	latitude	longitude	impf_TC
9	Tarawa	POINT (173.01757 1.33819)	9	1.338188	173.017571	1
13	Kigali	POINT (30.05859 -1.95164)	13	-1.951644	30.058586	1
15	Juba	POINT (31.58003 4.82998)	15	4.829975	31.580026	1
27	Bujumbura	POINT (29.36001 -3.37609)	27	-3.376087	29.360006	1
48	Kampala	POINT (32.58138 0.31860)	48	0.318605	32.581378	1

```
[11]: # Example 2: extract data in a polygon
from shapely.geometry import Polygon
```

(continues on next page)

(continued from previous page)

```

sel_polygon = exp_gpd.copy() # to keep the original exp_gpd Exposures data

poly = Polygon([(0, -10), (0, 10), (10, 5)])
sel_polygon.gdf = sel_polygon.gdf[sel_polygon.gdf.intersects(poly)]

# Let's have a look. Again, the sub-selection is a GeoDataFrame!
print('\n' + '\x1b[1;03;30;30m' + 'sel_exp contains a subset of the original data' + '\n'
      + '\x1b[0m')
sel_polygon.gdf

```

sel_exp contains a subset of the original data

```

[11]:
      name      geometry  value  latitude  longitude  impf_TC
36    Lome  POINT (1.22081 6.13388)    36  6.133883   1.220811         1
84   Malabo  POINT (8.78328 3.75002)    84  3.750015   8.783278         1
113  Cotonou  POINT (2.51804 6.40195)   113  6.401954   2.518045         1
125  Sao Tome  POINT (6.73333 0.33340)   125  0.333402   6.733325         1

```

```

[12]: # Example 3: change coordinate reference system
# use help to see more options: help(sel_exp.to_crs)
sel_polygon.to_crs(epsg=3395, inplace=True)
print('\n' + '\x1b[1;03;30;30m' + 'the crs has changed to ' + str(sel_polygon.crs))
print('the values for latitude and longitude are now according to the new coordinate_
      + '\x1b[0m')
sel_polygon.gdf

```

2021-06-04 17:07:22,376 - climada.entity.exposures.base - INFO - Setting latitude and_
 + '\x1b[0m' longitude attributes.

the crs has changed to epsg:3395

the values for latitude and longitude are now according to the new coordinate system:

```

[12]:
      name      geometry  value  latitude \
36    Lome  POINT (135900.088 679566.334)    36  679566.333952
84   Malabo  POINT (977749.984 414955.551)    84  414955.550857
113  Cotonou  POINT (280307.458 709388.810)   113  709388.810160
125  Sao Tome  POINT (749550.327 36865.909)   125  36865.908682

      longitude  impf_TC
36  135900.087901         1
84  977749.983897         1
113 280307.458315         1
125 749550.327404         1

```

```

[13]: # Example 4: concatenate exposures
exp_all = Exposures.concat([sel_polygon, sel_exp.to_crs(epsg=3395)])

# the output is of type Exposures
print('exp_all type and number of rows:', type(exp_all), exp_all.gdf.shape[0])
print('number of unique rows:', exp_all.gdf.drop_duplicates().shape[0])

# NaNs will appear in the missing values
exp_all.gdf.head()

```

```
2021-06-04 17:07:22,474 - climada.entity.exposures.base - INFO - Setting latitude and
↳ longitude attributes.
exp_all type and number of rows: <class 'climada.entity.exposures.base.Exposures'> 25
number of unique rows: 23
```

```
[13]:
```

	name	geometry	value	latitude	\
0	Lome	POINT (135900.088 679566.334)	36	679566.333952	
1	Malabo	POINT (977749.984 414955.551)	84	414955.550857	
2	Cotonou	POINT (280307.458 709388.810)	113	709388.810160	
3	Sao Tome	POINT (749550.327 36865.909)	125	36865.908682	
4	Tarawa	POINT (19260227.883 147982.749)	9	147982.748978	

	longitude	impf_TC
0	1.359001e+05	1
1	9.777500e+05	1
2	2.803075e+05	1
3	7.495503e+05	1
4	1.926023e+07	1

Exposures of any file type supported by Geopandas and Pandas

Geopandas can read almost any vector-based spatial data format including ESRI shapefile, GeoJSON files and more, see [readers geopandas](#). Pandas supports formats such as csv, html or sql; see [readers pandas](#). Using the corresponding readers, DataFrame and GeoDataFrame can be filled and provided to Exposures following the previous examples.

Exposures from an excel file

If you manually collect exposure data, Excel may be your preferred option. In this case, it is easiest if you format your data according to the structure provided in the template `climada_python/data/system/entity_template.xlsx`, in the sheet `assets`.

```
[14]: import pandas as pd
from climada.util.constants import ENT_TEMPLATE_XLS
from climada.entity import Exposures

# Read your Excel file into a pandas DataFrame (we will use the template example for
↳ this demonstration):
file_name = ENT_TEMPLATE_XLS
exp_tmpl = pd.read_excel(file_name)

# Let's have a look at the data:
print('\x1b[1;03;30;30m' + 'exp_tmpl is a DataFrame:', str(type(exp_tmpl)) + '\x1b[0m')
print('\x1b[1;03;30;30m' + 'exp_tmpl looks like:' + '\x1b[0m')
exp_tmpl.head()

exp_tmpl is a DataFrame: <class 'pandas.core.frame.DataFrame'>
exp_tmpl looks like:
```

```
[14]:
```

	latitude	longitude	value	deductible	cover	region_id	\
0	26.933899	-80.128799	1.392750e+10	0	1.392750e+10	1	
1	26.957203	-80.098284	1.259606e+10	0	1.259606e+10	1	
2	26.783846	-80.748947	1.259606e+10	0	1.259606e+10	1	

(continues on next page)

(continued from previous page)

3	26.645524	-80.550704	1.259606e+10	0	1.259606e+10	1
4	26.897796	-80.596929	1.259606e+10	0	1.259606e+10	1
	category_id	impf_TC	centr_TC	impf_FL	centr_FL	
0	1	1	1	1	1	
1	1	1	2	1	2	
2	1	1	3	1	3	
3	1	1	4	1	4	
4	1	1	5	1	5	

As we can see, the general structure is the same as always: the exposure has latitude, longitude and value columns. Further, this example specified several impact function ids: some for Tropical Cyclones (impf_TC), and some for Floods (impf_FL). It also provides some meta-info (region_id, category_id) and insurance info relevant to the impact calculation in later steps (cover, deductible).

```
[15]: # Generate an Exposures instance from the dataframe.
exp_tmpl = Exposures(exp_tmpl)
print('\n' + '\x1b[1;03;30;30m' + 'exp_tmpl is now an Exposures:', str(type(exp_tmpl)))
↪+ '\x1b[0m')

# set geometry attribute (shapely Points) from GeoDataFrame from latitude and longitude
print('\n' + '\x1b[1;03;30;30m' + 'set_geometry logs:' + '\x1b[0m')
exp_tmpl.set_geometry_points()
# as always, run check method to include metadata and check for missing mandatory
↪parameters

print('\n' + '\x1b[1;03;30;30m' + 'check exp_tmpl:' + '\x1b[0m')
exp_tmpl.check()

2021-06-04 17:07:22,936 - climada.entity.exposures.base - INFO - meta set to default
↪value {}
2021-06-04 17:07:22,937 - climada.entity.exposures.base - INFO - tag set to default
↪value File:
Description:
2021-06-04 17:07:22,938 - climada.entity.exposures.base - INFO - ref_year set to default
↪value 2018
2021-06-04 17:07:22,939 - climada.entity.exposures.base - INFO - value_unit set to
↪default value USD
2021-06-04 17:07:22,940 - climada.entity.exposures.base - INFO - crs set to default
↪value: EPSG:4326

exp_tmpl is now an Exposures: <class 'climada.entity.exposures.base.Exposures'>

set_geometry logs:
2021-06-04 17:07:22,958 - climada.util.coordinates - INFO - Setting geometry points.

check exp_tmpl:

/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↪221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now
↪deprecated and will not be supported in the future.
self.gdf.crs = self.meta['crs']
```

```
[16]: # Let's have a look at our Exposures instance!
print('\n' + '\x1b[1;03;30;30m' + 'exp_templ.gdf looks like:' + '\x1b[0m')
exp_templ.gdf.head()
```

exp_templ.gdf looks like:

```
[16]:
```

	latitude	longitude	value	deductible	cover	region_id	\
0	26.933899	-80.128799	1.392750e+10	0	1.392750e+10	1	
1	26.957203	-80.098284	1.259606e+10	0	1.259606e+10	1	
2	26.783846	-80.748947	1.259606e+10	0	1.259606e+10	1	
3	26.645524	-80.550704	1.259606e+10	0	1.259606e+10	1	
4	26.897796	-80.596929	1.259606e+10	0	1.259606e+10	1	

	category_id	impf_TC	centr_TC	impf_FL	centr_FL	\
0	1	1	1	1	1	
1	1	1	2	1	2	
2	1	1	3	1	3	
3	1	1	4	1	4	
4	1	1	5	1	5	

	geometry
0	POINT (-80.12880 26.93390)
1	POINT (-80.09828 26.95720)
2	POINT (-80.74895 26.78385)
3	POINT (-80.55070 26.64552)
4	POINT (-80.59693 26.89780)

Exposures from a raster file

Last but not least, you may have your exposure data stored in a raster file. Raster data may be read in from any file-type supported by `rasterio`.

```
[17]: from rasterio.windows import Window
from climada.util.constants import HAZ_DEMO_FL

# We take an example with a dummy raster file (HAZ_DEMO_FL), running the method set_from_
# raster directly loads the
# necessary info from the file into an Exposures instance.
exp_raster = Exposures()
exp_raster.set_from_raster(HAZ_DEMO_FL, window= Window(10, 20, 50, 60))
# There are several keyword argument options that come with the set_from_raster method
# (such as
# specifying a window, if not the entire file should be read, or a bounding box. Check
# them out.

2021-06-04 17:07:23,003 - climada.entity.exposures.base - INFO - meta set to default
# value {}
2021-06-04 17:07:23,005 - climada.entity.exposures.base - INFO - tag set to default
# value File:
Description:
2021-06-04 17:07:23,006 - climada.entity.exposures.base - INFO - ref_year set to default
# value 2018
```

(continues on next page)

(continued from previous page)

```

2021-06-04 17:07:23,008 - climada.entity.exposures.base - INFO - value_unit set to_
↳ default value USD
2021-06-04 17:07:23,010 - climada.entity.exposures.base - INFO - crs set to default_
↳ value: EPSG:4326
2021-06-04 17:07:23,026 - climada.util.coordinates - INFO - Reading /Users/
↳ zeliestalhanske/climada/demo/data/SC22000_VE__M1.grd.gz

/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳ 221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now_
↳ deprecated and will not be supported in the future.
    self.gdf.crs = self.meta['crs']
/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳ 411: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now_
↳ deprecated and will not be supported in the future.
    self.gdf.crs = meta['crs'].to_dict()
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be_
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
    return _prepare_from_string(" ".join(pjargs))

```

```

[18]: # As always, run the check method, such that metadata can be assigned and checked for_
↳ missing mandatory parameters.

```

```

exp_raster.check()
print('Meta:', exp_raster.meta)

```

```

2021-06-04 17:07:23,112 - climada.entity.exposures.base - INFO - Setting impf_ to_
↳ default impact functions ids 1.
2021-06-04 17:07:23,116 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-04 17:07:23,119 - climada.entity.exposures.base - INFO - cover not set.
2021-06-04 17:07:23,120 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-04 17:07:23,121 - climada.entity.exposures.base - INFO - geometry not set.
2021-06-04 17:07:23,122 - climada.entity.exposures.base - INFO - region_id not set.
2021-06-04 17:07:23,122 - climada.entity.exposures.base - INFO - centr_ not set.
Meta: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.7014100009187828e+38, 'width':_
↳ 50, 'height': 60, 'count': 1, 'crs': CRS.from_epsg(4326), 'transform': Affine(0.
↳ 0090000000000000341, 0.0, -69.2471495969998,
    0.0, -0.0090000000000000341, 10.248220966978932)}

```

```

[19]: # Let's have a look at the Exposures instance!
print('\n' + '\x1b[1;03;30;30m' + 'exp_raster looks like:' + '\x1b[0m')
exp_raster.gdf.head()

```

exp_raster looks like:

```

[19]:
  longitude  latitude  value  impf_
0   -69.24265   10.243721    0.0     1
1   -69.23365   10.243721    0.0     1
2   -69.22465   10.243721    0.0     1
3   -69.21565   10.243721    0.0     1
4   -69.20665   10.243721    0.0     1

```

Part 2: Loading CLIMADA-generated exposure files or generating new ones

In case you already have a CLIMADA-generated file containing Exposures info, you can of course load it back into memory. Most likely, the data format will either be of `.hdf5` or of `.mat`. In case you neither have your own data, nor a CLIMADA-generated file, you can also create an exposure on the fly using one of the three CLIMADA-internal exposure generators: *LitPop*, *BlackMarble* or *OpenStreetMap* modules. The latter three are extensively described in their own, linked, tutorials.

```
[20]: # read generated with the Python version with read_hdf5()
# note: for .mat data, use the method read_mat() analogously.
from climada.util.constants import EXP_DEMO_H5

exp_hdf5 = Exposures()
exp_hdf5.read_hdf5(EXP_DEMO_H5)
exp_hdf5.check()
print(type(exp_hdf5))

2021-06-04 17:07:23,158 - climada.entity.exposures.base - INFO - meta set to default_
↳value {}
2021-06-04 17:07:23,160 - climada.entity.exposures.base - INFO - tag set to default_
↳value File:
Description:
2021-06-04 17:07:23,162 - climada.entity.exposures.base - INFO - ref_year set to default_
↳value 2018
2021-06-04 17:07:23,163 - climada.entity.exposures.base - INFO - value_unit set to_
↳default value USD
2021-06-04 17:07:23,165 - climada.entity.exposures.base - INFO - crs set to default_
↳value: EPSG:4326
2021-06-04 17:07:23,186 - climada.entity.exposures.base - INFO - Reading /Users/
↳zeliestalhanske/climada/demo/data/exp_demo_today.h5

/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now_
↳deprecated and will not be supported in the future.
self.gdf.crs = self.meta['crs']

2021-06-04 17:07:23,277 - climada.entity.exposures.base - INFO - meta set to default_
↳value {}
2021-06-04 17:07:23,278 - climada.entity.exposures.base - INFO - tag set to default_
↳value File:
Description:
2021-06-04 17:07:23,279 - climada.entity.exposures.base - INFO - ref_year set to default_
↳value 2018
2021-06-04 17:07:23,280 - climada.entity.exposures.base - INFO - value_unit set to_
↳default value USD
2021-06-04 17:07:23,287 - climada.entity.exposures.base - INFO - crs set to default_
↳value: EPSG:4326
2021-06-04 17:07:23,307 - climada.entity.exposures.base - INFO - centr_ not set.
<class 'climada.entity.exposures.base.Exposures'>
```

Before you leave ...

After defining an `Exposures` instance use always the `check()` method to see which attributes are missing. This method will raise an `ERROR` if `value`, `longitude` or `latitude` are missing and an `INFO` messages for the optional variables not set.

Part 3: Visualize Exposures The method `plot_hexbin()` uses `cartopy` and `matplotlib`'s `hexbin function` to represent the exposures values as 2d bins over a map. Configure your plot by fixing the different inputs of the method or by modifying the returned `matplotlib` figure and axes.

The method `plot_scatter()` uses `cartopy` and `matplotlib`'s `scatter function` to represent the points values over a 2d map. As usual, it returns the figure and axes, which can be modified afterwards.

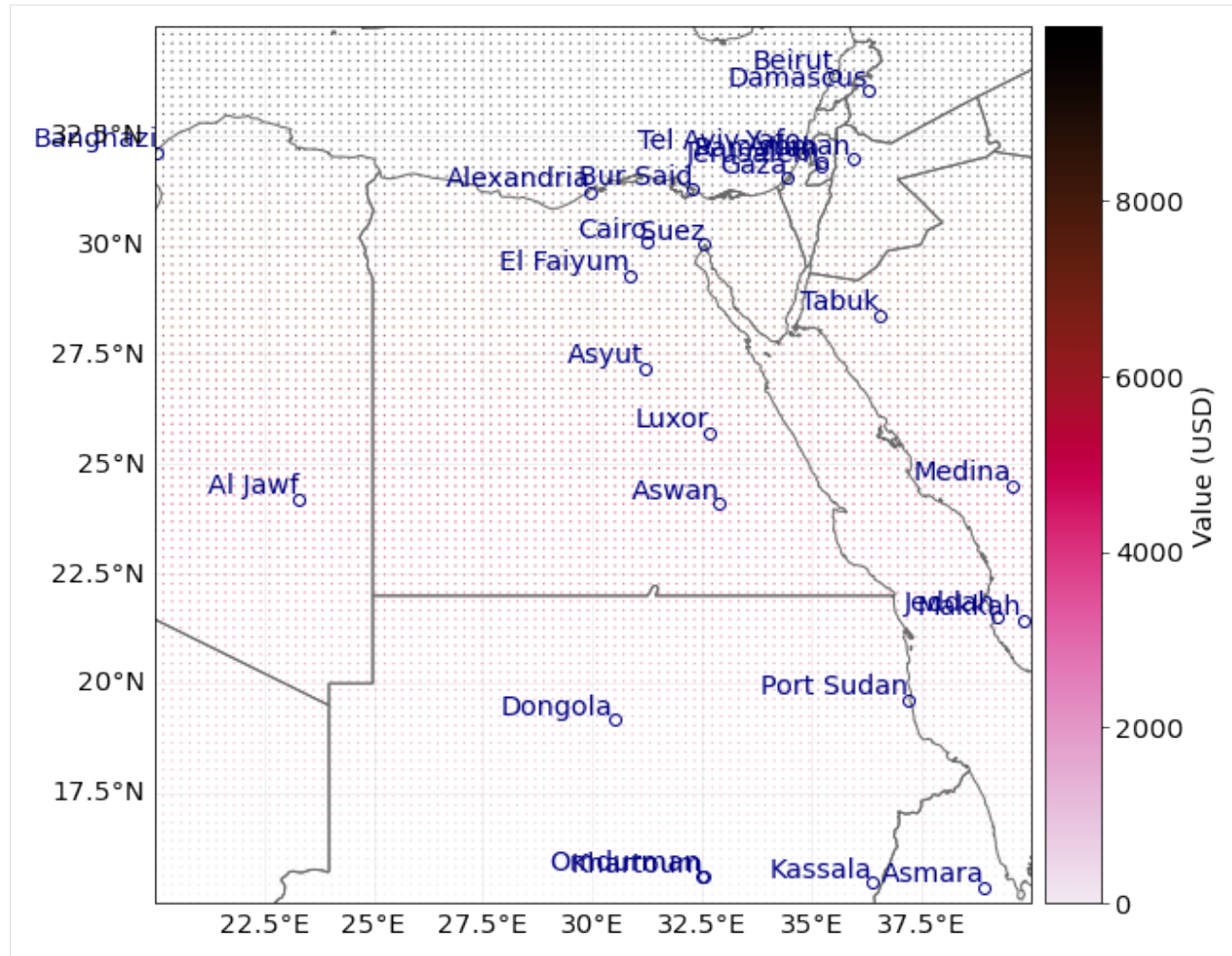
The method `plot_raster()` rasterizes the points into the given resolution. Use the `save_tiff` option to save the resulting `tiff` file and the `res_raster` option to re-set the raster's resolution.

Finally, the method `plot_basemap()` plots the scatter points over a satellite image using `contextily` library.

```
[21]: # Example 1: plot_hexbin method
print('\x1b[1;03;30;30m' + 'Plotting exp_df.' + '\x1b[0m')
axs = exp.plot_hexbin()

# further methods to check out:
# axs.set_xlim(15, 45) to modify x-axis borders, axs.set_ylim(10, 40) to modify y-axis
↪borders
# further keyword arguments to play around with: pop_name, buffer, gridsize, ...

Plotting exp_df.
```

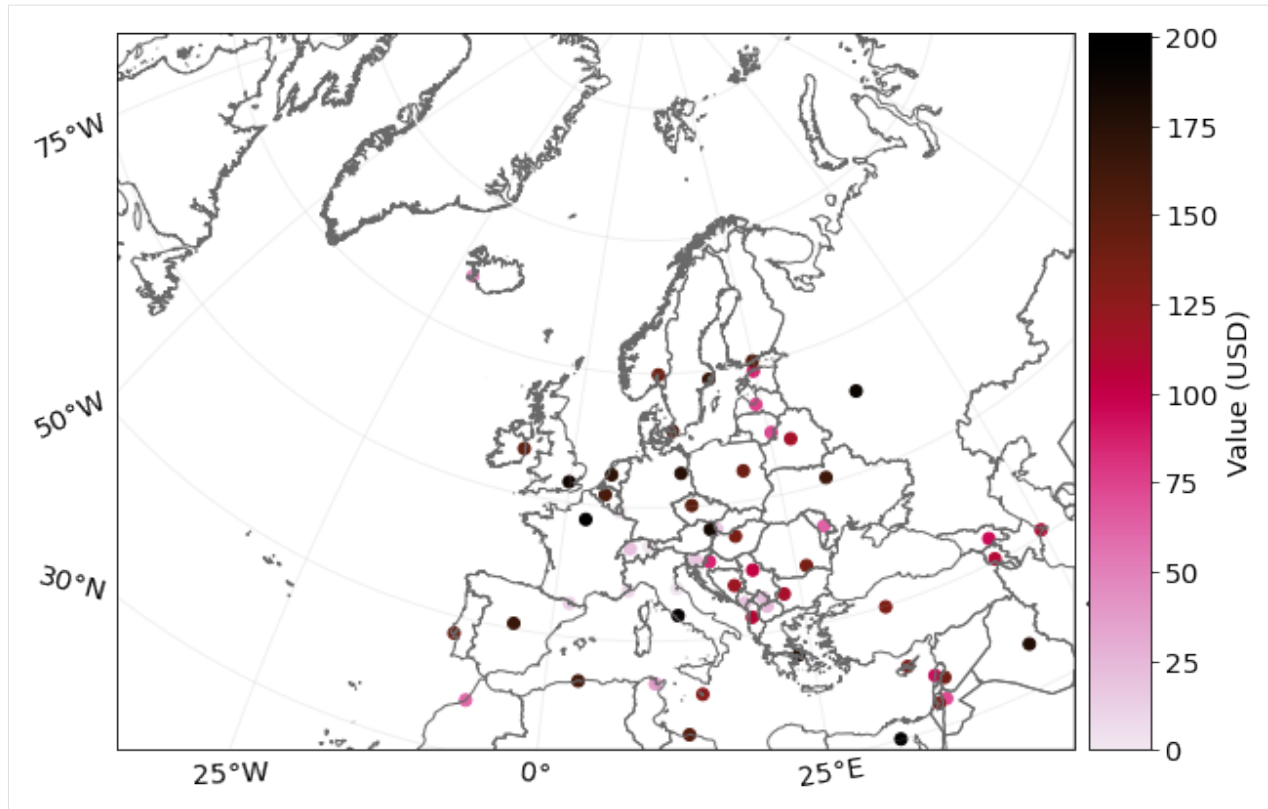



```
[22]: # Example 2: plot_scatter method
```

```
exp_gpd.to_crs('epsg:3035', inplace=True)
exp_gpd.plot_scatter(pop_name=False)
```

```
2021-06-04 17:07:33,832 - climada.entity.exposures.base - INFO - Setting latitude and
↳ longitude attributes.
```

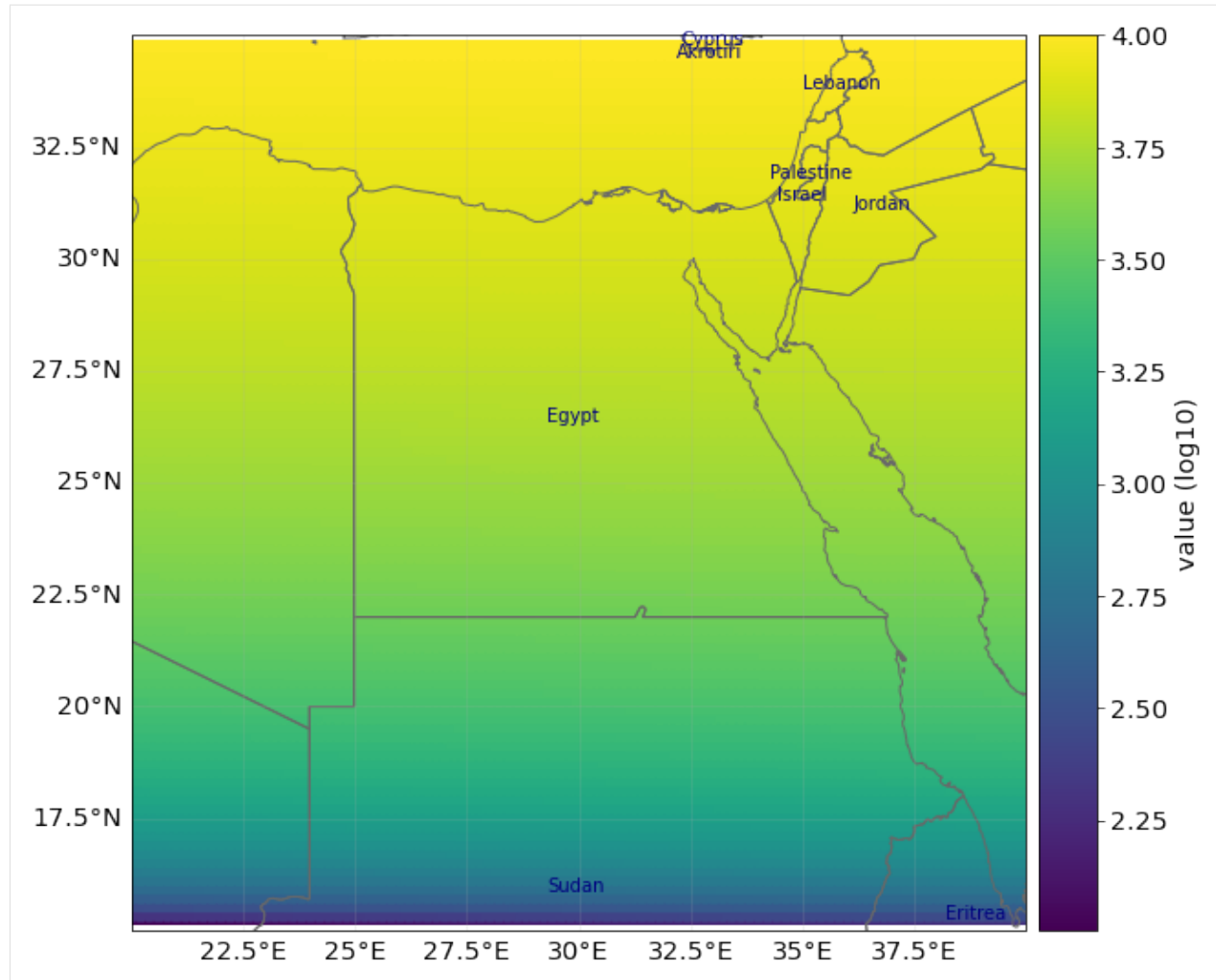
```
[22]: <GeoAxesSubplot:>
```



```
[23]: # Example 3: plot_raster method
from climada.util.plot import add_cntry_names # use climada's plotting utilities
ax = exp.plot_raster() # plot with same resolution as data
add_cntry_names(ax, [exp.gdf.longitude.min(), exp.gdf.longitude.max(), exp.gdf.latitude.
    ↪ min(), exp.gdf.latitude.max()])

# use keyword argument save_tiff='filepath.tiff' to save the corresponding raster in tiff_
    ↪ format
# use keyword argument raster_res='desired number' to change resolution of the raster.

2021-06-04 17:07:42,654 - climada.util.coordinates - INFO - Raster from resolution 0.
    ↪ 20202020202019355 to 0.20202020202019355.
```



```
[24]: # Example 4: plot_basemap method
import contextily as ctx
# select the background image from the available ctx.sources
ax = exp_tmpl.plot_basemap(buffer=30000, cmap='brg') # using open street map
ax = exp_tmpl.plot_basemap(buffer=30000, url=ctx.sources.T_WATERCOLOR, cmap='brg',
↪zoom=9) # set image zoom
```

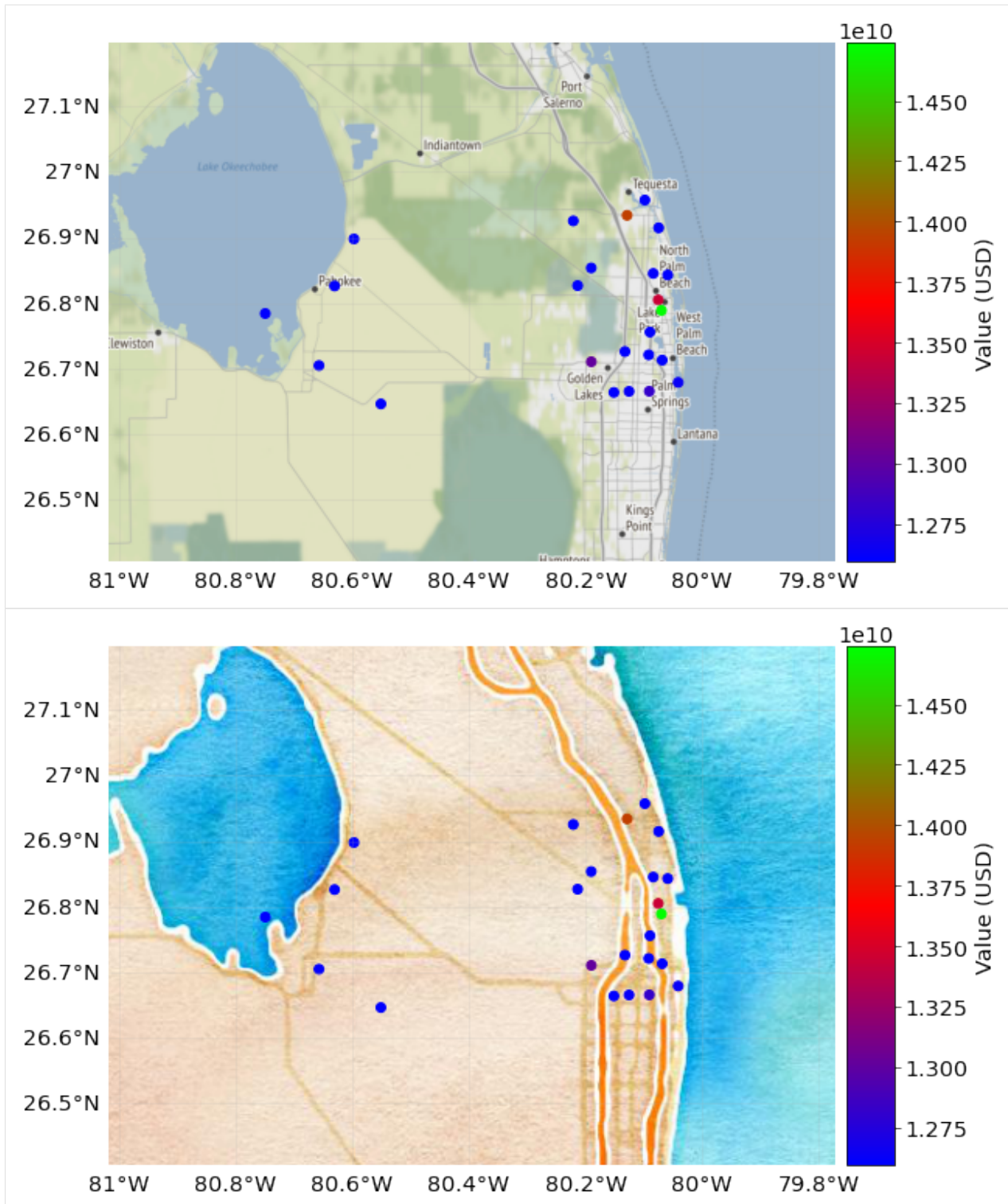
2021-06-04 17:07:51,154 - climada.entity.exposures.base - INFO - Setting latitude and
↪longitude attributes.

/Users/zeliestalhanske/miniconda3/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(

2021-06-04 17:07:55,046 - climada.entity.exposures.base - INFO - Setting latitude and
↪longitude attributes.

2021-06-04 17:07:55,124 - climada.entity.exposures.base - INFO - Setting latitude and
↪longitude attributes.

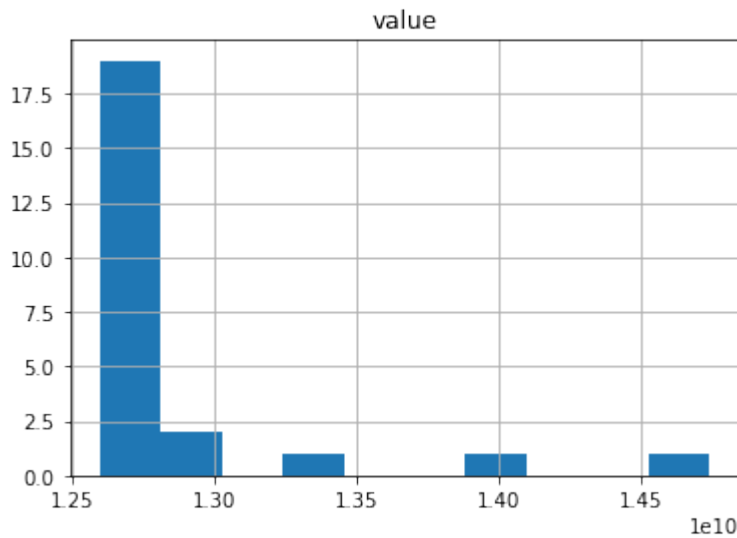
2021-06-04 17:07:58,604 - climada.entity.exposures.base - INFO - Setting latitude and
↪longitude attributes.



Since `Exposures` is a `GeoDataFrame`, any function for visualization from `geopandas` can be used. Check [making maps](#) and [examples gallery](#).

```
[25]: # other visualization types
exp_templ.gdf.hist(column='value')
```

```
[25]: array([[<AxesSubplot:title={'center':'value'}>]], dtype=object)
```



Part 4: Write (Save) Exposures Exposures can be saved in any format available for GeoDataFrame (see `fiona.supported_drivers`) and DataFrame ([pandas IO tools](#)). Take into account that in many of these formats the metadata (e.g. variables `ref_year`, `value_unit` and `tag`) will not be saved. Use instead the format `hdf5` provided by Exposures methods `write_hdf5()` and `read_hdf5()` to handle all the data.

```
[26]: import fiona; fiona.supported_drivers
from climada import CONFIG
results = CONFIG.local_data.save_dir.dir()

# GeoDataFrame default: ESRI shape file in current path. metadata not saved!
exp_tmpl.gdf.to_file(results.joinpath('exp_tmpl'))

# DataFrame save to csv format. geometry written as string, metadata not saved!
exp_tmpl.gdf.to_csv(results.joinpath('exp_tmpl.csv'), sep='\t')
```

```
[27]: # write as hdf5 file
exp_tmpl.write_hdf5(results.joinpath('exp_temp.h5'))

2021-06-04 17:07:59,684 - climada.entity.exposures.base - INFO - Writting /Users/
↳ zeliestalhanske/python_projects/climada_python/doc/tutorial/results/exp_temp.h5

/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/IPython/
↳ core/interactiveshell.py:3437: PerformanceWarning:
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->mixed,key->block2_values] [items->Index([
↳ 'geometry'], dtype='object')]

exec(code_obj, self.user_global_ns, self.user_ns)
```

Finally, as with any Python object, use climada's save option to save it in pickle format.

```
[28]: # save in pickle format
from climada.util.save import save
# this generates a results folder in the current path and stores the output there
```

(continues on next page)

(continued from previous page)

```
save('exp_tmpl.pkl.p', exp_tmpl) # creates results folder and stores there
```

```
2021-06-04 17:07:59,755 - climada.util.save - INFO - Written file /Users/zeliestalhanske/
python_projects/climada_python/doc/tutorial/results/exp_tmpl.pkl.p
```

Part 5: Dask - improving performance for big exposure Dask is used in some methods of CLIMADA and can be activated easily by providing the scheduler.

```
[29]: # set_geometry_points is expensive for big exposures
# for small amount of data, the execution time might be even greater when using dask
exp.gdf.drop(columns=['geometry'], inplace=True)
print(exp.gdf.head())
%time exp.set_geometry_points(scheduler='processes')
print(exp.gdf.head())
```

	value	latitude	longitude	impf_TC
0	0	15.0	20.000000	1
1	1	15.0	20.202020	1
2	2	15.0	20.404040	1
3	3	15.0	20.606061	1
4	4	15.0	20.808081	1

```
2021-06-04 17:07:59,771 - climada.util.coordinates - INFO - Setting geometry points.
CPU times: user 327 ms, sys: 130 ms, total: 457 ms
Wall time: 3.22 s
```

	value	latitude	longitude	impf_TC	geometry
0	0	15.0	20.000000	1	POINT (20.000000 15.000000)
1	1	15.0	20.202020	1	POINT (20.202020 15.000000)
2	2	15.0	20.404040	1	POINT (20.404040 15.000000)
3	3	15.0	20.606061	1	POINT (20.606061 15.000000)
4	4	15.0	20.808081	1	POINT (20.808081 15.000000)

5.3 LitPop class

5.3.1 Introduction

LitPop is an *Exposures*-type class. It is used to initiate grided exposure data with estimates of either asset value, economic activity or population based on nightlight intensity and population count data.

Background

The modeling of economic disaster risk on a global scale requires high-resolution maps of exposed asset values. We have developed a generic and scalable method to downscale national asset value estimates proportional to a combination of nightlight intensity (“Lit”) and population data (“Pop”).

Asset exposure value is disaggregated to the grid points proportionally to $Lit^m Pop^n$, computed at each grid cell:

$Lit^m Pop^n = Lit^m * Pop^n$, with $exponents = [m, n] \in +$ (Default values are $m = n = 1$).

For more information please refer to the related publication (<https://doi.org/10.5194/essd-12-817-2020>) and data archive (<https://doi.org/10.3929/ethz-b-000331316>).

How to cite: Eberenz, S., Stocker, D., Rösli, T., and Bresch, D. N.: *Asset exposure data for global physical risk assessment*, Earth Syst. Sci. Data, 12, 817–833, <https://doi.org/10.5194/essd-12-817-2020>, 2020.

Input data

Note: All required data except for the population data from Gridded Population of the World (GPW) is downloaded automatically when an `LitPop.set_*` method is called.

Nightlight intensity

Black Marble annual composite of the VIIRS day-night band (Grayscale) at 15 arcsec resolution is downloaded from the NASA Earth Observatory: <https://earthobservatory.nasa.gov/Features/NightLights> (available for 2012 and 2016 at 15 arcsec resolution (~500m)). The first time a nightlight image is used, it is downloaded and stored locally. This might take some time.

Population count

Gridded Population of the World (GPW), v4: Population Count, v4.10, v4.11 or later versions (2000, 2005, 2010, 2015, 2020), available from <http://sedac.ciesin.columbia.edu/data/collection/gpw-v4/sets/browse>.

The GPW file of the year closest to the requested year (`reference_year`) is required. To download GPW data a (free) login for the NASA SEDAC website is required.

Direct download links are available, also for older versions, i.e.: - v4.11: http://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/gpw-v4-population-count-rev11/gpw-v4-population-count-rev11_2015_30_sec_tif.zip - v4.10: http://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/gpw-v4-population-count-rev10/gpw-v4-population-count-rev10_2015_30_sec_tif.zip, - Overview over all versions of GPW v4: <https://beta.sedac.ciesin.columbia.edu/data/collection/gpw-v4/sets/browse>

The population data from GWP needs to be downloaded manually as TIFF from this site and placed in the `SYSTEM_DIR` folder of your climada installation.

Downloading existing LitPop asset exposure data

Readily computed LitPop asset exposure data based on *Lit¹Pop¹* for 224 countries, distributing produced capital / non-financial wealth of 2014 at a resolution of 30 arcsec can be downloaded from the ETH Research Repository: <https://doi.org/10.3929/ethz-b-000331316>. The dataset contains gridded data for more than 200 countries as CSV files.

5.3.2 Attributes

The `LitPop` class inherits from ``Exposures <climada_entity_Exposures.ipynb#Exposures-class``. It adds the following attributes:

```
exponents : Defining powers (m, n) with which nightlights and population go into Lit**m.
            ↪ * Pop**n.
fin_mode : Socio-economic indicator to be used as total asset value for disaggregation.
gpw_version : Version number of GPW population data, e.g. 11 for v4.11
```

fin_mode

The choice of `fin_mode` is crucial. Implemented choices are: * `'pc'`: produced capital (Source: World Bank), incl. manufactured or built assets such as machinery, equipment, and physical structures. The pc-data is stored in the subfolder `data/system/Wealth-Accounts_CSV/`. Source: <https://datacatalog.worldbank.org/dataset/wealth-accounting> * `'pop'`: population count (source: GPW, same as gridded population) * `'gdp'`: gross-domestic product (Source: World Bank) * `'income_group'`: gdp multiplied by country's income group+1 * `'nfw'`: non-financial household wealth (Source: Credit Suisse) * `'tw'`: total household wealth (Source: Credit Suisse) * `'norm'`: normalized, total value of country or region is 1. * `'none'`: None – LitPop per pixel is returned unchanged

Regarding the 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.

5.3.3 Key Methods

- `set_countries`: set exposure for one or more countries, see section `set_countries` below.
- `set_country`: alias of `set_countries`
- `set_lit`: wrapper around `set_countries` and `set_custom_shape` to load nightlight data to exposure.
- `set_pop`: wrapper around `set_countries` and `set_custom_shape_population` to load pure population data to exposure. This can be used to initiate a population exposure set.
- `set_custom_shape_from_countries`: given a shape and a list of countries, exposure is initiated for the countries and then cropped to the shape. See section *Set custom shapes* below.
- `set_custom_shape`: given any shape or geometry and an estimate of total values, exposure is initiated for the shape directly. See section *Set custom shapes* below.

```
[2]: # Import class LitPop:
from climada.entity import LitPop
```

set_countries

In the following, we will create exposure data sets and plots for a variety of countries, comparing different settings. ##### Default Settings Per default, the exposure entity was initiated using the default parameters, i.e. a resolution of 30 arcsec, produced capital 'pc' as total asset value and using the exponents (1, 1).

```
[5]: # Initiate a default LitPop exposure entity for Switzerland and Liechtenstein (ISO3-
↳ Codes 'CHE' and 'LIE'):
exp = LitPop()
try:
    exp.set_countries(['CHE', 'Liechtenstein']) # you can provide either single_
↳ countries or a list of countries
except FileNotFoundError as err:
    print("Reason for error: The GPW population data has not been downloaded, c.f.
↳ section 'Input data' above.")
    raise err
exp.plot_scatter()

# Note that `exp.gdf.region_id` is a number identifying each country:
print('\n Region IDs (`region_id`) in this exposure:')
print(exp.gdf.region_id.unique())
```



```

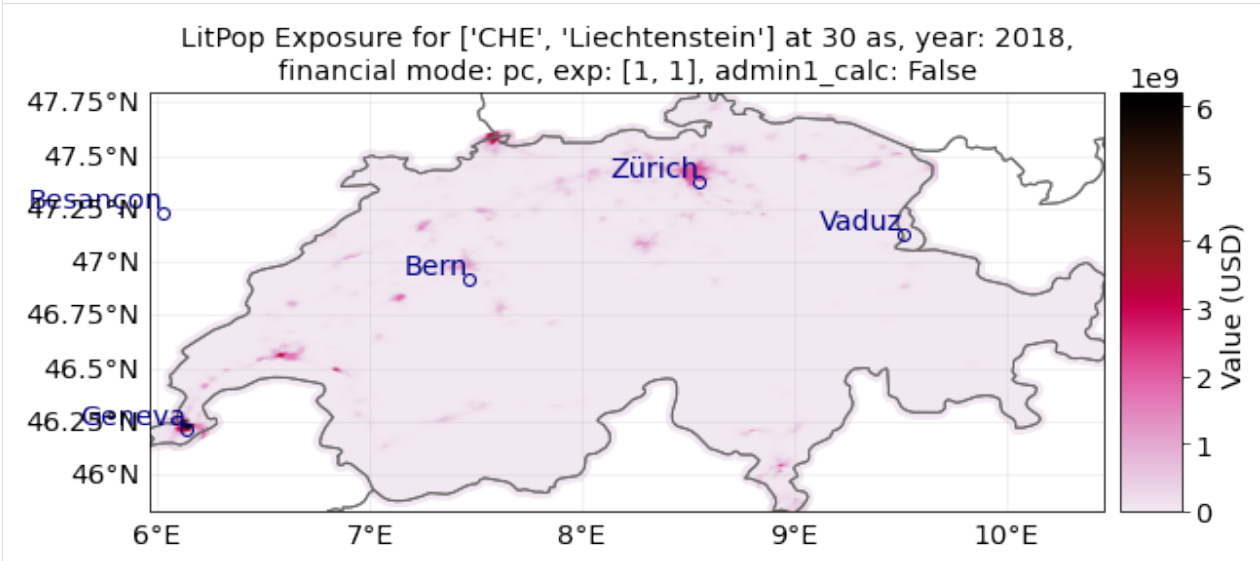
2021-06-25 11:28:33,942 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: CHE (756)...

2021-06-25 11:28:33,943 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW population data: 2020
2021-06-25 11:28:33,944 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2021-06-25 11:28:38,136 - climada.util.finance - INFO - GDP CHE 2014: 7.092e+11.
2021-06-25 11:28:38,586 - climada.util.finance - INFO - GDP CHE 2018: 7.051e+11.
2021-06-25 11:28:38,933 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: LIE (438)...

2021-06-25 11:28:38,934 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW population data: 2020
2021-06-25 11:28:38,935 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2021-06-25 11:28:39,053 - climada.util.finance - WARNING - No data available for country.
↳ Using non-financial wealth instead
2021-06-25 11:28:39,504 - climada.util.finance - INFO - GDP LIE 2018: 6.877e+09.
2021-06-25 11:28:39,510 - climada.util.finance - WARNING - No data for country, using
↳mean factor.
2021-06-25 11:28:39,566 - climada.entity.exposures.base - INFO - Hazard type not set in
↳impf_
2021-06-25 11:28:39,567 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-25 11:28:39,567 - climada.entity.exposures.base - INFO - cover not set.
2021-06-25 11:28:39,568 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-25 11:28:39,570 - climada.entity.exposures.base - INFO - centr_ not set.

Region IDs (`region_id`) in this exposure:
[756 438]

```



fin_mode, resolution and exponents

Instead on produced capital, we can also downscale other available macroeconomic indicators as estimates of asset value. The indicator can be set via the parameter `fin_mode`, either to 'pc', 'pop', 'gdp', 'income_group', 'nfw', 'tw', 'norm', or 'none'. See descriptions of each alternative above in the introduction.

We can also change the resolution via `res_arcsec` and the exponents.

The default resolution is 30 arcsec \approx 1 km. A resolution of 3600 arcsec = 1 degree corresponds to roughly 110 km close to the equator.

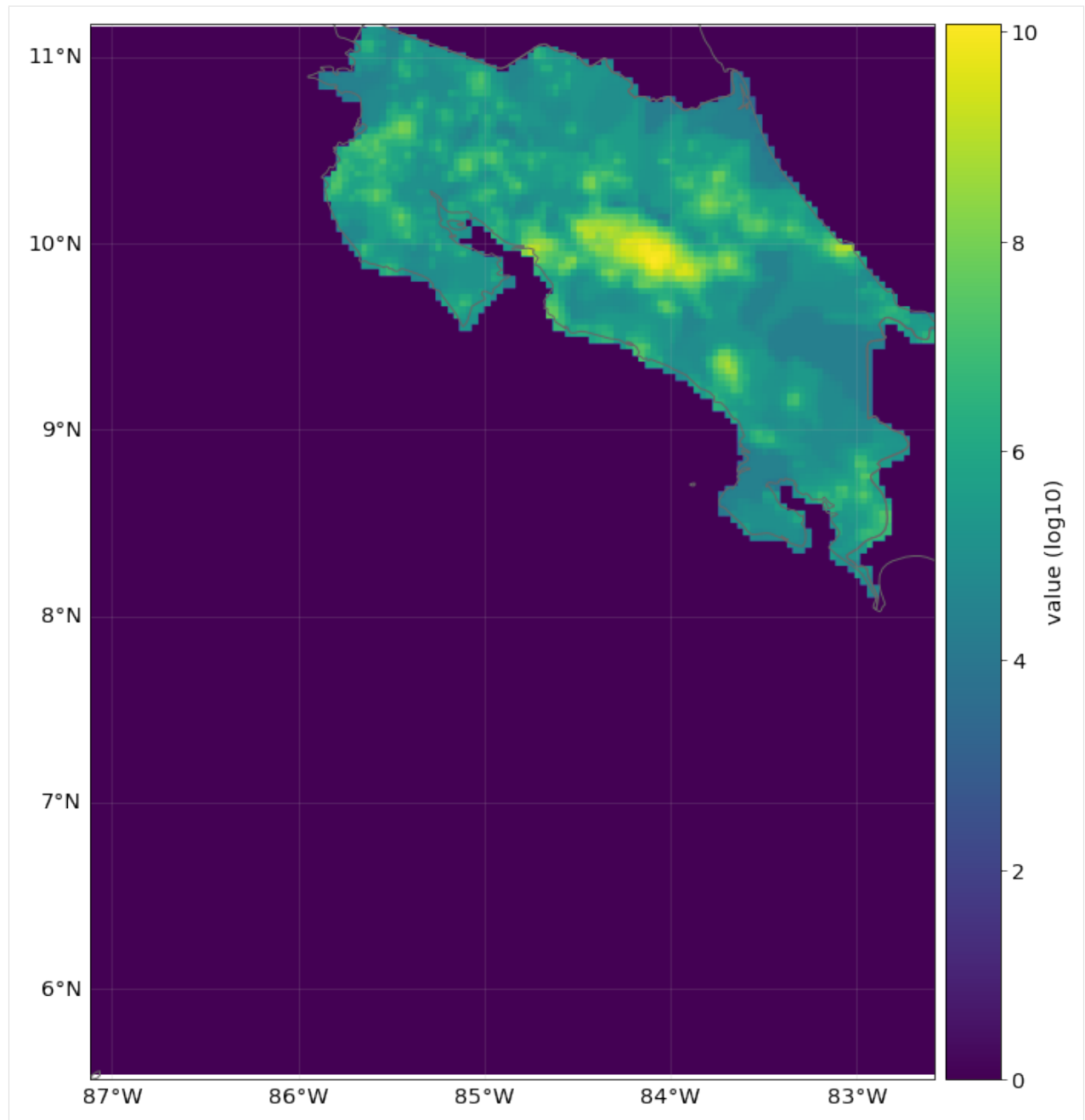
Let's initiate an exposure instance with the financial mode "income_group" and at a resolution of 120 arcsec (roughly 4 km).

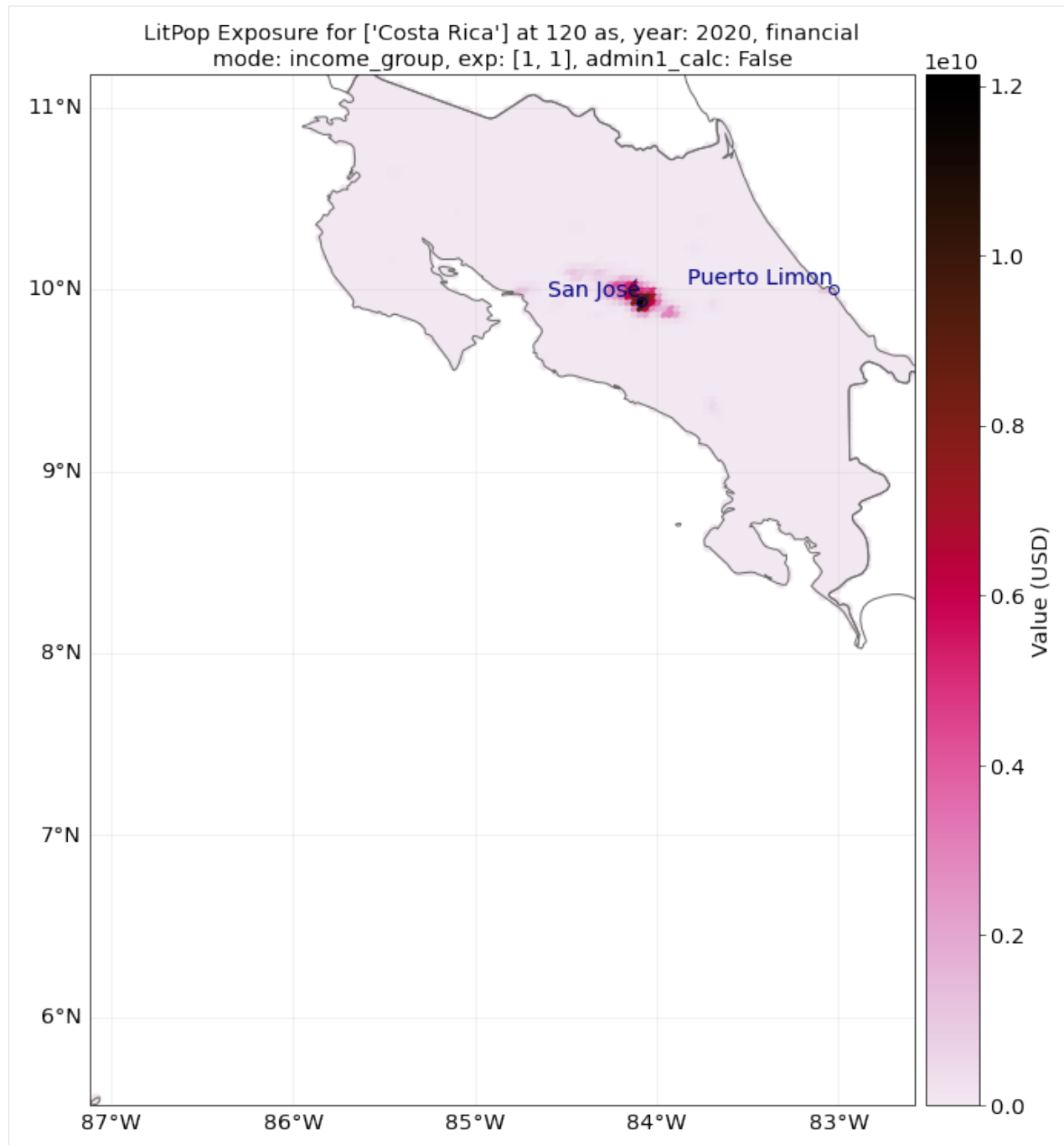
```
[27]: # Initiate a LitPop exposure entity for Costa Rica with varied resolution, fin_mode, and
      ↪ exponents:
exp = LitPop()
exp.set_country('Costa Rica', fin_mode='income_group', res_arcsec=120, exponents=(1,1))
      ↪ # change the parameters and see what happens...
# exp.set_country('Costa Rica', fin_mode='gdp', res_arcsec=90, exponents=(3,0)) # example
      ↪ of variation
exp.plot_raster() # note the log scale of the colorbar
exp.plot_scatter()

2021-06-23 11:50:02,676 - climada.entity.exposures.litpop.litpop - INFO -
  LitPop: Init Exposure for country: CRI (188)...

2021-06-23 11:50:02,677 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
      ↪ Version v4.11
2021-06-23 11:50:03,906 - climada.util.finance - INFO - GDP CRI 2019: 6.180e+10.
2021-06-23 11:50:03,948 - climada.util.finance - INFO - Income group CRI 2019: 3.
2021-06-23 11:50:03,960 - climada.entity.exposures.base - INFO - Hazard type not set in
      ↪ impf_
2021-06-23 11:50:03,960 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-23 11:50:03,961 - climada.entity.exposures.base - INFO - cover not set.
2021-06-23 11:50:03,961 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-23 11:50:03,962 - climada.entity.exposures.base - INFO - centr_ not set.
2021-06-23 11:50:03,967 - climada.util.coordinates - INFO - Raster from resolution 0.
      ↪ 03333333333333144 to 0.03333333333333144.

[27]: <GeoAxesSubplot:title={'center':"LitPop Exposure for ['Costa Rica'] at 120 as, year:
      ↪ 2020, financial\nmode: income_group, exp: [1, 1], admin1_calc: False"}>
```





Reference year

Additionally, we can change the year our exposure is supposed to represent. For this, nightlight and population data are used that are closest to the requested years. Macroeconomic indicators like produced capital are interpolated from available data or scaled proportional to GDP.

Let's load a population exposure map for Switzerland in 2000 and 2021 with a resolution of 300 arcsec:

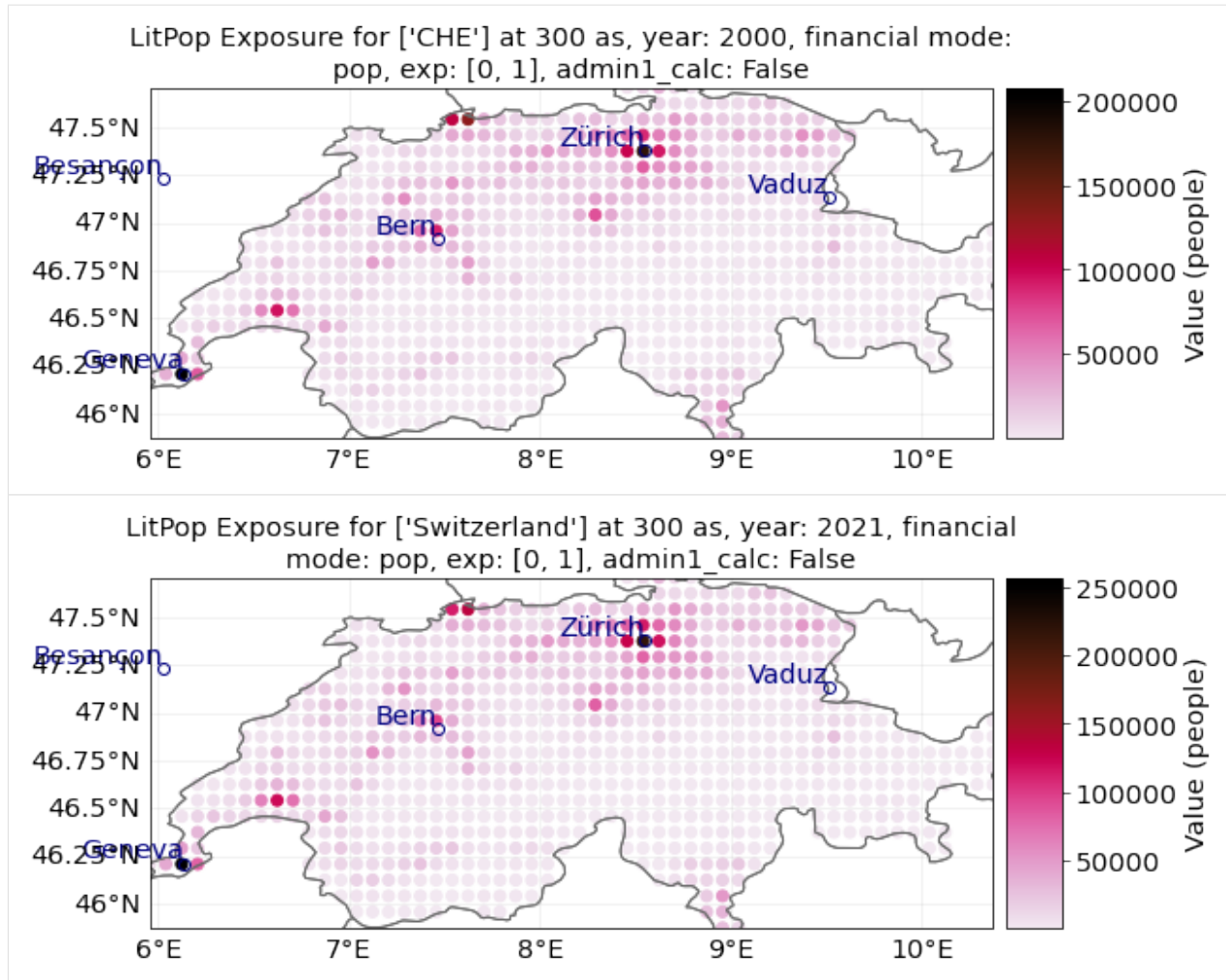
```
[29]: pop_2000 = LitPop()
pop_2000.set_country('CHE', fin_mode='pop', res_arcsec=300, exponents=(0,1), reference_
↳year=2000)
# Alternatively, we can use `set_population`:
pop_2021 = LitPop()
pop_2021.set_population(countries='Switzerland', res_arcsec=300, reference_year=2021)
# Since no population data for 2021 is available, the closest data point, 2020, is used.
↳(see LOGGER.warning)
pop_2000.plot_scatter()
pop_2021.plot_scatter()
"""Note the difference in total values on the color bar."""

2021-06-23 11:57:57,916 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: CHE (756)...

2021-06-23 11:57:57,918 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2021-06-23 11:57:58,302 - climada.entity.exposures.base - INFO - Hazard type not set in
↳impf_
2021-06-23 11:57:58,302 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-23 11:57:58,303 - climada.entity.exposures.base - INFO - cover not set.
2021-06-23 11:57:58,304 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-23 11:57:58,305 - climada.entity.exposures.base - INFO - centr_ not set.
2021-06-23 11:57:58,569 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: CHE (756)...

2021-06-23 11:57:58,570 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2021. Using nearest available year for GPW population data: 2020
2021-06-23 11:57:58,571 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2021-06-23 11:57:58,942 - climada.entity.exposures.base - INFO - Hazard type not set in
↳impf_
2021-06-23 11:57:58,943 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-23 11:57:58,943 - climada.entity.exposures.base - INFO - cover not set.
2021-06-23 11:57:58,944 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-23 11:57:58,945 - climada.entity.exposures.base - INFO - centr_ not set.
```

[29]: 'Note the difference in total values on the color bar.'



set_nightlight_intensity and set_population

These wrapper methods can be used to produce exposures that are showing purely nightlight intensity or purely population count.

```
[3]: exp_nightlights = LitPop()
exp_population = LitPop()
res = 30
country = 'JAM' # Try different countries, i.e. 'JAM', 'CHE', 'RWA', 'MEX'
markersize = 4 # for plotting
buffer_deg=.04

exp_nightlights.set_nightlight_intensity(countries=country, res_arcsec=res) # nightlight_
exp_nightlights.plot_hexbin(linewidth=markersize, buffer=buffer_deg)
# Compare to the population map:
exp_population.set_population(countries=country, res_arcsec=res)
exp_population.plot_hexbin(linewidth=markersize, buffer=buffer_deg)
# Compare to default LitPop exposures:
```

(continues on next page)

(continued from previous page)

```

exp = LitPop()
exp.set_countries('JAM', res_arcsec=res)
exp.plot_hexbin(linewidth=markersize, buffer=buffer_deg)

2021-06-23 14:01:33,293 - climada.entity.exposures.litpop.litpop - INFO -
  LitPop: Init Exposure for country: JAM (388)...

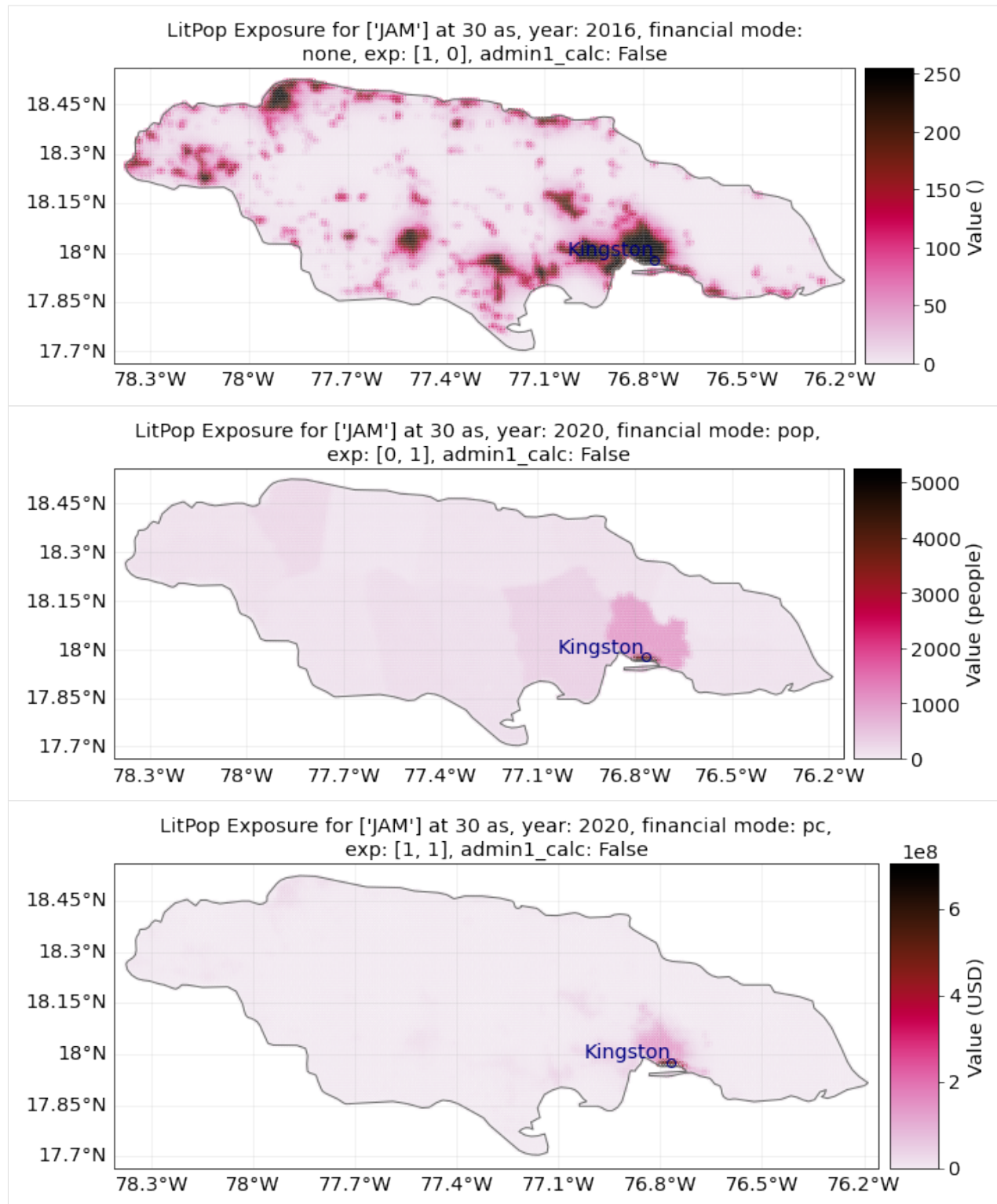
2021-06-23 14:01:33,297 - climada.entity.exposures.litpop.gpw_population - WARNING -
  ↳Reference year: 2016. Using nearest available year for GPW population data: 2015
2021-06-23 14:01:33,299 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
  ↳Version v4.11
2021-06-23 14:01:34,441 - climada.entity.exposures.base - INFO - Hazard type not set in
  ↳impf_
2021-06-23 14:01:34,442 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-23 14:01:34,442 - climada.entity.exposures.base - INFO - cover not set.
2021-06-23 14:01:34,443 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-23 14:01:34,445 - climada.entity.exposures.base - INFO - centr_ not set.
2021-06-23 14:01:40,165 - climada.entity.exposures.litpop.litpop - INFO -
  LitPop: Init Exposure for country: JAM (388)...

2021-06-23 14:01:40,167 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
  ↳Version v4.11
2021-06-23 14:01:40,976 - climada.entity.exposures.base - INFO - Hazard type not set in
  ↳impf_
2021-06-23 14:01:40,977 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-23 14:01:40,977 - climada.entity.exposures.base - INFO - cover not set.
2021-06-23 14:01:40,978 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-23 14:01:40,980 - climada.entity.exposures.base - INFO - centr_ not set.
2021-06-23 14:01:46,518 - climada.entity.exposures.litpop.litpop - INFO -
  LitPop: Init Exposure for country: JAM (388)...

2021-06-23 14:01:46,519 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
  ↳Version v4.11
2021-06-23 14:01:47,929 - climada.util.finance - INFO - GDP JAM 2014: 1.390e+10.
2021-06-23 14:01:48,373 - climada.util.finance - INFO - GDP JAM 2019: 1.646e+10.
2021-06-23 14:01:48,400 - climada.entity.exposures.base - INFO - Hazard type not set in
  ↳impf_
2021-06-23 14:01:48,400 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-23 14:01:48,402 - climada.entity.exposures.base - INFO - cover not set.
2021-06-23 14:01:48,404 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-23 14:01:48,407 - climada.entity.exposures.base - INFO - centr_ not set.

[3]: <GeoAxesSubplot:title={'center':"LitPop Exposure for ['JAM'] at 30 as, year: 2020,
  ↳financial mode: pc,\nexp: [1, 1], admin1_calc: False"}>

```

For **Switzerland**, population is resolved on the 3rd administrative level, with 2538 distinct geographical units. Therefore, the purely population-based map is highly resolved.

For **Jamaica**, population is only resolved on the 1st administrative level, with only 14 distinct geographical units. Therefore, the purely population-based map shows large monotonous patches. The combination of Lit and Pop results

in a concentration of asset value estimates around the capital city Kingston.

5.3.4 Set custom shapes

The methods `LitPop.set_custom_shape_from_countries` and `LitPop.set_custom_shape` initiate a `LitPop`-exposure instance for a given custom shape instead of a country. This can be used to initiate exposure for admin1-regions, i.e. cantons, states, districts, - but also for bounding boxes etc.

The difference between the two methods is that for `set_custom_shape_from_countries`, the exposure for one or more whole countries is initiated first and then it is cropped to the shape. Please make sure that the shape is contained in the given countries. With `set_custom_shape`, the shape is initiated directly which is much more resource efficient but requires a `total_value` to be provided by the user.

A population exposure for a custom shape can be initiated directly via `set_population` without providing `total_value`.

Using `LitPop.set_custom_shape_from_countries` and `LitPop.set_custom_shape` we initiate `LitPop` exposures for Florida:

```
[13]: import time
import climada.util.coordinates as u_coord
import climada.entity.exposures.litpop as lp

country_iso3a = 'USA'
state_name = 'Florida'
reslution_arcsec = 600
"""First, we need to get the shape of Florida:."""
admin1_info, admin1_shapes = u_coord.get_admin1_info(country_iso3a)
admin1_info = admin1_info[country_iso3a]
admin1_shapes = admin1_shapes[country_iso3a]
admin1_names = [record['name'] for record in admin1_info]
print(admin1_names)
for idx, name in enumerate(admin1_names):
    if admin1_names[idx]==state_name:
        break
print('Florida index: ' + str(idx))

"""Secondly, we estimate the `total_value`"""
# `total_value` required user input for `set_custom_shape`, here we assume 5% of total_
↪value of the whole USA:
total_value = 0.05 * lp.get_total_value_per_country(country_iso3a, 'pc', 2020)

"""Then, we can initiate the exposures for Florida:."""
exp = LitPop()
start = time.process_time()
exp.set_custom_shape(admin1_shapes[idx], total_value, res_arcsec=600, reference_
↪year=2020)
print(f'\n Runtime `set_custom_shape` : {time.process_time() - start:1.2f} sec.\n')
exp.plot_scatter(vmin=100, buffer=.5)
```

```
['Minnesota', 'Washington', 'Idaho', 'Montana', 'North Dakota', 'Michigan', 'Maine',
↪ 'Ohio', 'New Hampshire', 'New York', 'Vermont', 'Pennsylvania', 'Arizona', 'California',
↪ 'New Mexico', 'Texas', 'Alaska', 'Louisiana', 'Mississippi', 'Alabama', 'Florida',
↪ 'Georgia', 'South Carolina', 'North Carolina', 'Virginia', 'District of Columbia',
↪ 'Maryland', 'Delaware', 'New Jersey', 'Connecticut', 'Rhode Island', 'Massachusetts',
↪ 'Oregon', 'Hawaii', 'Utah', 'Wyoming', 'Nevada', 'Colorado', 'South Dakota', 'Nebraska',
↪ 'Kansas', 'Oklahoma', 'Iowa', 'Missouri', 'Wisconsin', 'Illinois', 'Kentucky',
↪ 'Arkansas', 'Tennessee', 'West Virginia', 'Indiana']
```

(continued from previous page)

Florida index: 20

2021-06-25 11:41:14,547 - climada.util.finance - INFO - GDP USA 2014: 1.753e+13.

2021-06-25 11:41:14,999 - climada.util.finance - INFO - GDP USA 2019: 2.143e+13.

2021-06-25 11:41:15,006 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
→ Version v4.112021-06-25 11:41:15,910 - climada.util.coordinates - INFO - Setting region_id 1610
→ points.

```

/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
→ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
→ ' is the preferred initialization method. When making the change, be mindful of axis
→ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
→ in-proj-6
    return _prepare_from_string(" ".join(pjargs))

```

2021-06-25 11:41:16,972 - climada.util.coordinates - INFO - Setting region_id 1 points.

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
→ 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.

```

```

    countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
→ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
→ ' is the preferred initialization method. When making the change, be mindful of axis
→ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
→ in-proj-6
    return _prepare_from_string(" ".join(pjargs))

```

2021-06-25 11:41:17,984 - climada.util.coordinates - INFO - Setting region_id 1 points.

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
→ 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.

```

```

    countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
→ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
→ ' is the preferred initialization method. When making the change, be mindful of axis
→ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
→ in-proj-6
    return _prepare_from_string(" ".join(pjargs))

```

2021-06-25 11:41:19,033 - climada.util.coordinates - INFO - Setting region_id 4 points.

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
→ 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.

```

```

    countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
→ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
→ ' is the preferred initialization method. When making the change, be mindful of axis
→ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
→ in-proj-6

```

(continued from previous page)

```

return _prepare_from_string(" ".join(pjargs))
2021-06-25 11:41:20,128 - climada.util.coordinates - INFO - Setting region_id 1 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))
2021-06-25 11:41:21,152 - climada.util.coordinates - INFO - Setting region_id 8 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))
2021-06-25 11:41:22,193 - climada.util.coordinates - INFO - Setting region_id 3 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))
2021-06-25 11:41:23,207 - climada.util.coordinates - INFO - Setting region_id 5 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6

```

(continued from previous page)

```

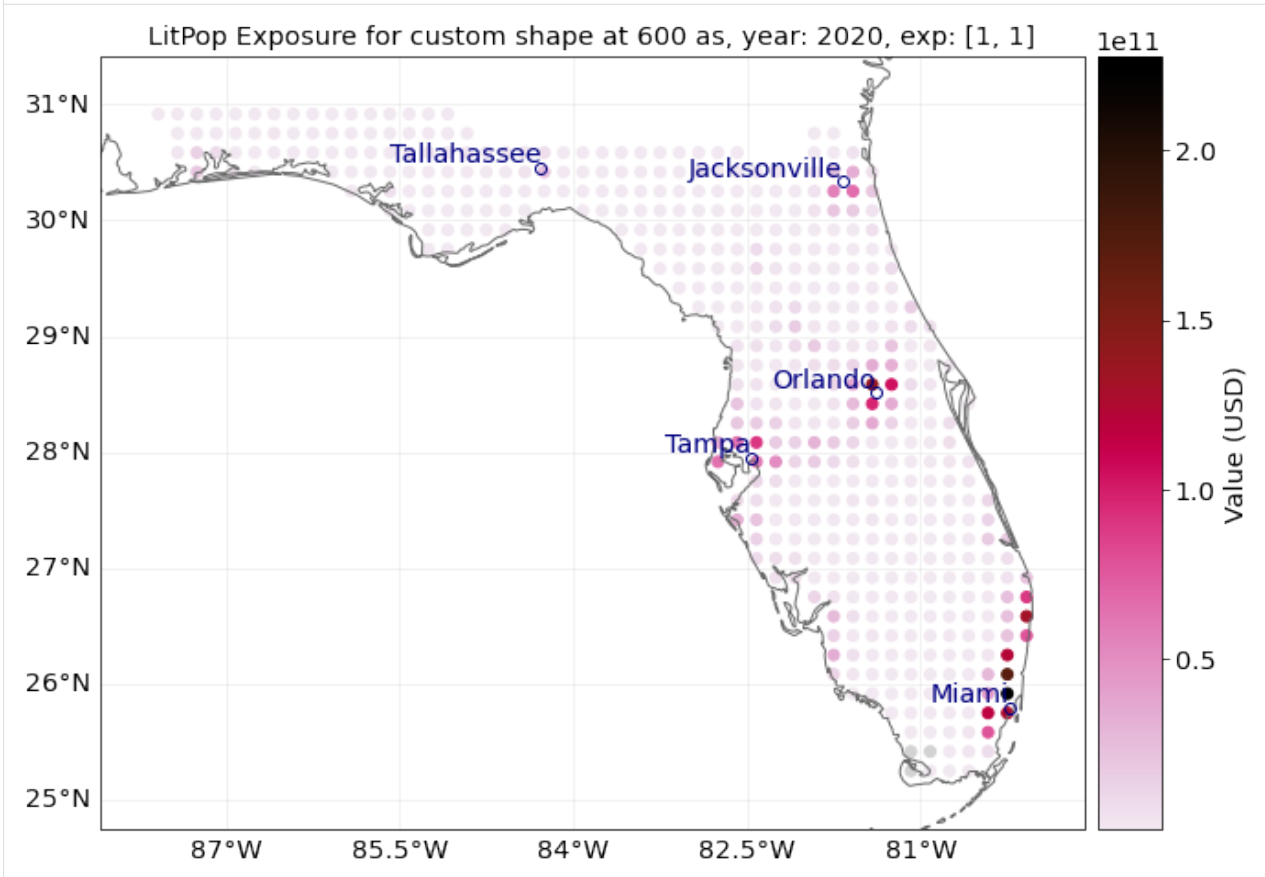
return _prepare_from_string(" ".join(pjargs))
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
↳ UserWarning: Geometry is in a geographic CRS. Results from 'area' are likely incorrect.
↳ Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this
↳ operation.

countries['area'] = countries.geometry.area

```

Runtime `set_custom_shape` : 9.01 sec.

[13]: <GeoAxesSubplot:title={'center':'LitPop Exposure for custom shape at 600 as, year: 2020, exp: [1, 1]'}>



```

[12]: # `set_custom_shape_from_countries` does not require `total_value`, but is slower to
↳ compute than `set_custom_shape`,
# because first, the exposure for the whole USA is initiated:
exp = LitPop()
start = time.process_time()
exp.set_custom_shape_from_countries(admin1_shapes[idx], country_iso3a, res_arcsec=600,
↳ reference_year=2020)
print(f'\n Runtime `set_custom_shape_from_countries` : {time.process_time() - start:1.2f}
↳ sec.\n')
exp.plot_scatter(vmin=100, buffer=.5)

```

(continues on next page)

(continued from previous page)

```

"""Note the differences in computational speed and total value between the two approaches
↪ """

```

```

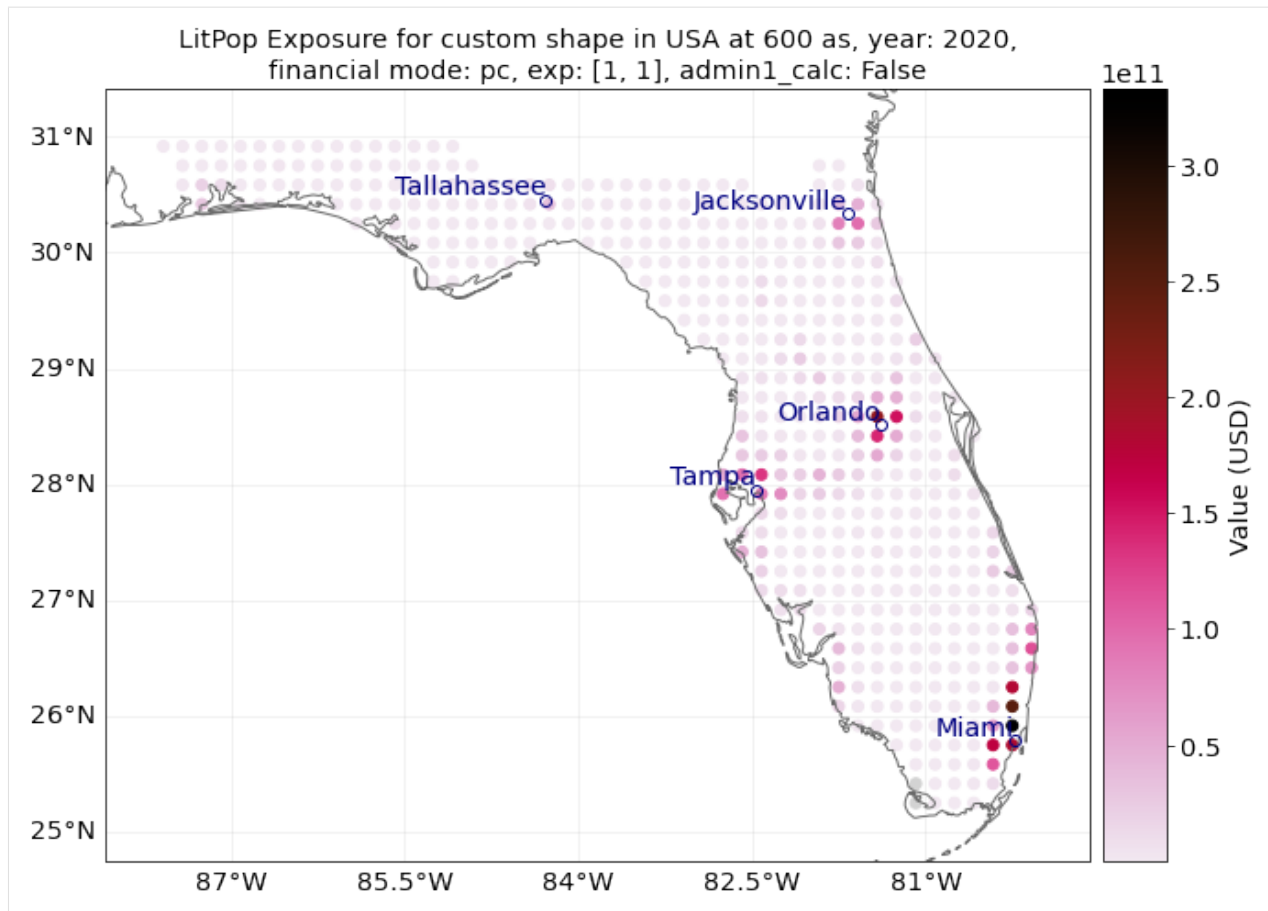
2021-06-25 11:40:25,127 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: USA (840)...

2021-06-25 11:40:25,129 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪ Version v4.11
2021-06-25 11:40:54,832 - climada.util.finance - INFO - GDP USA 2014: 1.753e+13.
2021-06-25 11:40:55,361 - climada.util.finance - INFO - GDP USA 2019: 2.143e+13.
2021-06-25 11:40:55,400 - climada.entity.exposures.base - INFO - Hazard type not set in
↪ impf_
2021-06-25 11:40:55,401 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-25 11:40:55,401 - climada.entity.exposures.base - INFO - cover not set.
2021-06-25 11:40:55,402 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-25 11:40:55,403 - climada.entity.exposures.base - INFO - centr_ not set.
2021-06-25 11:41:00,182 - climada.entity.exposures.base - INFO - Hazard type not set in
↪ impf_
2021-06-25 11:41:00,184 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-25 11:41:00,184 - climada.entity.exposures.base - INFO - cover not set.
2021-06-25 11:41:00,185 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-25 11:41:00,186 - climada.entity.exposures.base - INFO - centr_ not set.

Runtime `set_custom_shape_from_countries` : 33.33 sec.

```

[12]: 'Note the differences in computational speed and total value between the two approaches'



You can also define your own shape as a Polygon:

```
[14]: import time
from shapely.geometry import Polygon

"""initiate LitPop exposures for a geographical box around the city of Zurich:"""
bounds = (8.41, 47.25, 8.70, 47.47) # (min_lon, max_lon, min_lat, max_lat)
total_value=10000 # required user input for `set_custom_shape`, here we just assume USD_
↳ 10000 of total value
shape = Polygon([
    (bounds[0], bounds[3]),
    (bounds[2], bounds[3]),
    (bounds[2], bounds[1]),
    (bounds[0], bounds[1])
])

exp = LitPop()
import time
start = time.process_time()
exp.set_custom_shape(shape, total_value)
print(f'\n Runtime `set_custom_shape` : {time.process_time() - start:1.2f} sec.\n')
exp.plot_scatter()
# `set_custom_shape_from_countries` does not require `total_value`, but is slower to_
↳ compute:
start = time.process_time()
```

(continues on next page)

(continued from previous page)

```

exp.set_custom_shape_from_countries(shape, 'Switzerland')
print(f'\n Runtime `set_custom_shape_from_countries` : {time.process_time() - start:1.2f}
↳ sec.\n')
exp.plot_scatter()
"""Note the difference in total value between the two exposure sets!"""

"""For comparison, initiate population exposure for a geographical box around the city
↳ of Zurich:"""
exp_pop = LitPop()
start = time.process_time()
exp_pop.set_population(shape=shape)
print(f'\n Runtime `set_population` : {time.process_time() - start:1.2f} sec.\n')
exp_pop.plot_scatter()

"""Population exposure for a custom shape can be initiated directly via `set_population`
↳ without providing `total_value`"""

2021-06-25 11:41:55,436 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳ Reference year: 2018. Using nearest available year for GPW population data: 2020
2021-06-25 11:41:55,441 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳ Version v4.11
2021-06-25 11:41:55,575 - climada.util.coordinates - INFO - Setting region_id 972 points.

/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ ' is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
    return _prepare_from_string(" ".join(pjargs))
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
↳ 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.

countries['area'] = countries.geometry.area

Runtime `set_custom_shape` : 1.11 sec.

2021-06-25 11:41:57,492 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: CHE (756)...

2021-06-25 11:41:57,493 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳ Reference year: 2018. Using nearest available year for GPW population data: 2020
2021-06-25 11:41:57,494 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳ Version v4.11
2021-06-25 11:42:01,417 - climada.util.finance - INFO - GDP CHE 2014: 7.092e+11.
2021-06-25 11:42:01,861 - climada.util.finance - INFO - GDP CHE 2018: 7.051e+11.
2021-06-25 11:42:01,902 - climada.entity.exposures.base - INFO - Hazard type not set in
↳ impf_
2021-06-25 11:42:01,903 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-25 11:42:01,903 - climada.entity.exposures.base - INFO - cover not set.
2021-06-25 11:42:01,904 - climada.entity.exposures.base - INFO - deductible not set.

```

(continues on next page)

(continued from previous page)

```

2021-06-25 11:42:01,904 - climada.entity.exposures.base - INFO - centr_ not set.
2021-06-25 11:42:02,192 - climada.entity.exposures.base - INFO - Hazard type not set in.
↳ impf_
2021-06-25 11:42:02,193 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-25 11:42:02,194 - climada.entity.exposures.base - INFO - cover not set.
2021-06-25 11:42:02,194 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-25 11:42:02,196 - climada.entity.exposures.base - INFO - centr_ not set.

Runtime `set_custom_shape_from_countries` : 4.33 sec.

2021-06-25 11:42:02,713 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳ Reference year: 2018. Using nearest available year for GPW population data: 2020
2021-06-25 11:42:02,714 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳ Version v4.11
2021-06-25 11:42:02,807 - climada.util.coordinates - INFO - Setting region_id 972 points.

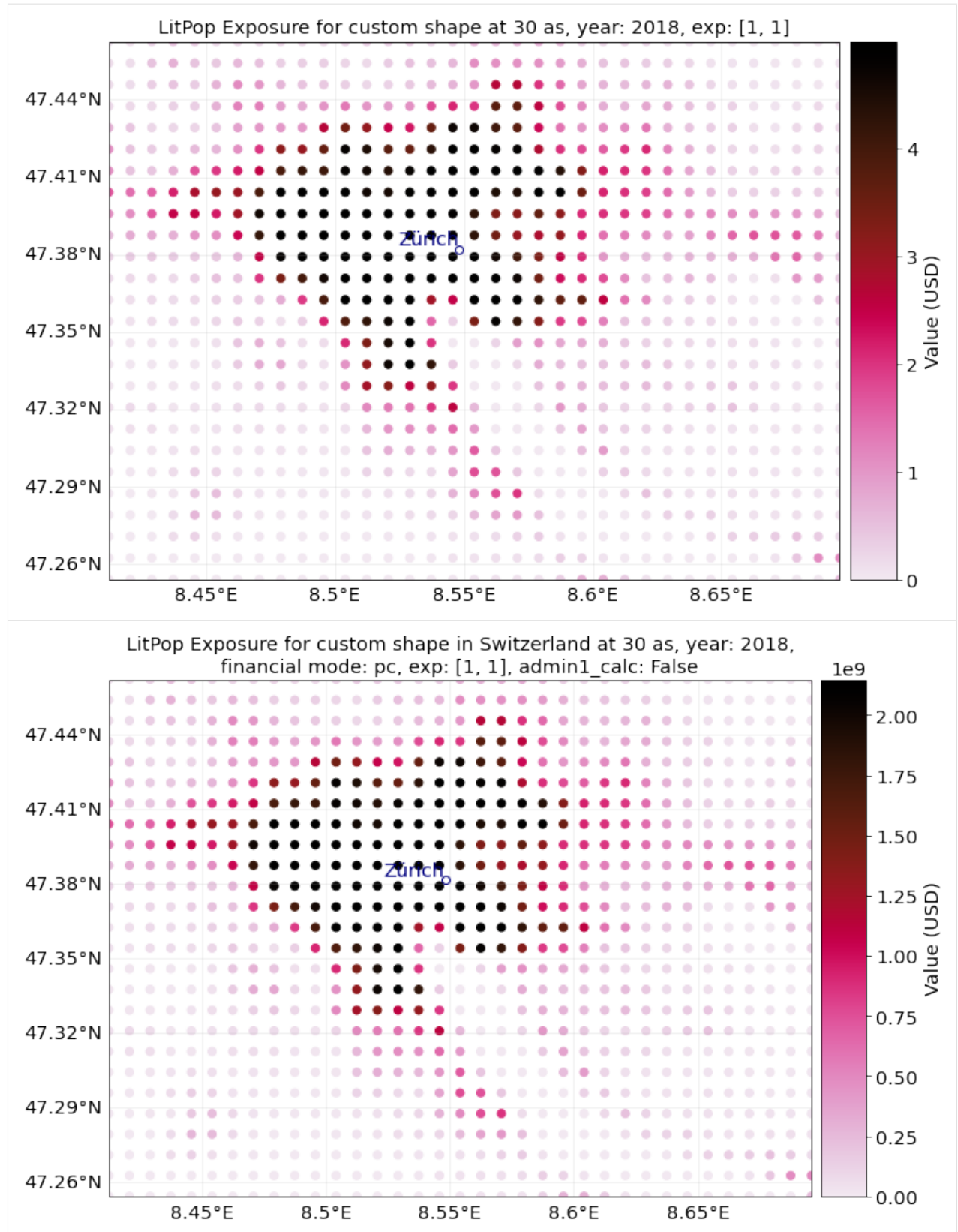
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
    return _prepare_from_string(" ".join(pjargs))
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1117:
↳ 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.

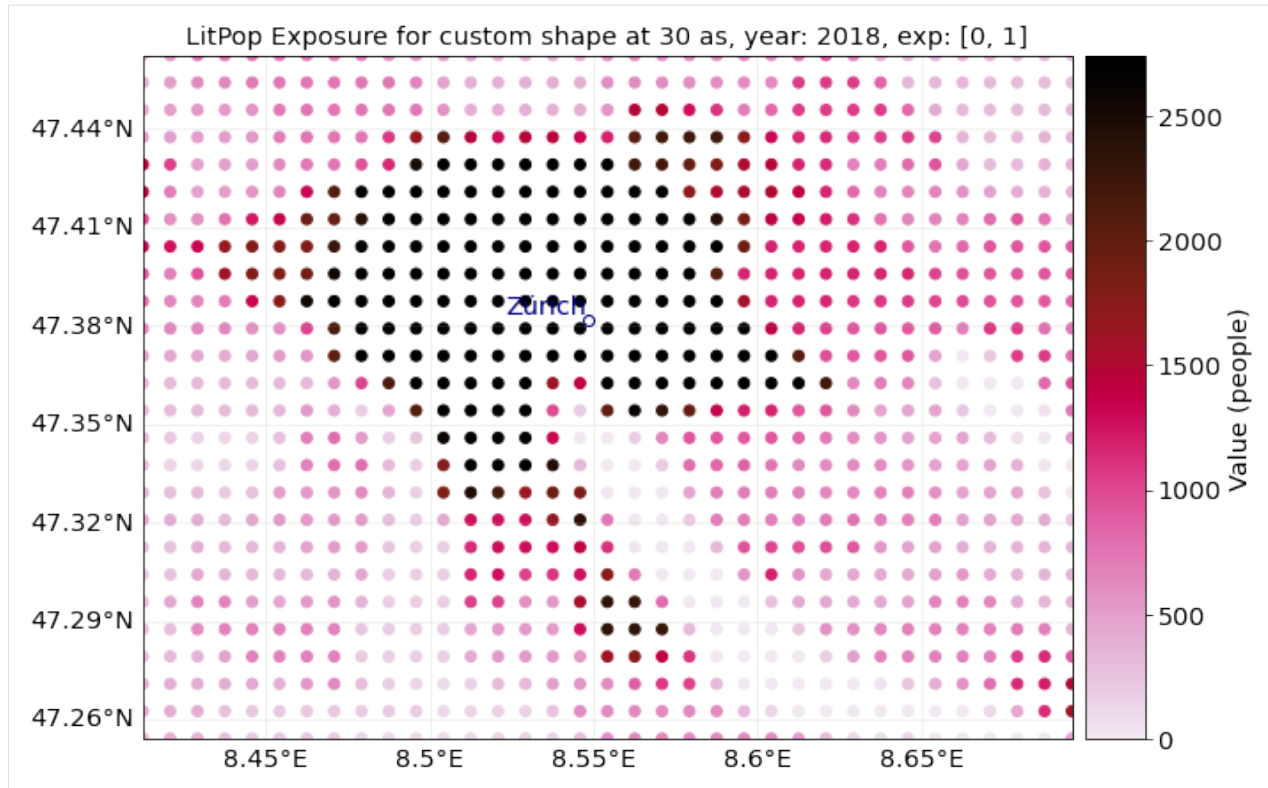
countries['area'] = countries.geometry.area

Runtime `set_population` : 1.33 sec.

[14]: 'Population exposure for a custom shape can be initiated directly via `set_population`
↳ without providing `total_value`'

```



Sub-national (admin-1) GDP as intermediate downscaling layer

In order to improve downscaling for countries with large regional differences within, a subnational breakdown of GDP can be used as an intermediate downscaling layer wherever available.

The sub-national (admin-1) GDP-breakdown needs to be added manually as a “.xls”-file to the folder `data/system/GSDP` in the CLIMADA-directory. Currently, such data is provided for more than 10 countries, including USA, India, and China.

The xls-file requires at least the following columns (with names specified in row 1): - `State_Province`: Names of admin-1 regions, i.e. states, cantons, provinces. Names need to match the naming of admin-1 shapes in the data used by the python package `cartopy.io` (c.f. `shapereader.natural_earth(name='admin_1_states_provinces')`) - `GSDP_ref`: value of sub-national GDP to be used (absolute or relative values) - `Postal`, optional: Alternative identifier of region, if names do not match with `cartopy`. Needs to correspond to the Postal-identifiers used in the `shapereader` of `cartopy.io`.

Please note that while admin1-GDP will per definition improve the downscaling of *GDP*, it might not necessarily improve the downscaling quality for other asset bases like produced capital (pc). ##### How To: The intermediate downscaling layer can be activated with the parameter `admin1_calc=True`.

```
[5]: # Initiate GDP-Entity for Switzerland, with and without admin1_calc:

ent_adm0 = LitPop()
ent_adm0.set_country('CHE', res_arcsec=120, fin_mode='gdp', admin1_calc=False)
ent_adm0.set_geometry_points()

ent_adm1 = LitPop()
ent_adm1.set_country('CHE', res_arcsec=120, fin_mode='gdp', admin1_calc=True)
```

(continues on next page)

(continued from previous page)

```

ent_adm0.check()
ent_adm1.check()
print('Done.')

2021-09-09 12:41:00,329 - climada.entity.exposures.litpop.litpop - INFO -
LitPop: Init Exposure for country: CHE (756)...

2021-09-09 12:41:00,331 - climada.entity.exposures.litpop.gpw_population - WARNING -
↳Reference year: 2018. Using nearest available year for GPW data: 2020
2021-09-09 12:41:00,337 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↳Version v4.11
2021-09-09 12:41:01,329 - climada.util.finance - INFO - GDP CHE 2018: 7.359e+11.
2021-09-09 12:41:01,366 - climada.entity.exposures.base - INFO - Hazard type not set in
↳impf_
2021-09-09 12:41:01,369 - climada.entity.exposures.base - INFO - category_id not set.
2021-09-09 12:41:01,371 - climada.entity.exposures.base - INFO - cover not set.
2021-09-09 12:41:01,372 - climada.entity.exposures.base - INFO - deductible not set.
2021-09-09 12:41:01,374 - climada.entity.exposures.base - INFO - centr_ not set.
2021-09-09 12:41:01,379 - climada.util.coordinates - INFO - Setting geometry points.
2021-09-09 12:41:02,670 - climada.util.finance - INFO - GDP CHE 2018: 7.359e+11.
2021-09-09 12:41:02,670 - climada.entity.exposures.litpop.litpop - INFO - Valais
2021-09-09 12:41:02,827 - climada.util.coordinates - INFO - Setting region_id 1248
↳points.

/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

2021-09-09 12:41:03,520 - climada.entity.exposures.litpop.litpop - INFO - Ticino
2021-09-09 12:41:03,662 - climada.util.coordinates - INFO - Setting region_id 576 points.

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

2021-09-09 12:41:04,362 - climada.entity.exposures.litpop.litpop - INFO - Graubünden

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

```

(continues on next page)

(continued from previous page)

```

countries['area'] = countries.geometry.area
2021-09-09 12:41:04,548 - climada.util.coordinates - INFO - Setting region_id 1485
↳points.

/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

2021-09-09 12:41:05,238 - climada.entity.exposures.litpop.litpop - INFO - Schaffhausen
2021-09-09 12:41:05,305 - climada.util.coordinates - INFO - Setting region_id 105 points.

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

2021-09-09 12:41:05,993 - climada.entity.exposures.litpop.litpop - INFO - Thurgau
2021-09-09 12:41:06,068 - climada.util.coordinates - INFO - Setting region_id 250 points.

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

2021-09-09 12:41:06,747 - climada.entity.exposures.litpop.litpop - INFO - Zürich
2021-09-09 12:41:06,846 - climada.util.coordinates - INFO - Setting region_id 304 points.

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6

```

(continues on next page)

(continued from previous page)

```

return _prepare_from_string(" ".join(pjargs))
2021-09-09 12:41:07,529 - climada.entity.exposures.litpop.litpop - INFO - Aargau
2021-09-09 12:41:07,630 - climada.util.coordinates - INFO - Setting region_id 315 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))
2021-09-09 12:41:08,320 - climada.entity.exposures.litpop.litpop - INFO - Basel-Stadt
2021-09-09 12:41:08,360 - climada.util.coordinates - INFO - Setting region_id 10 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))
2021-09-09 12:41:09,068 - climada.entity.exposures.litpop.litpop - INFO - Basel-
↳ Landschaft
2021-09-09 12:41:09,130 - climada.util.coordinates - INFO - Setting region_id 133 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))
2021-09-09 12:41:09,822 - climada.entity.exposures.litpop.litpop - INFO - Sankt Gallen
2021-09-09 12:41:09,945 - climada.util.coordinates - INFO - Setting region_id 540 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

```

(continues on next page)

(continued from previous page)

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

```

2021-09-09 12:41:10,633 - climada.entity.exposures.litpop.litpop - INFO - Solothurn
2021-09-09 12:41:10,721 - climada.util.coordinates - INFO - Setting region_id 273 points.

```

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

```

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

```

2021-09-09 12:41:11,396 - climada.entity.exposures.litpop.litpop - INFO - Jura
2021-09-09 12:41:11,473 - climada.util.coordinates - INFO - Setting region_id 220 points.

```

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

```

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

```

2021-09-09 12:41:12,140 - climada.entity.exposures.litpop.litpop - INFO - Genève
2021-09-09 12:41:12,200 - climada.util.coordinates - INFO - Setting region_id 60 points.

```

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

```

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6

```

(continues on next page)

(continued from previous page)

```

return _prepare_from_string(" ".join(pjargs))
2021-09-09 12:41:12,951 - climada.entity.exposures.litpop.litpop - INFO - Vaud
2021-09-09 12:41:13,100 - climada.util.coordinates - INFO - Setting region_id 888 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))
2021-09-09 12:41:13,798 - climada.entity.exposures.litpop.litpop - INFO - Neuchâtel
2021-09-09 12:41:13,875 - climada.util.coordinates - INFO - Setting region_id 200 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))
2021-09-09 12:41:14,563 - climada.entity.exposures.litpop.litpop - INFO - Bern
2021-09-09 12:41:14,750 - climada.util.coordinates - INFO - Setting region_id 1519
↳ points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))
2021-09-09 12:41:15,424 - climada.entity.exposures.litpop.litpop - INFO - Lucerne
2021-09-09 12:41:15,519 - climada.util.coordinates - INFO - Setting region_id 320 points.
/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

```

(continues on next page)

(continued from previous page)

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

```

2021-09-09 12:41:16,196 - climada.entity.exposures.litpop.litpop - INFO - Zug
2021-09-09 12:41:16,247 - climada.util.coordinates - INFO - Setting region_id 45 points.

```

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

```

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

```

2021-09-09 12:41:16,936 - climada.entity.exposures.litpop.litpop - INFO - Uri
2021-09-09 12:41:17,031 - climada.util.coordinates - INFO - Setting region_id 224 points.

```

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

```

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

```

2021-09-09 12:41:17,739 - climada.entity.exposures.litpop.litpop - INFO - Schwyz
2021-09-09 12:41:17,812 - climada.util.coordinates - INFO - Setting region_id 190 points.

```

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

```

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6

```

(continues on next page)

(continued from previous page)

```

return _prepare_from_string(" ".join(pjargs))

2021-09-09 12:41:18,504 - climada.entity.exposures.litpop.litpop - INFO - Glarus
2021-09-09 12:41:18,588 - climada.util.coordinates - INFO - Setting region_id 121 points.

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))

2021-09-09 12:41:19,268 - climada.entity.exposures.litpop.litpop - INFO - Nidwalden
2021-09-09 12:41:19,334 - climada.util.coordinates - INFO - Setting region_id 88 points.

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))

2021-09-09 12:41:20,015 - climada.entity.exposures.litpop.litpop - INFO - Fribourg
2021-09-09 12:41:20,125 - climada.util.coordinates - INFO - Setting region_id 323 points.

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳ 68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳ is the preferred initialization method. When making the change, be mindful of axis
↳ order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳ in-proj-6
return _prepare_from_string(" ".join(pjargs))

2021-09-09 12:41:20,810 - climada.entity.exposures.litpop.litpop - INFO - Obwalden
2021-09-09 12:41:20,867 - climada.util.coordinates - INFO - Setting region_id 98 points.

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

```

(continues on next page)

(continued from previous page)

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

```

2021-09-09 12:41:21,562 - climada.entity.exposures.litpop.litpop - INFO - Appenzell_
↳Ausserrhoden

```

```

2021-09-09 12:41:21,627 - climada.util.coordinates - INFO - Setting region_id 78 points.

```

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

```

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

```

2021-09-09 12:41:22,364 - climada.entity.exposures.litpop.litpop - INFO - Appenzell_
↳Innerrhoden

```

```

2021-09-09 12:41:22,427 - climada.util.coordinates - INFO - Setting region_id 63 points.

```

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳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.

```

```

countries['area'] = countries.geometry.area
/Users/eberenzs/anaconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳68: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>'
↳' is the preferred initialization method. When making the change, be mindful of axis_
↳order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-
↳in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

```

2021-09-09 12:41:23,163 - climada.entity.exposures.base - INFO - Hazard type not set in_
↳impf_

```

```

2021-09-09 12:41:23,163 - climada.entity.exposures.base - INFO - category_id not set.

```

```

2021-09-09 12:41:23,164 - climada.entity.exposures.base - INFO - cover not set.

```

```

2021-09-09 12:41:23,165 - climada.entity.exposures.base - INFO - deductible not set.

```

```

2021-09-09 12:41:23,168 - climada.entity.exposures.base - INFO - centr_ not set.

```

```

2021-09-09 12:41:23,173 - climada.entity.exposures.base - INFO - Hazard type not set in_
↳impf_

```

```

2021-09-09 12:41:23,174 - climada.entity.exposures.base - INFO - category_id not set.

```

```

2021-09-09 12:41:23,175 - climada.entity.exposures.base - INFO - cover not set.

```

```

2021-09-09 12:41:23,178 - climada.entity.exposures.base - INFO - deductible not set.

```

(continues on next page)

(continued from previous page)

```

2021-09-09 12:41:23,179 - climada.entity.exposures.base - INFO - centr_ not set.
2021-09-09 12:41:23,182 - climada.entity.exposures.base - INFO - Hazard type not set in
↳ impf_
2021-09-09 12:41:23,184 - climada.entity.exposures.base - INFO - category_id not set.
2021-09-09 12:41:23,185 - climada.entity.exposures.base - INFO - cover not set.
2021-09-09 12:41:23,188 - climada.entity.exposures.base - INFO - deductible not set.
2021-09-09 12:41:23,188 - climada.entity.exposures.base - INFO - centr_ not set.
Done.

```

```

/Users/eberenzs/Documents/Projects/climada_python/climada/util/coordinates.py:1129:
↳ 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.

```

```
countries['area'] = countries.geometry.area
```

```

[32]: # Plotting:
from matplotlib import colors
norm=colors.LogNorm(vmin=1e5, vmax=1e9) # setting range for the log-normal scale
markersize = 5
ent_adm0.plot_hexbin(buffer=.3, norm=norm, linewidth=markersize)
ent_adm1.plot_hexbin(buffer=.3, norm=norm, linewidth=markersize)

print('admin-0: First figure')
print('admin-1: Second figure')
'''Do you spot the small differences in Graubünden (eastern Switzerland)?'''

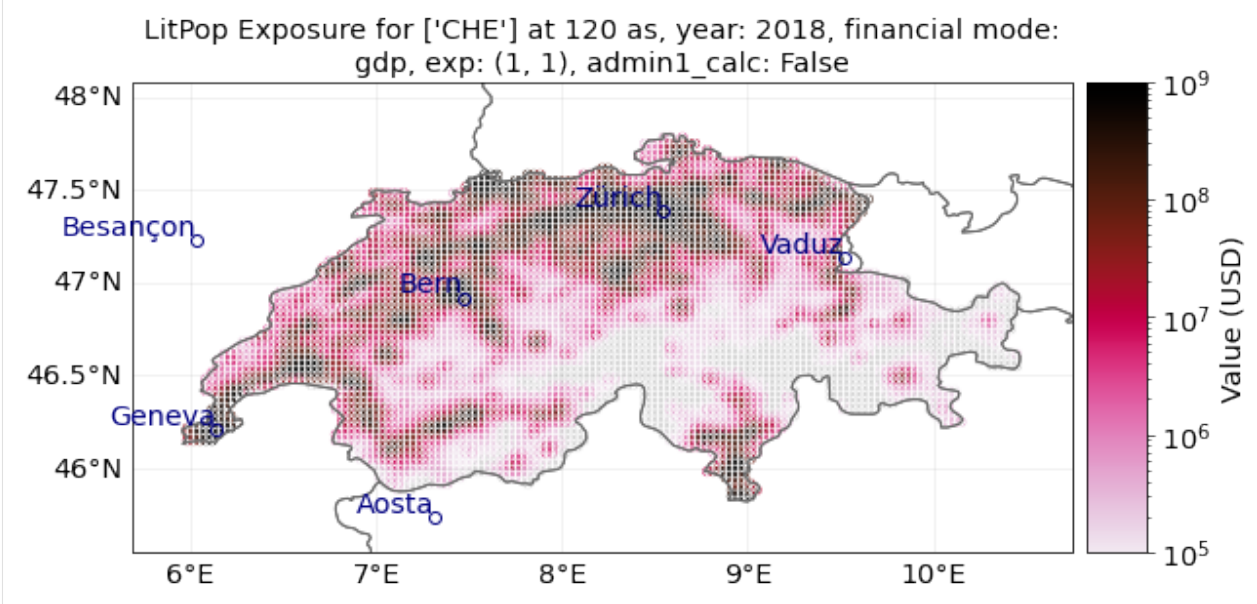
```

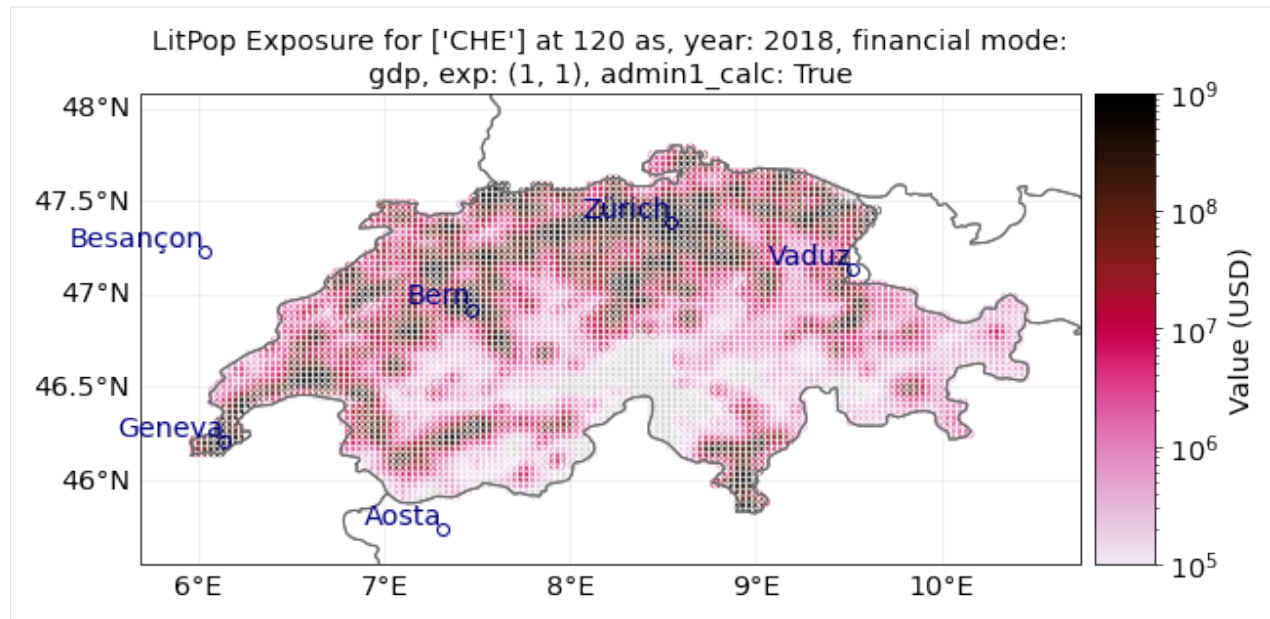
```

admin-0: First figure
admin-1: Second figure

```

```
[32]: 'Do you spot the small differences in Graubünden (eastern Switzerland)?'
```





[]:

5.4 Impact Functions

5.4.1 What is an impact function?

An impact function relates the percentage of damage in the exposure to the hazard intensity, also commonly referred to as a “vulnerability curves” in the modelling community. Every hazard and exposure types are characterized by an impact function.

5.4.2 What is the difference between `ImpactFunc` and `ImpactFuncSet`?

An `ImpactFunc` is a class for a single impact function. E.g. a function that relates the percentage of damage of a reinforced concrete building (exposure) to the wind speed of a tropical cyclone (hazard intensity).

An `ImpactFuncSet` class is a container that contains multiple `ImpactFunc`. For instance, there are 100 `ImpactFunc` represent 100 types of buildings exposed to tropical cyclone’s wind damage. These 100 `ImpactFunc` are all gathered in an `ImpactFuncSet`.

5.4.3 What does an `ImpactFunc` look like in CLIMADA?

The `ImpactFunc` class requires users to define the following attributes.

Mandatory attributes	Data Type	Description
haz_type	(str)	Hazard type acronym (e.g. 'TC')
id	(int or str)	Unique id of the impact function. Exposures of the same type will refer to the same impact function id
name	(str)	Name of the impact function
intensity	(np.array)	Intensity values
intensity_unit	(str)	Unit of the intensity
mdd	(np.array)	Mean damage (impact) degree for each intensity (numbers in [0,1])
paa	(np.array)	Percentage of affected assets (exposures) for each intensity (numbers in [0,1])

Users may use `ImpactFunc.check()` to check that the attributes have been set correctly. The mean damage ratio `mdr` ($mdr = mdd * paa$) is calculated by the method `ImpactFunc.calc_mdr()`.

5.4.4 What does an `ImpactFuncSet` look like in CLIMADA?

The `ImpactFuncSet` class contains all the `ImpactFunc` classes. Users are not required to define any attributes in `ImpactFuncSet`.

To add an `ImpactFunc` into an `ImpactFuncSet`, simply use the method `ImpactFuncSet.append(ImpactFunc)`. If the users only has one impact function, they should generate an `ImpactFuncSet` that contains one impact function. `ImpactFuncSet` is to be used in the *impact calculation*.

Tag stores information about the data. E.g. the original file name of the impact functions and descriptions.

At-tributes	Data Type	Description
tag	Tag	Information about the source data
_data	(dict)	Contains <code>ImpactFunc</code> classes. Not supposed to be directly accessed. Use the class methods instead.

5.4.5 Structure of the tutorial

****Part 1:**** Defining `ImpactFunc` from your own data

****Part 2:**** Loading `ImpactFunc` from CLIMADA in-built impact functions

****Part 3:**** Add `ImpactFunc` into the container `ImpactFuncSet`

****Part 4:**** Read and write `ImpactFuncSet` into Excel sheets

****Part 5:**** Loading `ImpactFuncSet` from CLIMADA in-built impact functions

Part 1: Defining `ImpactFunc` from your own data

The essential attributes are listed in the table above. The following example shows you how to define an `ImpactFunc` from scratch, and using the method `ImpactFunc.calc_mdr()` to calculate the mean damage ratio.

5.4.6 Generate a dummy impact function from scratch.

Here we generate an impact function with random dummy data for illustrative reasons. Assuming this impact function is a function that relates building damage to tropical cyclone (TC) wind, with an arbitrary id 3.

```
[1]: import numpy as np
from climada.entity import ImpactFunc

# We initialise a dummy ImpactFunc for tropical cyclone wind damage to building.
# Giving the ImpactFunc an arbitrary id 3.
imp_fun = ImpactFunc()
imp_fun.haz_type = 'TC'
imp_fun.id = 3
imp_fun.name = 'TC building damage'
# provide unit of the hazard intensity
imp_fun.intensity_unit = 'm/s'
# provide values for the hazard intensity, mdd, and paa
imp_fun.intensity = np.linspace(0, 100, num=15)
imp_fun.mdd = np.concatenate((np.array([0]),np.sort(np.random.rand(14)))), axis=0)
imp_fun.paa = np.concatenate((np.array([0]),np.sort(np.random.rand(14)))), axis=0)
# check if the all the attributes are set correctly
imp_fun.check()

[2]: # Calculate the mdr at hazard intensity 18.7 m/s
print('Mean damage ratio at intensity 18.7 m/s: ', imp_fun.calc_mdr(18.7))

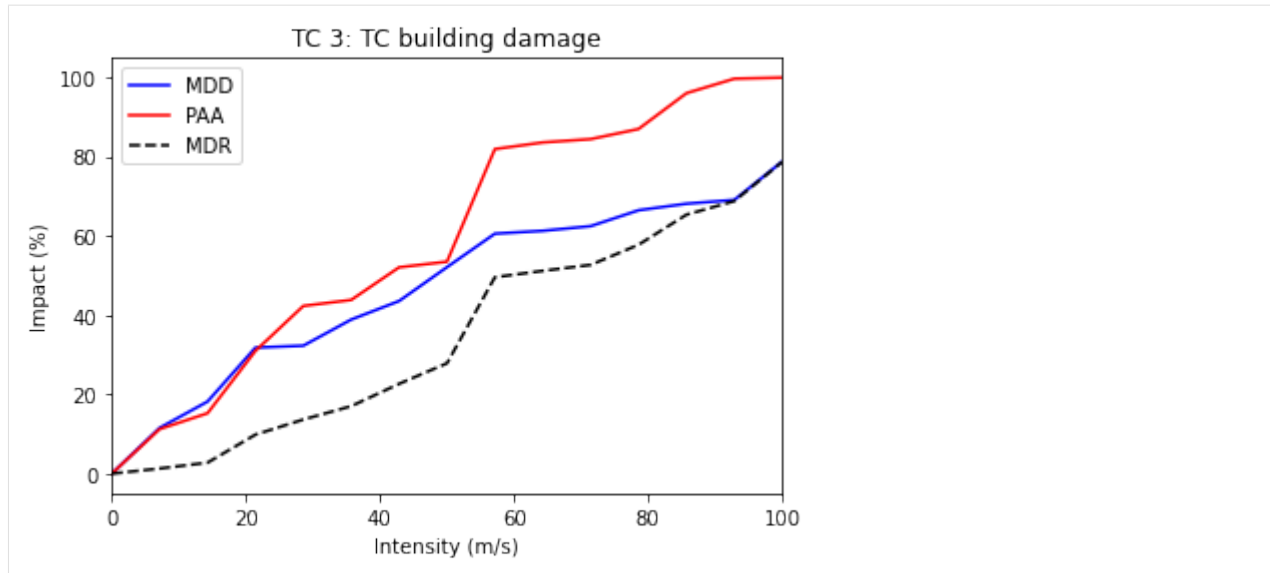
Mean damage ratio at intensity 18.7 m/s:  0.06631877847780823
```

5.4.7 Visualise the Impact function

The method `plot()` uses the `matplotlib`'s `axes plot` function to visualise the impact function. It returns a figure and axes, which can be modified by users.

```
[3]: # plot impact function
imp_fun.plot()

[3]: <AxesSubplot:title={'center':'TC 3: TC building damage'}, xlabel='Intensity (m/s)',
↪ ylabel='Impact (%)'>
```



Part 2: Loading impact functions from CLIMADA in-built impact functions

In CLIMADA there is several defined impact functions that users can directly load and use them. However, users should be aware of the applications of the impact functions to types of assets, reading the background references of the impact functions are strongly recommended. Currently available perils include [tropical cyclones](#), [river floods](#), [European windstorm](#), [crop yield](#), and [drought](#). Continuous updates of perils are available. Here we use the impact function of tropical cyclones as an example.

5.4.8 Loading CLIMADA in-built impact function for tropical cyclones

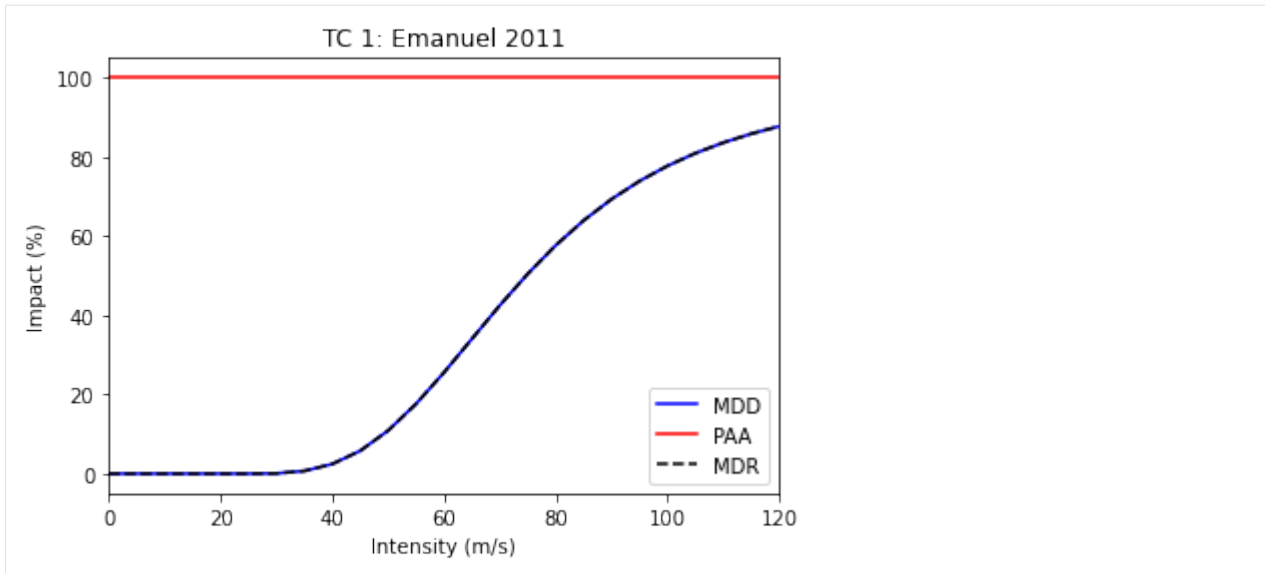
`ImpfTropCyclone` is a derivated class of `ImpactFunc`. This in-built impact function estimates the insured property damages by tropical cyclone wind in USA, forllowing the reference paper [Emanuel \(2011\)](#).

To generate the this impact function, method `set_emanuel_usa()` is used.

```
[4]: from climada.entity import ImpfTropCyclone

# Here we generate the impact function for TC damage using the formula of Emanuel 2011
impFunc_emanuel_usa = ImpfTropCyclone()
impFunc_emanuel_usa.set_emanuel_usa()
# plot the impact function
impFunc_emanuel_usa.plot()

[4]: <AxesSubplot:title={'center':'TC 1: Emanuel 2011'}, xlabel='Intensity (m/s)', ylabel=
     ↪ 'Impact (%)'>
```

Part 3: Add ImpactFunc into the container ImpactFuncSet

ImpactFuncSet is a container of multiple ImpactFunc, it is part of the arguments in `impact.calc()` (see [the impact tutorial](#)).

Here we generate 2 arbitrary impact functions and add them into an ImpactFuncSet class. To add them into the container, simply use the method `ImpactFuncSet.append(ImpactFunc)`.

```
[5]: import numpy as np
import matplotlib.pyplot as plt
from climada.entity import ImpactFunc, ImpactFuncSet

# generate the 1st arbitrary impact function
imp_fun_1 = ImpactFunc()
imp_fun_1.haz_type = 'TC'
imp_fun_1.id = 1
imp_fun_1.name = 'TC Default Damage Function'
imp_fun_1.intensity_unit = 'm/s'
imp_fun_1.intensity = np.linspace(0, 100, num=10)
imp_fun_1.mdd = np.concatenate((np.array([0]), np.sort(np.random.rand(9)))), axis=0)
imp_fun_1.paa = np.concatenate((np.array([0]), np.sort(np.random.rand(9)))), axis=0)
imp_fun_1.check()

# generate the 2nd arbitrary impact function
imp_fun_3 = ImpactFunc()
imp_fun_3.haz_type = 'TC'
imp_fun_3.id = 3
imp_fun_3.name = 'TC Building Damage'
imp_fun_3.intensity_unit = 'm/s'
imp_fun_3.intensity = np.linspace(0, 100, num=15)
imp_fun_3.mdd = np.concatenate((np.array([0]), np.sort(np.random.rand(14)))), axis=0)
imp_fun_3.paa = np.concatenate((np.array([0]), np.sort(np.random.rand(14)))), axis=0)
imp_fun_1.check()

# add the 2 impact functions into ImpactFuncSet
```

(continues on next page)

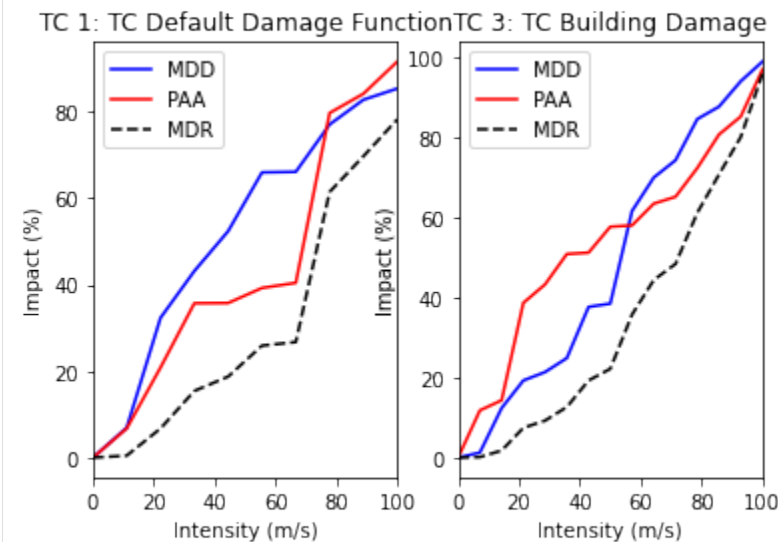
(continued from previous page)

```
imp_fun_set = ImpactFuncSet()
imp_fun_set.append(imp_fun_1)
imp_fun_set.append(imp_fun_3)
```

5.4.9 Plotting all the impact functions in an ImpactFuncSet

The method `plot()` in `ImpactFuncSet` also uses the `matplotlib`'s `axes plot function` to visualise the impact functions, returning a figure with all the subplots of impact functions. Users may modify these plots.

```
[6]: # plotting all the impact functions in impf_set
axes = imp_fun_set.plot()
```

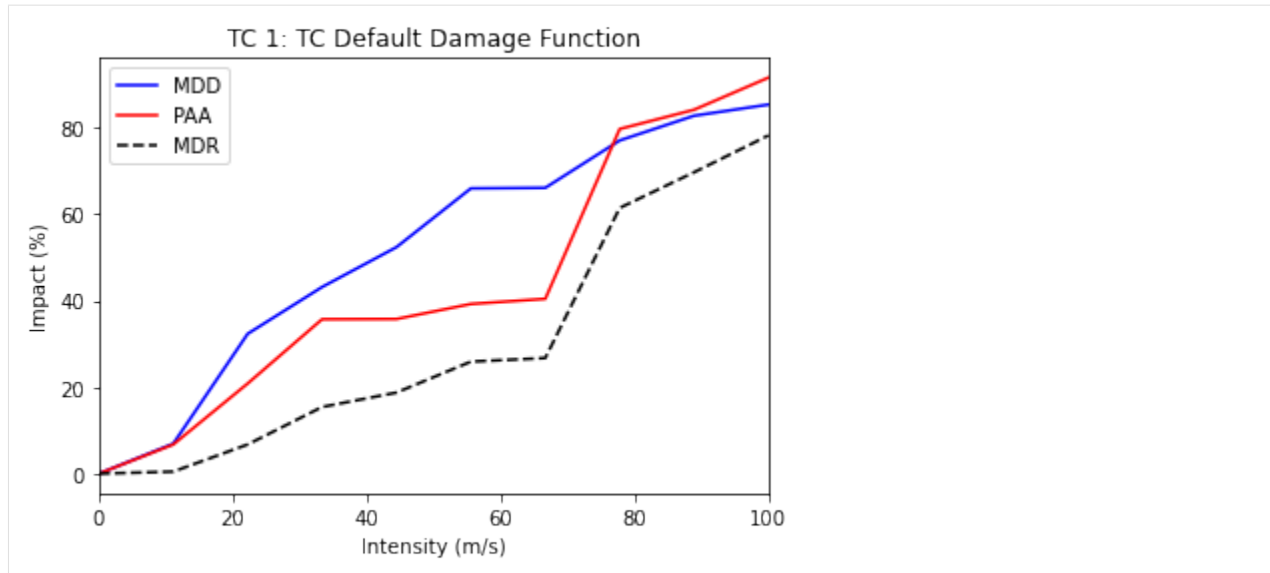


5.4.10 Retrieving an impact function from the ImpactFuncSet

User may want to retrieve a particular impact function from `ImpactFuncSet`. Using the method `get_func(haz_type, id)`, it returns an `ImpactFunc` class of the desired impact function. Below is an example of extracting the TC impact function with id 1, and using `plot()` to visualise the function.

```
[7]: # extract the TC impact function with id 1
impf_tc_1 = imp_fun_set.get_func('TC', 1)
# plot the impact function
impf_tc_1.plot()
```

```
[7]: <AxesSubplot:title={'center':'TC 1: TC Default Damage Function'}, xlabel='Intensity (m/s)
→', ylabel='Impact (%)'>
```



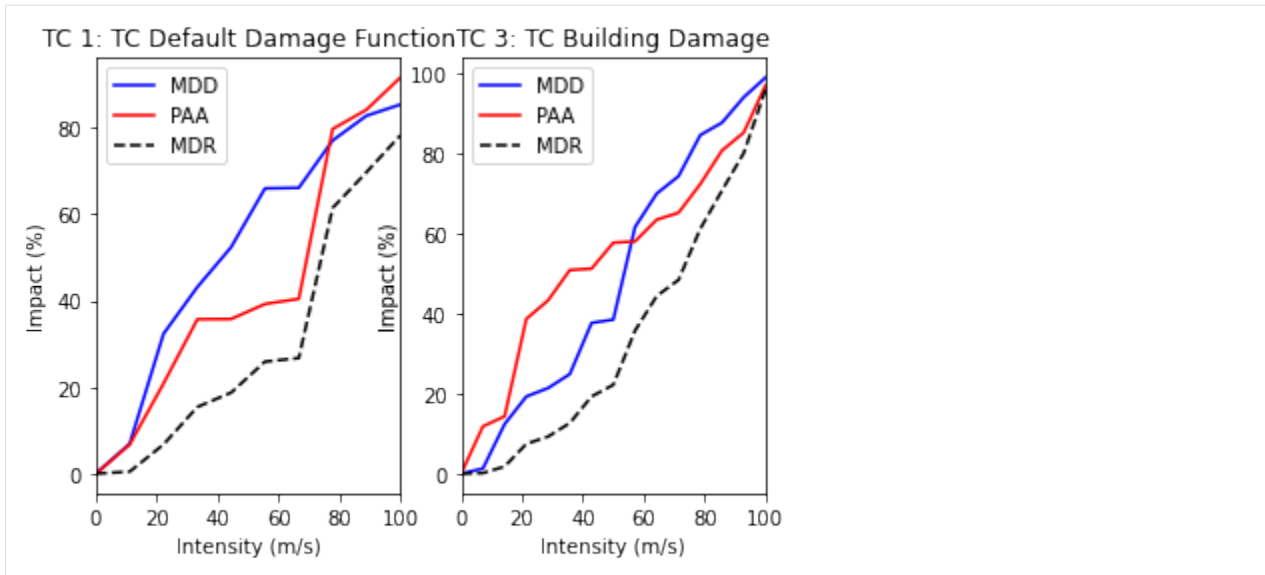
5.4.11 Removing an impact function from the ImpactFunctionSet

If there is an unwanted impact function from the `ImpactFuncSet`, we may remove it using the method `remove_func(haz_type, id)` to remove it from the set.

For example, in the previous generated impact function set `imp_fun_set` contains an unwanted TC impact function with id 3, we might thus would like to remove that from the set.

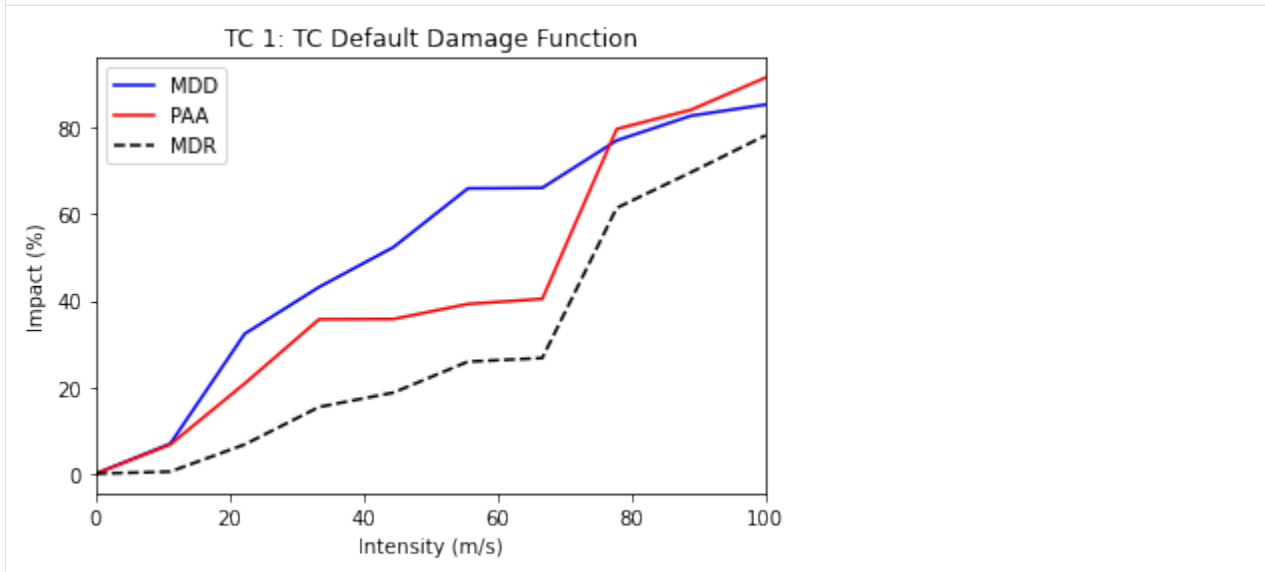
```
[8]: # first plotting all the impact functions in the impact function set to see what is in_
     ↪ there:
     imp_fun_set.plot()

[8]: array([<AxesSubplot:title={'center':'TC 1: TC Default Damage Function'}, xlabel=
     ↪ 'Intensity (m/s)', ylabel='Impact (%)'>,
          <AxesSubplot:title={'center':'TC 3: TC Building Damage'}, xlabel='Intensity (m/s)
     ↪ ', ylabel='Impact (%)'>],
      dtype=object)
```



```
[9]: # removing the TC impact function with id 3
imp_fun_set.remove_func('TC', 3)
# plot all the remaining impact functions in imp_fun_set
imp_fun_set.plot()
```

```
[9]: <AxesSubplot:title={'center':'TC 1: TC Default Damage Function'}, xlabel='Intensity (m/s)'
      ↪, ylabel='Impact (%)'>
```



Part 4: Read and write ImpactFuncSet into Excel sheets

Users may load impact functions to an `ImpactFuncSet` class from an excel sheets, or to write the `ImpactFuncSet` into an excel. This section will give an example of how to do it.

5.4.12 Reading impact functions from an Excel file

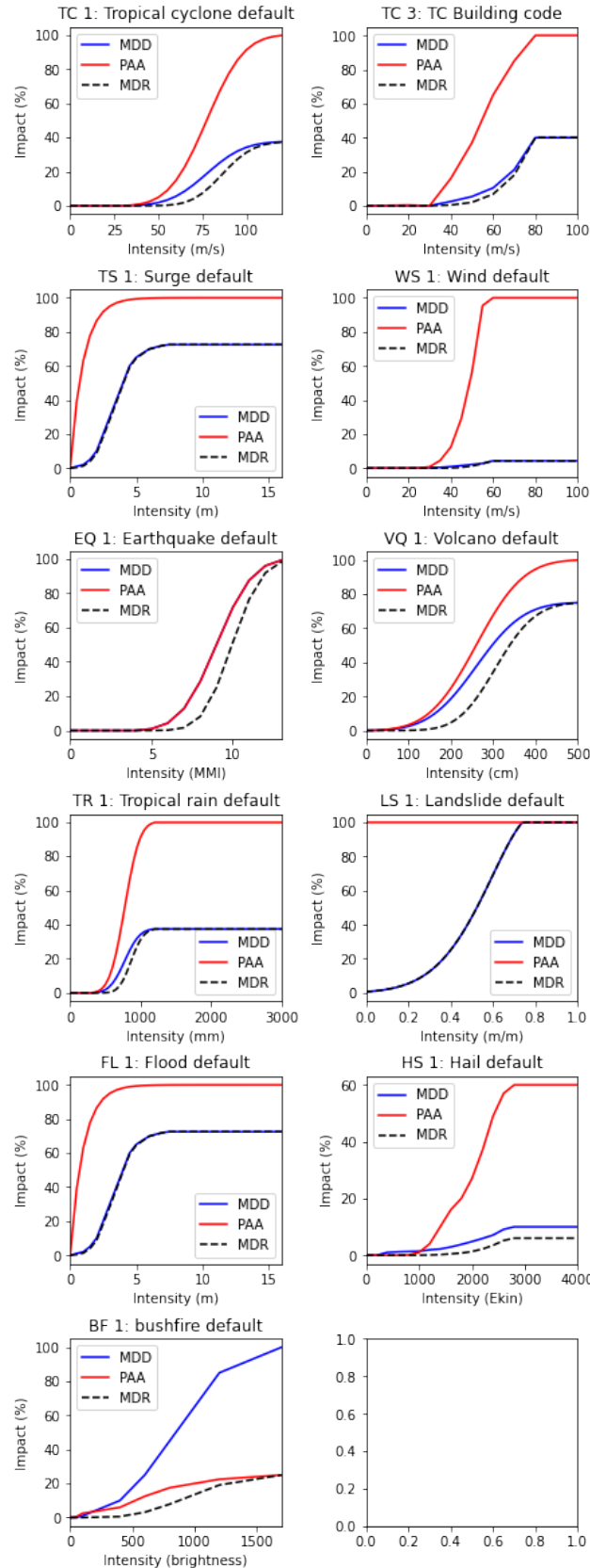
Impact functions defined in an excel file following the template provided in sheet `impact_functions` of `climada_python/data/system/entity_template.xlsx` can be ingested directly using the method `read_excel()`.

```
[10]: from climada.entity import ImpactFuncSet
      from climada.util import ENT_TEMPLATE_XLS
      import matplotlib.pyplot as plt

      # provide absolute path of the input excel file
      file_name = ENT_TEMPLATE_XLS
      # fill ImpactFuncSet from Excel file
      imp_set_xlsx = ImpactFuncSet()
      imp_set_xlsx.read_excel(file_name)

      # plot all the impact functions from the ImpactFuncSet
      print('Read file:', imp_set_xlsx.tag.file_name)
      imp_set_xlsx.plot()
      # adjust the plots
      plt.tight_layout()
      plt.subplots_adjust(right=1., top=4., hspace=0.4, wspace=0.4)

      Read file: C:\Users\me\climada\data\entity_template.xlsx
```



5.4.13 Write impact functions

Users may write the impact functions in Excel format using `write_excel()` method.

```
[11]: from climada.entity import ImpactFuncSet
      from climada.util import ENT_TEMPLATE_XLS

      # provide absolute path of the output excel file
      file_name_save = ENT_TEMPLATE_XLS

      # write imp_set_xlsx into an excel file
      imp_set_xlsx.write_excel('tutorial_impf_set.xlsx')
```

5.4.14 Alternative saveing format

Alternatively, users may also save the impact functions into `pickle format`, using CLIMADA in-built function `save()`.

```
[12]: from climada.util.save import save

      # this generates a results folder in the current path and stores the output there
      save('tutorial_impf_set.p', imp_set_xlsx)

2021-04-30 14:47:45,320 - climada.util.save - INFO - Written file ./results/tutorial_
↳ impf_set.p
```

Part 5: Loading ImpactFuncSet from CLIMADA in-built impact functions

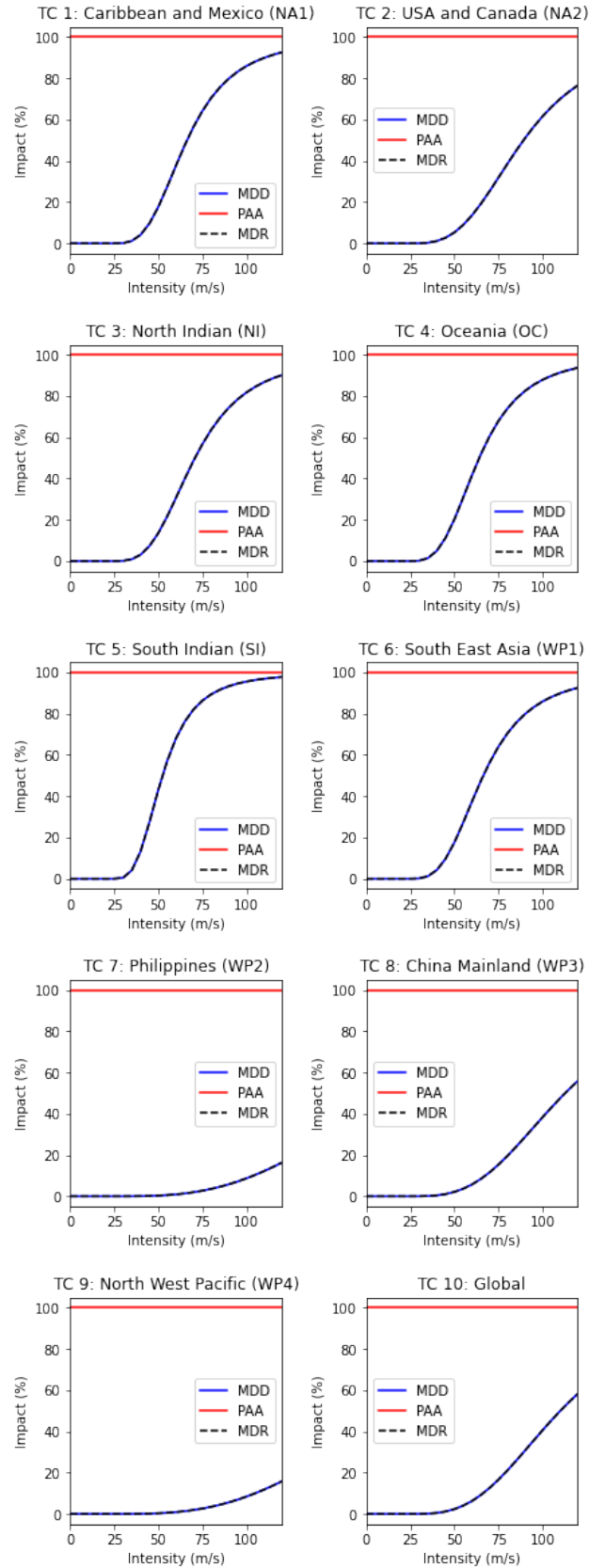
Similar to *Part 3*, some of the impact functions are available as `ImpactFuncSet` classes. Users may load them from the CLIMADA modules.

Here we use the example of the calibrated impact functions of TC wind damages per region to property damages, following the reference [Eberenz et al. \(2020\)](#). Method `set_calibrated_regional_ImpfSet()` returns a set of default calibrated impact functions for TC for different regions.

```
[13]: from climada.entity.impact_funcs.trop_cyclone import ImpfSetTropCyclone
      import matplotlib.pyplot as plt

      # generate the default calibrated TC impact functions for different regions
      imp_fun_set_TC = ImpfSetTropCyclone()
      imp_fun_set_TC.set_calibrated_regional_ImpfSet()

      # plot all the impact functions
      imp_fun_set_TC.plot()
      # adjust the plots
      plt.tight_layout()
      plt.subplots_adjust(right=1., top=4., hspace=0.4, wspace=0.4)
```



5.5 DiscRates class

Discount rates are used to calculate the net present value of any future or past value. They are thus used to compare amounts paid (costs) and received (benefits) in different years. A project is economically viable (attractive), if the net present value of benefits exceeds the net present value of costs - a const-benefit ratio < 1 .

There are several important implications that come along with discount rates. Namely, that higher discount rates lead to smaller net present values of future impacts (costs). As a consequence of that, climate action and mitigation measures can be postponed. In the literature higher discount rates are typically justified by the expectation of continued exponential growth of the economy. The most widely used interest rate in climate change economics is 1.4% as proposed by the Stern Review (2006). Neoliberal economists around Nordhaus (2007) claim that rates should be higher, around 4.3%. Environmental economists argue that future costs shouldn't be discounted at all. This is especially true for non-monetary variables such as ecosystems or human lives, where no price tag should be applied out of ethical reasons. This discussion has a long history, reaching back to the 18th century: "Some things have a price, or relative worth, while other things have a dignity, or inner worth" (Kant, 1785).

This class contains the discount rates for every year and discounts given values. Its attributes are:

- tag (Tag): information about the source data
- years (np.array): years
- rates (np.array): discount rates for each year (between 0 and 1)

```
[2]: from climada.entity import DiscRates
help(DiscRates)

2020-10-19 09:43:26,307 - climada - DEBUG - Loading default config file: /Users/sam/
↳ Documents/Python/climada_python/climada/conf/defaults.conf
Help on class DiscRates in module climada.entity.disc_rates.base:

class DiscRates(builtins.object)
|   Defines discount rates and basic methods. Loads from
|   files with format defined in FILE_EXT.
|
|   Attributes:
|       tag (Tag): information about the source data
|       years (np.array): years
|       rates (np.array): discount rates for each year (between 0 and 1)
|
|   Methods defined here:
|
|   __init__(self)
|       Empty initialization.
|
|   Examples:
|       Fill discount rates with values and check consistency data:
|
|       >>> disc_rates = DiscRates()
|       >>> disc_rates.years = np.array([2000, 2001])
|       >>> disc_rates.rates = np.array([0.02, 0.02])
|       >>> disc_rates.check()
|
|       Read discount rates from year_2050.mat and checks consistency data.
|
|       >>> disc_rates = DiscRates(ENT_TEMPLATE_XLS)
```

(continues on next page)

(continued from previous page)

```

| append(self, disc_rates)
|     Check and append discount rates to current DiscRates. Overwrite
|     discount rate if same year.
|
|     Parameters:
|         disc_rates (DiscRates): DiscRates instance to append
|
|     Raises:
|         ValueError
|
| check(self)
|     Check attributes consistency.
|
|     Raises:
|         ValueError
|
| clear(self)
|     Reinitialize attributes.
|
| net_present_value(self, ini_year, end_year, val_years)
|     Compute net present value between present year and future year.
|
|     Parameters:
|         ini_year (float): initial year
|         end_year (float): end year
|         val_years (np.array): cash flow at each year btw ini_year and
|             end_year (both included)
|     Returns:
|         float
|
| plot(self, axis=None, **kwargs)
|     Plot discount rates per year.
|
|     Parameters:
|         axis (matplotlib.axes._subplots.AxesSubplot, optional): axis to use
|         kwargs (optional): arguments for plot matplotlib function, e.g. marker='x'
|
|     Returns:
|         matplotlib.axes._subplots.AxesSubplot
|
| read_excel(self, file_name, description='', var_names={'sheet_name': 'discount',
| → 'col_name': {'year': 'year', 'disc': 'discount_rate'}})
|     Read excel file following template and store variables.
|
|     Parameters:
|         file_name (str): absolute file name
|         description (str, optional): description of the data
|         var_names (dict, optional): name of the variables in the file
|
| read_mat(self, file_name, description='', var_names={'sup_field_name': 'entity',
| → 'field_name': 'discount', 'var_name': {'year': 'year', 'disc': 'discount_rate'}})

```

(continues on next page)

(continued from previous page)

```

|     Read MATLAB file generated with previous MATLAB CLIMADA version.
|
|     Parameters:
|         file_name (str): absolute file name
|         description (str, optional): description of the data
|         var_names (dict, optional): name of the variables in the file
|
|     select(self, year_range)
|         Select discount rates in given years.
|
|     Parameters:
|         year_range (np.array): continuous sequence of selected years.
|
|     Returns:
|         DiscRates
|
|     write_excel(self, file_name, var_names={'sheet_name': 'discount', 'col_name': {'year
|     → ': 'year', 'disc': 'discount_rate'}})
|         Write excel file following template.
|
|     Parameters:
|         file_name (str): absolute file name to write
|         var_names (dict, optional): name of the variables in the file
|
|     -----
|     Data descriptors defined here:
|
|     __dict__
|         dictionary for instance variables (if defined)
|
|     __weakref__
|         list of weak references to the object (if defined)

```

An example of use - we define discount rates and apply them on a coastal protection scheme which initially costs 100 mn. USD plus 75'000 USD maintenance each year, starting after 10 years. Net present value of the project can be calculated as displayed:

```

[8]: %matplotlib inline
import numpy as np
from climada.entity import DiscRates

# define discount rates
disc = DiscRates()
disc.years = np.arange(1950, 2100)
disc.rates = np.ones(disc.years.size) * 0.014
disc.rates[51:55] = 0.025
disc.rates[95:120] = 0.035
disc.check()
disc.plot()

# Compute net present value between present year and future year.

```

(continues on next page)

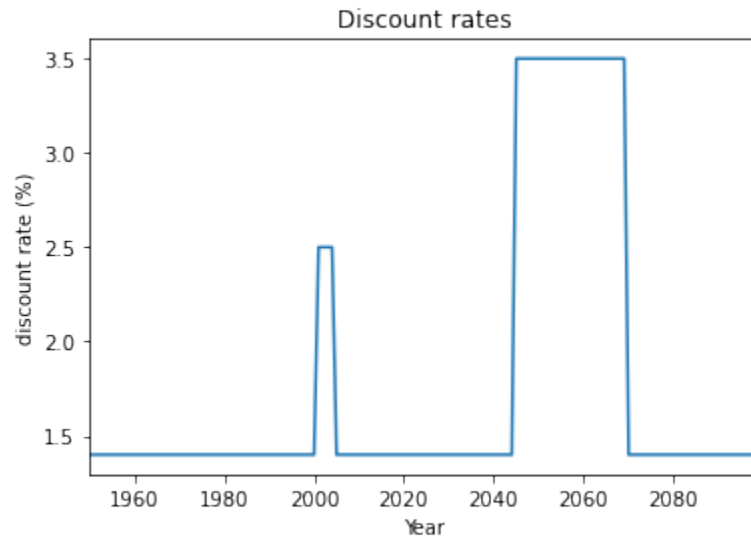
(continued from previous page)

```

ini_year = 2019
end_year = 2050
val_years = np.zeros(end_year-ini_year+1)
val_years[0] = 1000000000 # initial investment
val_years[10:] = 75000 # maintenance from 10th year
npv = disc.net_present_value(ini_year, end_year, val_years)
print('net present value: {:.5e}'.format(npv))

```

```
net present value: 1.01231e+08
```



5.5.1 Read discount rates of an Excel file

Discount rates defined in an excel file following the template provided in sheet discount of `climada_python/data/system/entity_template.xlsx` can be ingested directly using the method `read_excel()`.

```

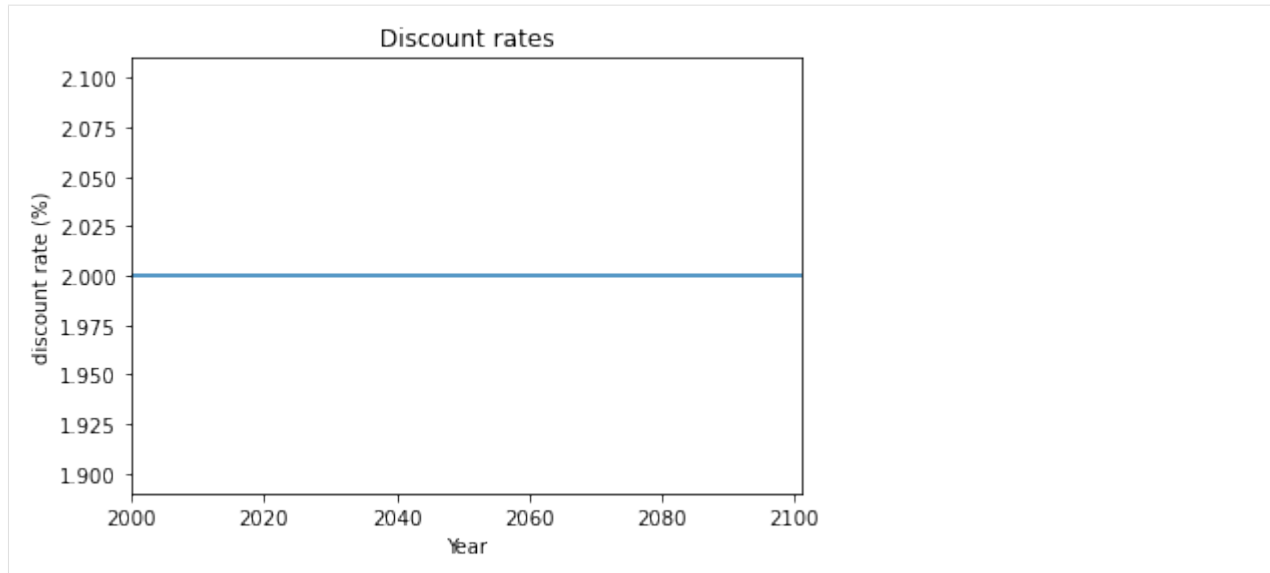
[4]: from climada.entity import DiscRates
      from climada.util import ENT_TEMPLATE_XLS

      # Fill DataFrame from Excel file
      file_name = ENT_TEMPLATE_XLS # provide absolute path of the excel file
      disc = DiscRates()
      disc.read_excel(file_name)
      print('Read file:', disc.tag.file_name)
      disc.plot()

```

```
Read file: /Users/sam/Documents/Python/climada_python/data/system/entity_template.xlsx
```

```
[4]: <matplotlib.axes._subplots.AxesSubplot at 0x121bf2518>
```



5.5.2 Write discount rates

Discount rates defined in an excel file following the template provided in sheet discount of `climada_python/data/system/entity_template.xlsx` can be ingested directly using the method `read_excel()`.

```
[6]: from climada.entity import DiscRates
      from climada.util import ENT_TEMPLATE_XLS

      # Fill DataFrame from Excel file
      file_name = ENT_TEMPLATE_XLS # provide absolute path of the excel file
      disc = DiscRates()
      disc.read_excel(file_name)

      # write file
      disc.write_excel('results/tutorial_disc.xlsx')
```

Pickle can always be used as well:

```
[7]: from climada.util.save import save
      # this generates a results folder in the current path and stores the output there
      save('tutorial_disc.p', disc)

2020-10-19 09:46:38,974 - climada.util.save - INFO - Written file /Users/sam/Documents/
Python/climada_python/doc/tutorial/results/tutorial_disc.p
```

```
[ ]:
```

5.6 How to use polygons or lines as exposure

Exposure in CLIMADA are usually represented as individual points or a raster of points. See [Exposures](#) tutorial to learn how to fill and use exposures. In this tutorial we show you how to use CLIMADA Impf you have your exposure in the form of shapes/polygons or in the form of lines.

The approach follows three steps: 1. transform your polygon or line in a set of points 2. do the impact calculation in CLIMADA with that set of points 3. transform the calculated Impact back to your polygon or line

5.6.1 Polygons

Polygons or shapes are a common geographical representation of countries, states etc. as for example in NaturalEarth. Here we want to show you how to deal with exposure information as polygons.

Lets assume we have the following data given. The polygons of the admin-1 regions of the netherlands and an exposure value each. We want to know the Impact of Lothar on each admin-1 region.

```
[1]: from cartopy.io import shapereader
from climada.entity.exposures.black_marble import country_iso_geom

# open the file containing the Netherlands admin-1 polygons
shp_file = shapereader.natural_earth(resolution='10m',
                                     category='cultural',
                                     name='admin_0_countries')
shp_file = shapereader.Reader(shp_file)

# extract the NL polygons
prov_names = {'Netherlands': ['Groningen', 'Drenthe',
                              'Overijssel', 'Gelderland',
                              'Limburg', 'Zeeland',
                              'Noord-Brabant', 'Zuid-Holland',
                              'Noord-Holland', 'Friesland',
                              'Flevoland', 'Utrecht']}

polygon_Netherlands, polygons_prov_NL = country_iso_geom(prov_names,
                                                         shp_file)

# assign a value to each admin-1 area (assumption 100'000 USD per inhabitant)
population_prov_NL = {'Drenthe':493449, 'Flevoland':422202,
                     'Friesland':649988, 'Gelderland':2084478,
                     'Groningen':585881, 'Limburg':1118223,
                     'Noord-Brabant':2562566, 'Noord-Holland':2877909,
                     'Overijssel':1162215, 'Zuid-Holland':3705625,
                     'Utrecht':1353596, 'Zeeland':383689}
value_prov_NL = {n: 100000 * population_prov_NL[n] for n in population_prov_NL.keys()}
```

Assume a unImpform distribution of values within your polygons

This helps you in the case you have a given total exposure value per polygon and we assume this value is distributed evenly within the polygon.

We can now perform the three steps for this example:

```
[2]: import numpy as np
      from pandas import DataFrame
      from climada.entity import Exposures
      from climada.util.coordinates import coord_on_land

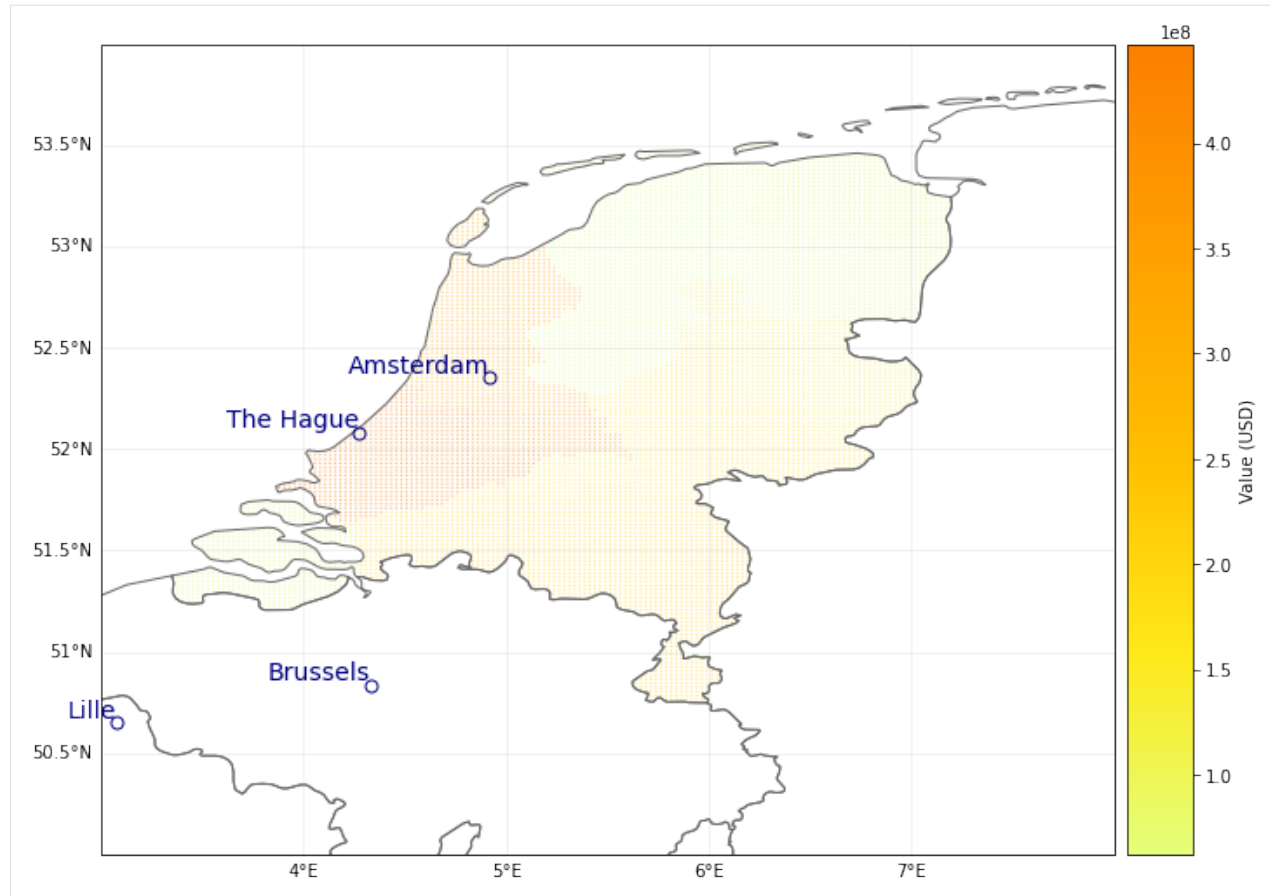
      ### 1. transform your polygon or line in a set of points
      # create exposure with points
      exp_df = DataFrame()
      n_exp = 200*200
      lat, lon = np.mgrid[50 : 54 : complex(0, np.sqrt(n_exp)),
                          3 : 8 : complex(0, np.sqrt(n_exp))]
      exp_df['latitude'] = lat.flatten() # provide latitude
      exp_df['longitude'] = lon.flatten() # provide longitude
      exp_df['impf_WS'] = np.ones(n_exp, int) # provide impact functions

      # now we assign each point a province and a value, Impf the points are within one of the
      ↪ polygons defined above
      exp_df['province'] = ''
      exp_df['value'] = np.ones((exp_df.shape[0],))*np.nan
      for prov_name_i, prob_polygon_i in zip(prov_names['Netherlands'], polygons_prov_NL['NLD
      ↪ ']):
          in_geom = coord_on_land(lat=exp_df['latitude'],
                                  lon=exp_df['longitude'],
                                  land_geom=prob_polygon_i)
          np.put(exp_df['province'].values, np.where(in_geom)[0], prov_name_i)
          np.put(exp_df['value'].values, np.where(in_geom)[0], value_prov_NL[prov_name_i]/
          ↪ sum(in_geom))

      exp_df = Exposures(exp_df)
      exp_df.set_geometry_points() # set geometry attribute (shapely Points) from GeoDataFrame
      ↪ from latitude and longitude
      exp_df.check()
      exp_df.plot_hexbin()

      2021-06-04 15:19:31,794 - climada.util.coordinates - INFO - Setting geometry points.
      2021-06-04 15:19:33,203 - climada.entity.exposures.base - INFO - category_id not set.
      2021-06-04 15:19:33,204 - climada.entity.exposures.base - INFO - cover not set.
      2021-06-04 15:19:33,205 - climada.entity.exposures.base - INFO - deductible not set.
      2021-06-04 15:19:33,206 - climada.entity.exposures.base - INFO - region_id not set.
      2021-06-04 15:19:33,207 - climada.entity.exposures.base - INFO - centr_ not set.
```

[2]: <GeoAxesSubplot:>



```
[3]: from climada.hazard.storm_europe import StormEurope
from climada.util.constants import WS_DEMO_NC
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.entity.impact_funcs import ImpactFuncSet
from climada.engine import Impact
```

```
### 2. do the impact calculation in CLIMADA with that set of points
# define hazard
```

```
storms = StormEurope()
storms.read_footprints(WS_DEMO_NC, description='test_description')
```

```
# define impact function
```

```
impact_func = ImpfStormEurope()
impact_func.set_welker()
impact_function_set = ImpactFuncSet()
impact_function_set.append(impact_func)
```

```
# calculate hazard
```

```
impact_NL = Impact()
impact_NL.calc(exp_df, impact_function_set, storms, save_mat=True)
impact_NL.plot_hexbin_impact_exposure()
```

```
2021-06-04 15:19:49,921 - climada.hazard.storm_europe - INFO - Constructing centroids.
↳ from C:\Users\me\climada\demo\data\fp_lothar_crop-test.nc
2021-06-04 15:19:49,971 - climada.hazard.centroids.centri - INFO - Convert centroids to
↳ GeoSeries of Point shapes.
```

```
/Users/zeliestalhanske/python_projects/climada_python/climada/hazard/centroids/centr.py:
↳611: UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are likely
↳incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
↳this operation.
```

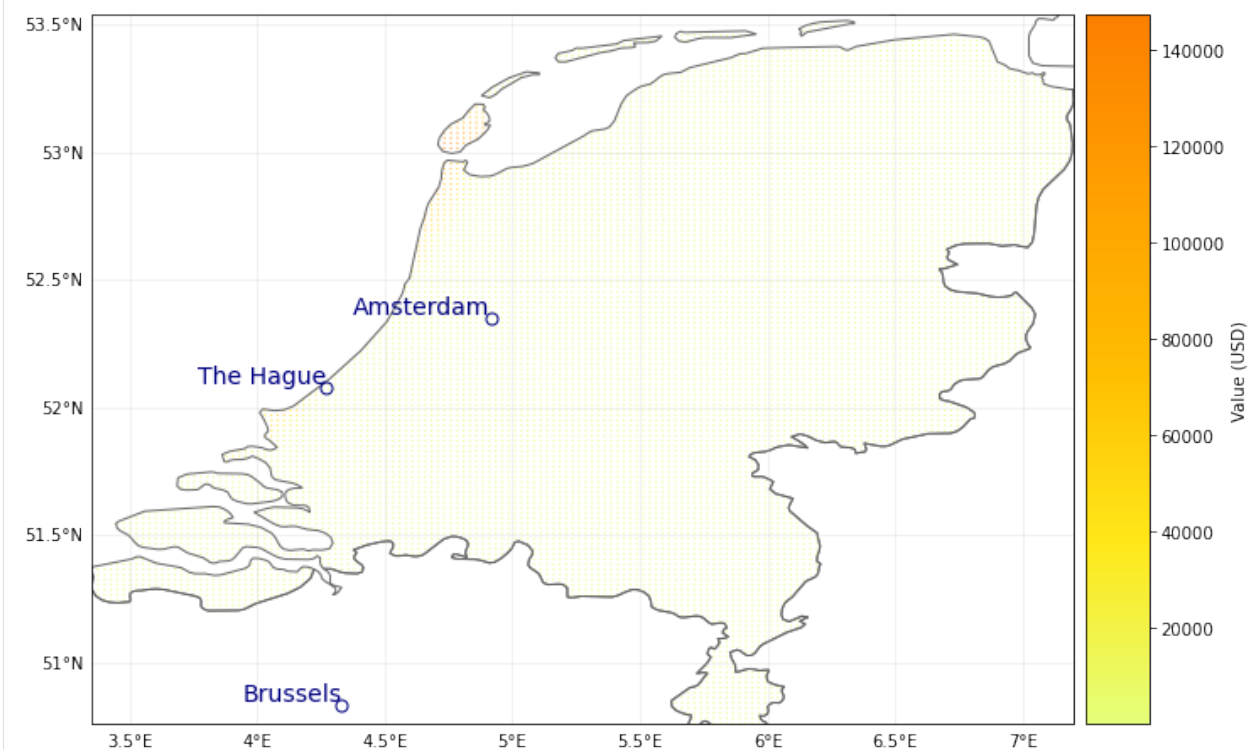
```
xy_pixels = self.geometry.buffer(res / 2).envelope
```

```
2021-06-04 15:19:53,574 - climada.hazard.storm_europe - INFO - Commencing to iterate
↳over netCDF files.
```

```
2021-06-04 15:19:53,650 - climada.entity.exposures.base - INFO - Matching 40000
↳exposures with 9944 centroids.
```

```
2021-06-04 15:19:55,490 - climada.engine.impact - INFO - Calculating damage for 9660
↳assets (>0) and 2 events.
```

[3]: <GeoAxesSubplot:>



[4]: `import pandas as pd`

```
### 3. transform the calculated Impact back to your polygon or line
impact_at_province_raw = pd.DataFrame(np.mean(impact_NL.imp_mat.todense().transpose(),
↳axis=1),
                                     index=exp_df.gdf['province'])
impact_at_province = impact_at_province_raw.groupby(impact_at_province_raw.index).sum()
print(impact_at_province)
```

```
0
province
Drenthe      0.000000e+00
Flevoland    2.716078e+05
```

(continues on next page)

(continued from previous page)

Friesland	7.782136e+05
Gelderland	4.056456e+05
Groningen	1.150089e+05
Limburg	2.559739e+05
Noord-Brabant	7.391625e+05
Noord-Holland	5.908293e+06
Overijssel	1.588620e+05
Utrecht	4.846288e+05
Zeeland	4.069767e+05
Zuid-Holland	2.893966e+06

Use the LitPop module to disaggregate your exposure values

Instead of a unImpform distribution, another geographical distribution can be chosen to disaggregate within the value within each polygon. We show here the example of *LitPop*. The same three steps apply:

```
[5]: import numpy as np
from climada.entity import LitPop
from climada.util.coordinates import coord_on_land

### 1. transform your polygon or line in a set of points
# create exposure with points
exp_df_lp = LitPop()
exp_df_lp.set_country('Netherlands', res_arcsec = 60, fin_mode = 'none')
exp_df_lp.gdf['impf_WS'] = np.ones(exp_df_lp.gdf.shape[0], int) # provide impact_
↳ functions

# now we assign each point a province and a value, Impf the points are within one of the_
↳ polygons defined above
exp_df_lp.gdf['province'] = ''
for prov_name_i, prob_polygon_i in zip(prov_names['Netherlands'], polygons_prov_NL['NLD
↳ '']):
    in_geom = coord_on_land(lat=exp_df_lp.gdf['latitude'],
                            lon=exp_df_lp.gdf['longitude'],
                            land_geom=prob_polygon_i)
    np.put(exp_df_lp.gdf['province'].values, np.where(in_geom)[0], prov_name_i)
    exp_df_lp.gdf['value'][np.where(in_geom)[0]] = \
        exp_df_lp.gdf['value'][np.where(in_geom)[0]] * value_prov_NL[prov_name_
↳ i]/sum(exp_df_lp.gdf['value'][np.where(in_geom)[0]])
exp_df_lp.gdf = exp_df_lp.gdf.drop(np.where(exp_df_lp.gdf['province']=='')[0]) #drop_
↳ caribbean islands for this example
exp_df_lp.set_geometry_points()
exp_df_lp.check()
exp_df_lp.plot_hexbin()

2021-06-04 15:20:33,993 - climada.entity.exposures.litpop - INFO - Generating LitPop_
↳ data at a resolution of 60 arcsec.
2021-06-04 15:20:47,826 - climada.entity.exposures.gpw_import - INFO - Reference year:_
↳ 2016. Using nearest available year for GPW population data: 2015
2021-06-04 15:20:47,836 - climada.entity.exposures.gpw_import - INFO - GPW Version v4.11
```

(continues on next page)

(continued from previous page)

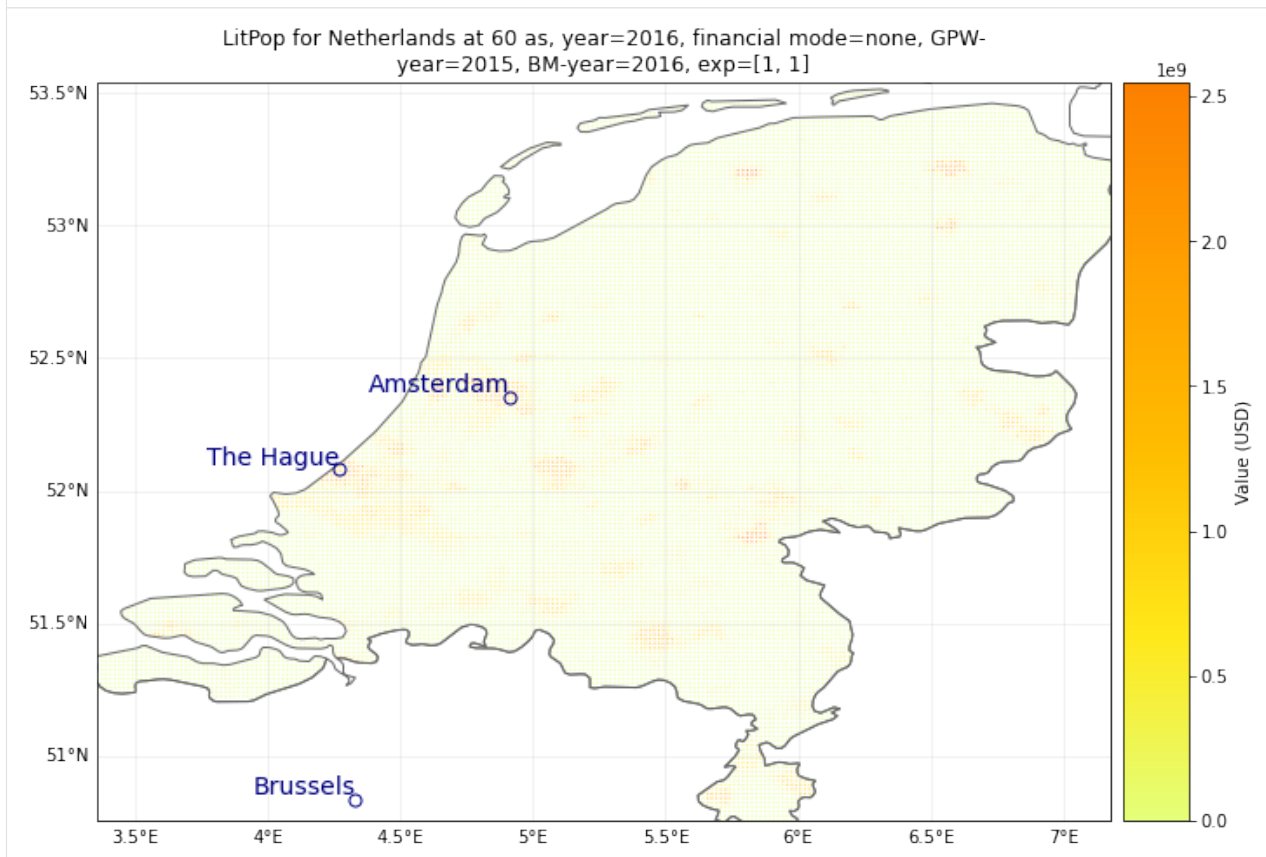
```

2021-06-04 15:21:21,206 - climada.entity.exposures.litpop - INFO - fin_mode=None --> no_
↳downscaling; admin1_calc is ignored
2021-06-04 15:21:21,219 - climada.entity.exposures.base - INFO - Hazard type not set in_
↳impf_
2021-06-04 15:21:21,220 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-04 15:21:21,222 - climada.entity.exposures.base - INFO - cover not set.
2021-06-04 15:21:21,223 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-04 15:21:21,225 - climada.entity.exposures.base - INFO - geometry not set.
2021-06-04 15:21:21,227 - climada.entity.exposures.base - INFO - centr_ not set.
2021-06-04 15:21:21,238 - climada.entity.exposures.base - INFO - Hazard type not set in_
↳impf_
2021-06-04 15:21:21,240 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-04 15:21:21,242 - climada.entity.exposures.base - INFO - cover not set.
2021-06-04 15:21:21,245 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-04 15:21:21,248 - climada.entity.exposures.base - INFO - geometry not set.
2021-06-04 15:21:21,251 - climada.entity.exposures.base - INFO - centr_ not set.
2021-06-04 15:21:22,064 - climada.util.coordinates - INFO - Setting geometry points.
2021-06-04 15:21:22,738 - climada.entity.exposures.base - INFO - Hazard type not set in_
↳impf_
2021-06-04 15:21:22,739 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-04 15:21:22,741 - climada.entity.exposures.base - INFO - cover not set.
2021-06-04 15:21:22,742 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-04 15:21:22,743 - climada.entity.exposures.base - INFO - centr_ not set.

```

[5]: <GeoAxesSubplot:title={'center': 'LitPop for Netherlands at 60 as, year=2016, financial_

↳mode=None, GPW-\nyear=2015, BM-year=2016, exp=[1, 1]'}>



```
[6]: from climada.hazard.storm_europe import StormEurope
from climada.util.constants import WS_DEMO_NC
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.entity.impact_funcs import ImpactFuncSet
from climada.engine import Impact

### 2. do the impact calculation in CLIMADA with that set of points
# define hazard
storms = StormEurope()
storms.read_footprints(WS_DEMO_NC, description='test_description')
# define impact function
impact_func = ImpfStormEurope()
impact_func.set_welker()
impact_function_set = ImpactFuncSet()
impact_function_set.append(impact_func)
# calculate hazard
impact_NL = Impact()
impact_NL.calc(exp_df_lp, impact_function_set, storms, save_mat=True)
impact_NL.plot_hexbin_impact_exposure()
```

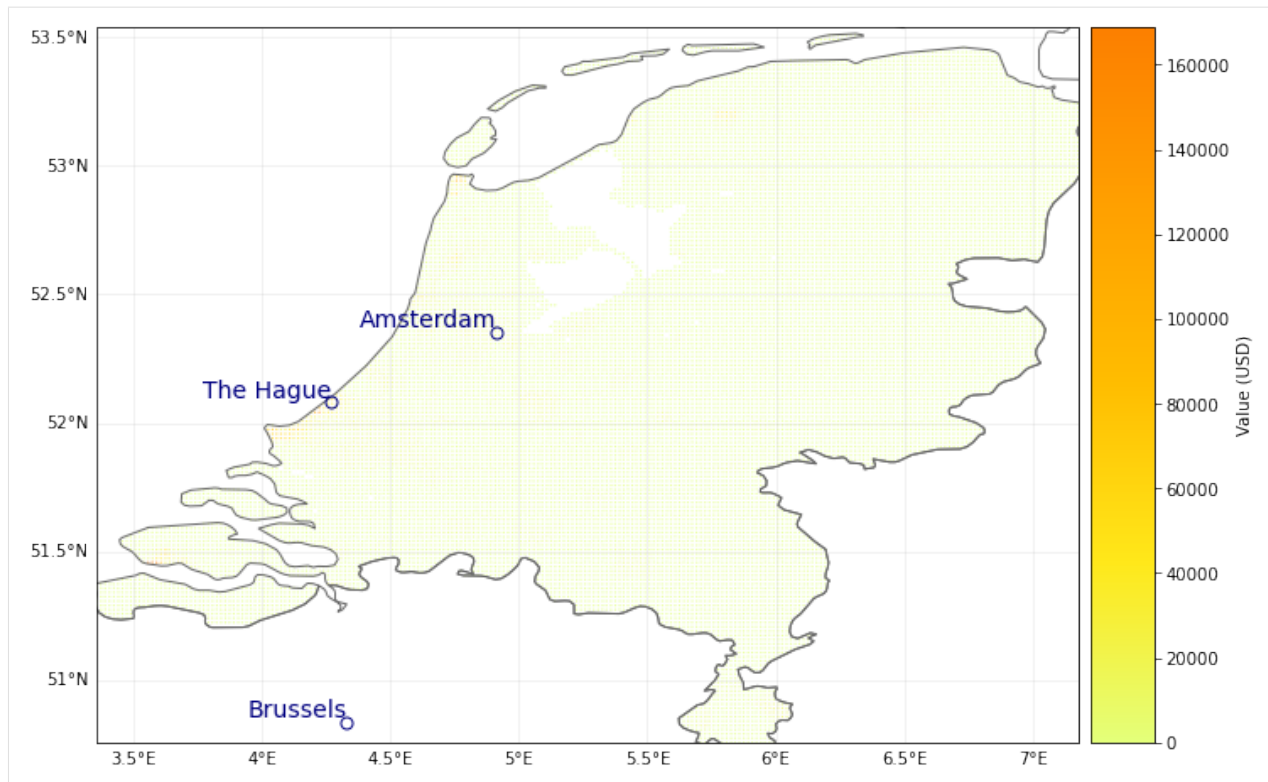
```
2021-06-04 15:21:53,619 - climada.hazard.storm_europe - INFO - Constructing centroids.
↳ from C:\Users\me\climada\demo\data\fp_lothar_crop-test.nc
2021-06-04 15:21:53,680 - climada.hazard.centroids.centri - INFO - Convert centroids to
↳ GeoSeries of Point shapes.
```

```
/Users/zeliestalhanske/python_projects/climada_python/climada/hazard/centroids/centri.py:
↳ 611: UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are likely
↳ incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
↳ this operation.
```

```
xy_pixels = self.geometry.buffer(res / 2).envelope
```

```
2021-06-04 15:21:56,137 - climada.hazard.storm_europe - INFO - Commencing to iterate
↳ over netCDF files.
2021-06-04 15:21:56,204 - climada.entity.exposures.base - INFO - Matching 17576
↳ exposures with 9944 centroids.
2021-06-04 15:21:56,877 - climada.engine.impact - INFO - Calculating damage for 16834
↳ assets (>0) and 2 events.
```

```
[6]: <GeoAxesSubplot:>
```



```
[7]: import pandas as pd
```

```
### 3. transform the calculated Impact back to your polygon or line
impact_at_province_raw = pd.DataFrame(np.mean(impact_NL.imp_mat.todense().transpose(),
axis=1),
index=exp_df_lp.gdf['province'])
impact_at_province_lp = impact_at_province_raw.groupby(impact_at_province_raw.index).
sum()
print(impact_at_province_lp)
```

```
0
province
Drenthe      6.397671e+04
Flevoland    1.666099e+05
Friesland    4.602394e+05
Gelderland   3.284452e+05
Groningen    1.563104e+05
Limburg      3.730663e+05
Noord-Brabant 6.247242e+05
Noord-Holland 1.819026e+06
Overijssel   1.074240e+05
Utrecht      4.484917e+05
Zeeland      7.632003e+05
Zuid-Holland 2.898284e+06
```

Comparison of both modelled impacts:

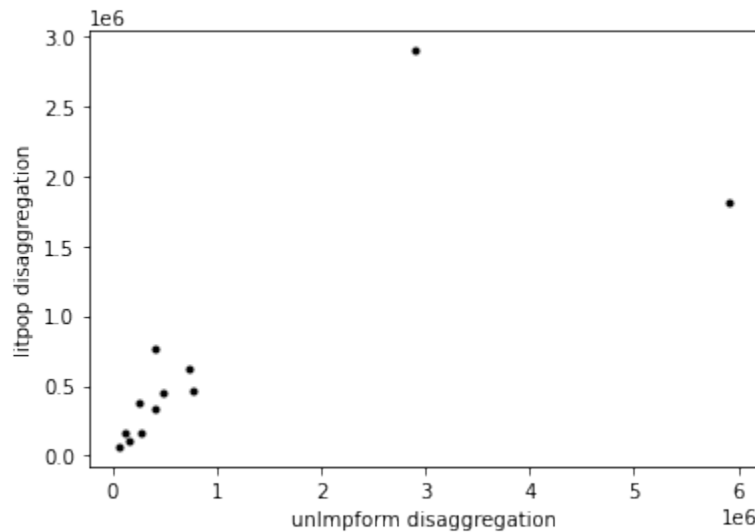
```
[8]: from matplotlib import pyplot as plt
```

(continues on next page)

(continued from previous page)

```
plt.plot(impact_at_province[impact_at_province.index!=''], impact_at_province_lp, '.k')
plt.xlabel('unImpform disaggregation')
plt.ylabel('litpop disaggregation')
```

```
[8]: Text(0, 0.5, 'litpop disaggregation')
```



Further statistical analysis of hazard on polygon level

imagine that you need access to the hazard centroids in order to provide some statistical analysis on the province level

```
[9]: %%time
# this provides the wind speed value for each event at the corresponding exposure
import scipy

exp_df_lp.gdf[:5]
l1,l2,vals = scipy.sparse.find(storms.intensity)
exp_df_lp.gdf['wind_0']=0; exp_df_lp.gdf['wind_1']=0 # provide columns for both events
for evt,idx,val in zip(l1,l2,vals):
    if evt==0:
        exp_df_lp.gdf.loc[exp_df_lp.gdf.index[exp_df_lp.gdf['centr_WS']==idx], 'wind_0']
        ↪ =val
    else:
        exp_df_lp.gdf.loc[exp_df_lp.gdf.index[exp_df_lp.gdf['centr_WS']==idx], 'wind_1']
        ↪ =val
```

Wall time: 13.9 s

```
[10]: # now you can perform additional statistical analysis and aggregate it to the province_
        ↪ level
import pandas as pd
import geopandas as gpd

exp_province_raw = exp_df_lp.copy()
```

(continues on next page)

(continued from previous page)

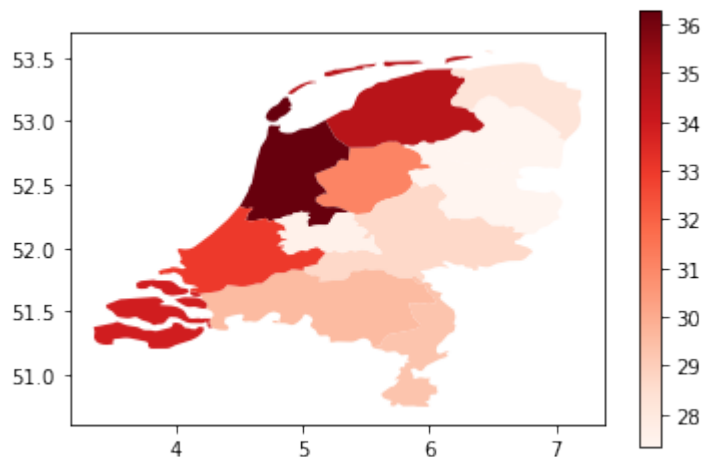
```

def f(x): # define function for statistical aggregation with pandas
    d = {}
    d['value'] = x['value'].sum()
    d['wind_0'] = x['wind_0'].max()
    d['wind_1'] = x['wind_1'].mean()
    # one could also be interested in centroid of max wind with respect to province
    # d['centr_WS'] = x.loc[x.index[x['wind_0'].max()], 'centr_WS']
    return pd.Series(d, index=['value', 'wind_0', 'wind_1'])

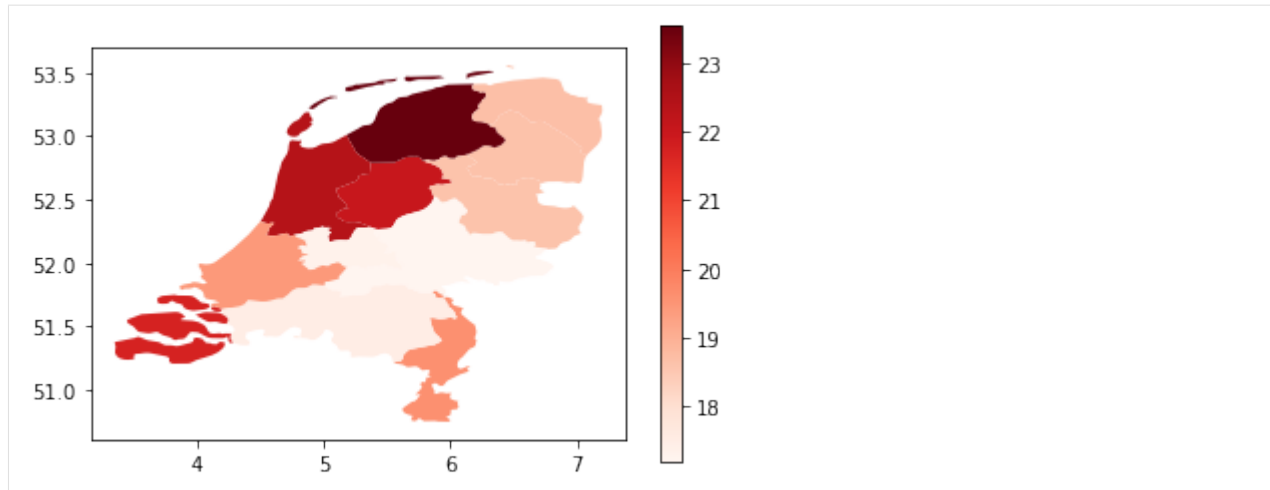
exp_province = exp_province_raw.gdf.groupby('province').apply(f).reset_index() # Result_
↳ is not a GeoDataFrame anymore
# add geometries to DataFrame and plot results
exp_province = gpd.GeoDataFrame(exp_province, geometry=None)
for prov, poly in zip(list(prov_names.values())[0], polygons_prov_NL['NLD']):
    exp_province.loc[exp_province.index[exp_province['province'] == prov], 'geometry'] =
↳ gpd.GeoDataFrame(geometry=[poly]).geometry.values
exp_province
print('Plot maximum wind per province for first event')
exp_province.plot(column='wind_0', cmap='Reds', legend=True)
plt.show()
print('Plot mean wind per province for second event')
exp_province.plot(column='wind_1', cmap='Reds', legend=True)
plt.show()

```

Plot maximum wind per province for first event



Plot mean wind per province for second event



5.6.2 Lines

Lines are common geographical representation of transport infrastructure like streets, train tracks or powerlines etc.

```
[11]: # under construction. here follows an example on how to deal with lines
```

5.7 Adaptation Measures

Adaptation measures are defined by parameters that alter the exposures, hazard or impact functions. Risk transfer options are also considered. Single measures are defined in the `Measure` class, which can be aggregated to a `MeasureSet`.

5.7.1 Measure class

A measure is characterized by the following attributes:

Related to measure's description: * `name` (str): name of the action * `haz_type` (str): related hazard type (peril), e.g. TC * `color_rgb` (np.array): integer array of size 3. Gives color code of this measure in RGB * `cost` (float): discounted cost (in same units as assets). Needs to be provided by the user. See the example provided in `climada_python/data/system/entity_template.xlsx` sheets `_measures_details` and `_discounting_sheet` to see how the discounting is done.

Related to a measure's impact: * `hazard_set` (str): file name of hazard to use * `hazard_freq_cutoff` (float): hazard frequency cutoff * `exposure_set` (str): file name of exposure to use * `hazard_inten_imp` (tuple): parameter a and b of hazard intensity change * `mdd_impact` (tuple): parameter a and b of the impact over the mean damage degree * `paa_impact` (tuple): parameter a and b of the impact over the percentage of affected assets * `imp_fun_map` (str): change of impact function id, e.g. '1to3' * `exp_region_id` (int): region id of the selected exposures to consider ALL the previous parameters * `risk_transf_attach` (float): risk transfer attachment. Applies to the whole exposure. * `risk_transf_cover` (float): risk transfer cover. Applies to the whole exposure.

Parameters description:

`hazard_set` and `exposures_set` provide the file names in h5 format (generated by CLIMADA) of the hazard and exposures to use as a result of the implementation of the measure. These might be further modified when applying the other parameters.

`hazard_inten_imp`, `mdd_impact` and `paa_impact` transform the impact functions linearly as follows:


```
intensity = intensity*hazard_inten_imp[0] + hazard_inten_imp[1]
mdd = mdd*mdd_impact[0] + mdd_impact[1]
paa = paa*paa_impact[0] + paa_impact[1]
```

`hazard_freq_cutoff` modifies the hazard by putting 0 intensities to the events whose impact exceedance frequency are greater than `hazard_freq_cutoff`.

`imp_fun_map` indicates the ids of the impact function to replace and its replacement. The `impf_XX` variable of Exposures with the affected impact function id will be correspondingly modified (XX refers to the `haz_type` of the measure).

`exp_region_id` will apply all the previous changes only to the `region_id` indicated. This means that only the exposures with that `region_id` and the hazard's centroids close to them will be modified with the previous changes, the other regions will remain unaffected to the measure.

`risk_transf_attach` and `risk_transf_cover` are the deductible and coverage of any event to happen.

Methods description:

The method `check()` validates the attributes. `apply()` applies the measure to a given exposure, impact function and hazard, returning their modified values. The parameters related to insurability (`risk_transf_attach` and `risk_transf_cover`) affect the resulting impact and are therefore not applied in the `apply()` method yet.

`calc_impact()` calls to `apply()`, applies the insurance parameters and returns the final impact and risk transfer of the measure. This method is called from the `CostBenefit` class.

The method `apply()` allows to visualize the effect of a measure. Here are some examples:

```
[1]: # effect of mdd_impact, paa_impact, hazard_inten_imp
%matplotlib inline
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard

# define measure
meas = Measure()
meas.name = 'Mangrove'
meas.haz_type = 'TC'
meas.color_rgb = np.array([1, 1, 1])
meas.cost = 5000000000
meas.mdd_impact = (1, 0)
meas.paa_impact = (1, -0.15)
meas.hazard_inten_imp = (1, -10) # reduces intensity by 10

# impact functions
impf_tc = ImpfTropCyclone()
impf_tc.set_emanuel_usa()
impf_all = ImpactFuncSet()
impf_all.append(impf_tc)
impf_all.plot()

# dummy Hazard and Exposures
haz = Hazard('TC') # this measure does not change hazard
exp = Exposures() # this measure does not change exposures
```

(continues on next page)

(continued from previous page)

new impact functions

new_exp, new_impfs, new_haz = meas.apply(exp, impf_all, haz)

axes = new_impfs.plot()

axes.set_title('TC: Modified impact function')

2021-04-23 15:54:09,171 - climada.entity.exposures.base - INFO - meta set to default_

↳value {}

2021-04-23 15:54:09,172 - climada.entity.exposures.base - INFO - tag set to default_

↳value File:

Description:

2021-04-23 15:54:09,173 - climada.entity.exposures.base - INFO - ref_year set to default_

↳value 2018

2021-04-23 15:54:09,173 - climada.entity.exposures.base - INFO - value_unit set to_

↳default value USD

2021-04-23 15:54:09,174 - climada.entity.exposures.base - INFO - crs set to default_

↳value: EPSG:4326

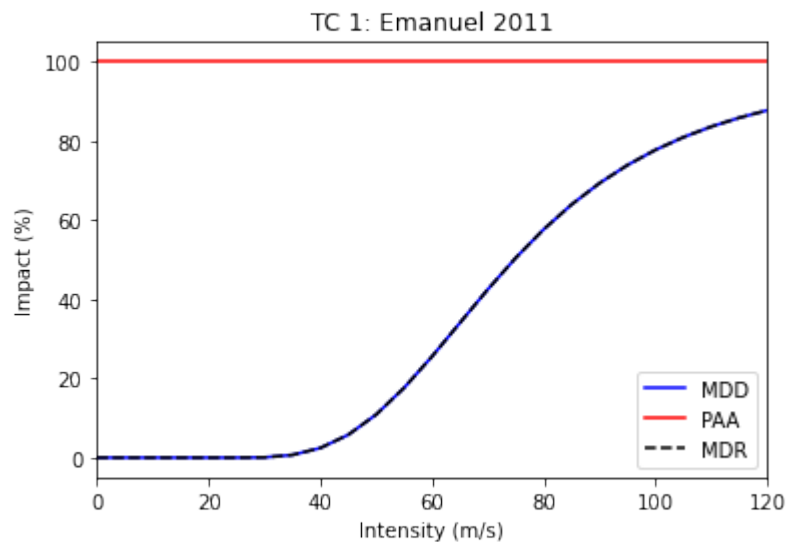
/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:

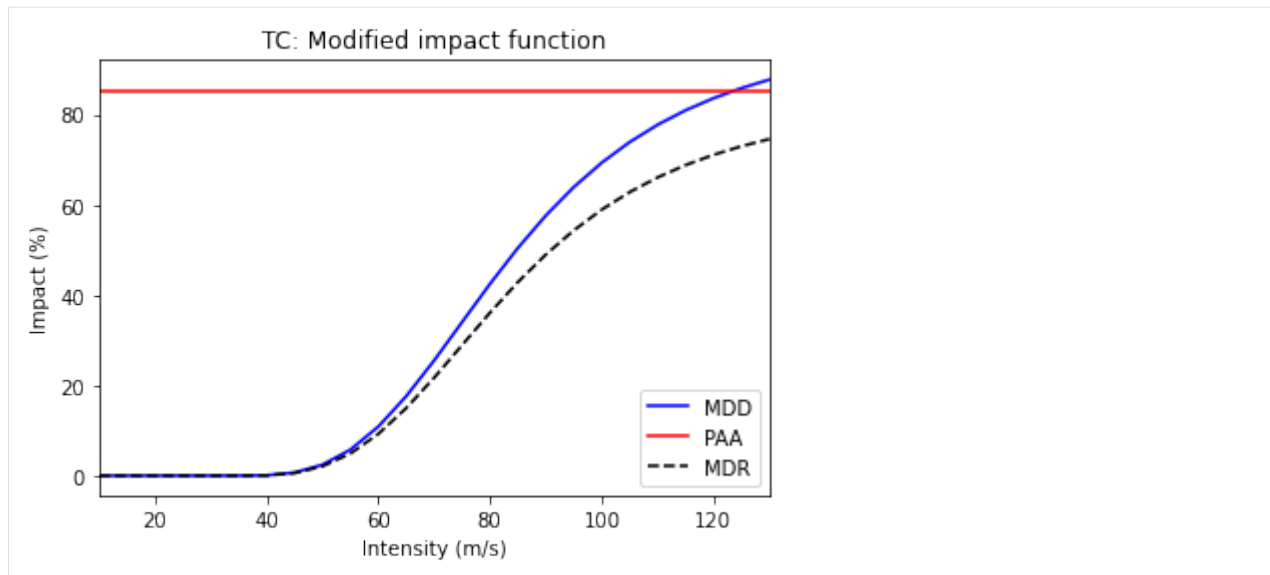
↳221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now_

↳deprecated and will not be supported in the future.

self.gdf.crs = self.meta['crs']

[1]: Text(0.5, 1.0, 'TC: Modified impact function')





```
[2]: # effect of hazard_freq_cutoff
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard
from climada.engine import Impact

from climada.util import HAZ_DEMO_H5, EXP_DEMO_H5

# define measure
meas = Measure()
meas.name = 'Mangrove'
meas.haz_type = 'TC'
meas.color_rgb = np.array([1, 1, 1])
meas.cost = 5000000000
meas.hazard_freq_cutoff = 0.0255

# impact functions
impf_tc = ImpfTropCyclone()
impf_tc.set_emanuel_usa()
impf_all = ImpactFuncSet()
impf_all.append(impf_tc)

# Hazard
haz = Hazard('TC')
haz.read_hdf5(HAZ_DEMO_H5)
haz.check()

# Exposures
exp = Exposures()
exp.read_hdf5(EXP_DEMO_H5)
exp.check()

# new hazard
```

(continues on next page)

(continued from previous page)

```

new_exp, new_impfs, new_haz = meas.apply(exp, impf_all, haz)
# if you look at the maximum intensity per centroid: new_haz does not contain the event.
↳with smaller impact (the most frequent)
haz.plot_intensity(0)
new_haz.plot_intensity(0)
# you might also compute the exceedance frequency curve of both hazard
imp = Impact()
imp.calc(exp, impf_all, haz)
ax = imp.calc_freq_curve().plot(label='original')

new_imp = Impact()
new_imp.calc(new_exp, new_impfs, new_haz)
new_imp.calc_freq_curve().plot(axis=ax, label='measure') # the damages for events with.
↳return periods > 1/0.0255 ~ 40 are 0
ax.legend()

2021-04-23 15:54:09,493 - climada.hazard.base - INFO - Reading /Users/zeliestalhanske/
↳climada/demo/data/tc_fl_1990_2004.h5
2021-04-23 15:54:09,533 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-23 15:54:09,534 - climada.entity.exposures.base - INFO - tag set to default.
↳value File:
Description:
2021-04-23 15:54:09,535 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-04-23 15:54:09,535 - climada.entity.exposures.base - INFO - value_unit set to.
↳default value USD
2021-04-23 15:54:09,536 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-04-23 15:54:09,548 - climada.entity.exposures.base - INFO - Reading /Users/
↳zeliestalhanske/climada/demo/data/exp_demo_today.h5
2021-04-23 15:54:09,617 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-23 15:54:09,618 - climada.entity.exposures.base - INFO - tag set to default.
↳value File:
Description:
2021-04-23 15:54:09,619 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-04-23 15:54:09,620 - climada.entity.exposures.base - INFO - value_unit set to.
↳default value USD
2021-04-23 15:54:09,627 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-04-23 15:54:09,640 - climada.entity.exposures.base - INFO - centr_ not set.
2021-04-23 15:54:09,642 - climada.entity.exposures.base - INFO - Matching 50 exposures.
↳with 2500 centroids.
2021-04-23 15:54:09,660 - climada.engine.impact - INFO - Calculating damage for 50.
↳assets (>0) and 216 events.

/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳<authority>:<code>' is the preferred initialization method. When making the change, be.
↳mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳#axis-order-changes-in-proj-6

```

(continues on next page)

(continued from previous page)

```

return _prepare_from_string(" ".join(pjargs))
/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now
↳deprecated and will not be supported in the future.
self.gdf.crs = self.meta['crs']

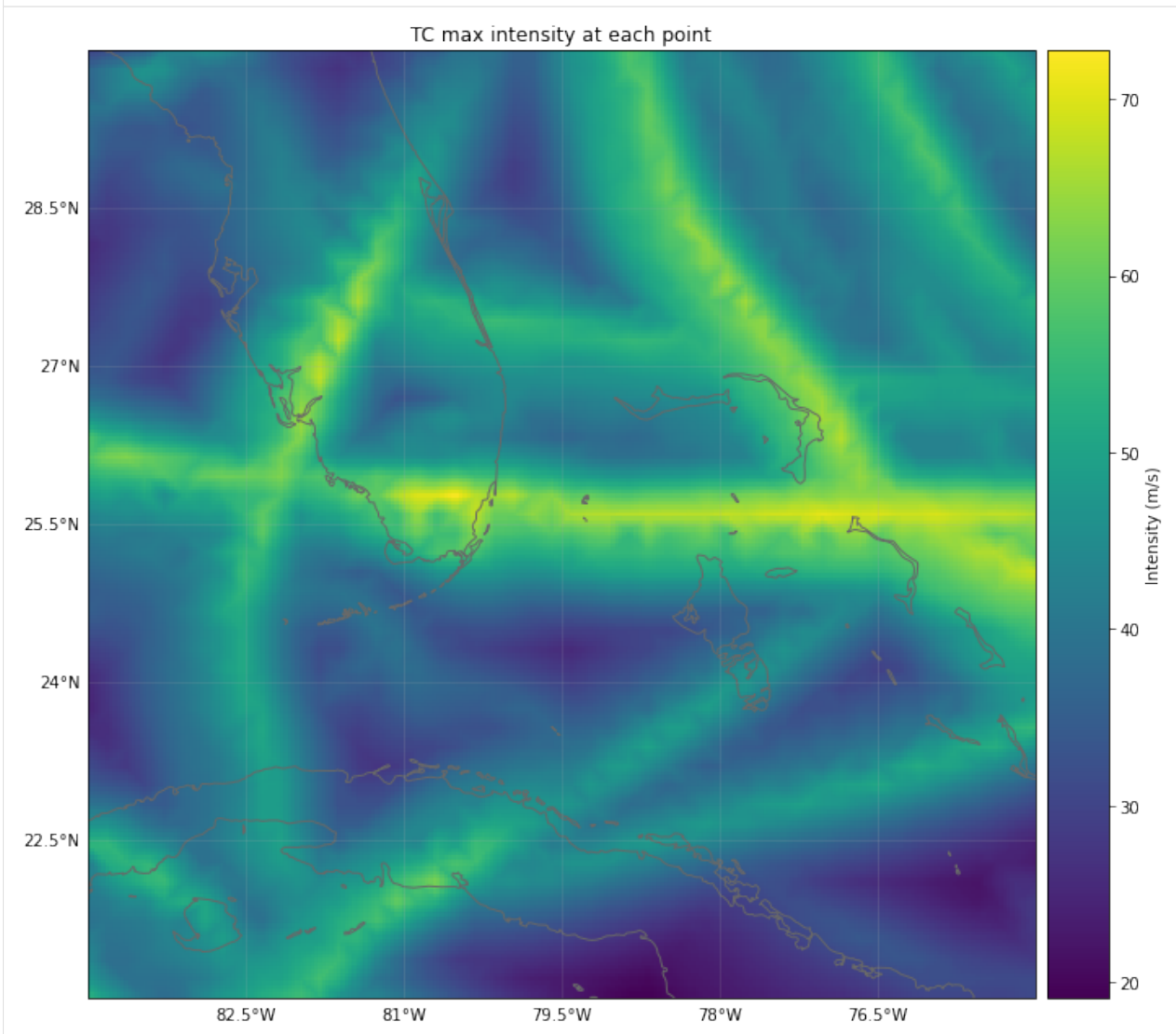
```

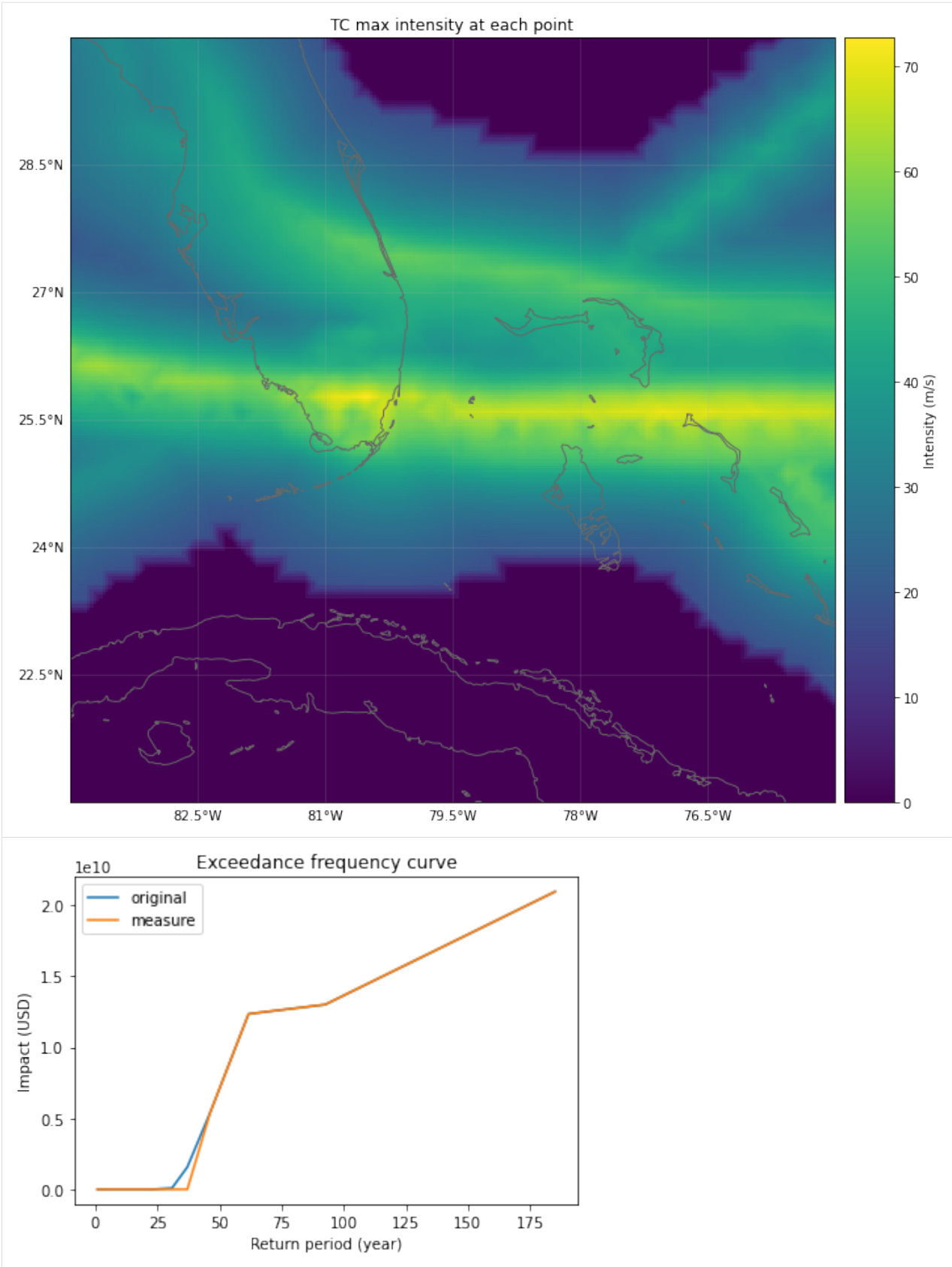
```

2021-04-23 15:54:10,721 - climada.engine.impact - INFO - Exposures matching centroids
↳found in centr_TC
2021-04-23 15:54:10,723 - climada.engine.impact - INFO - Calculating damage for 50
↳assets (>0) and 216 events.
2021-04-23 15:54:10,745 - climada.engine.impact - INFO - Exposures matching centroids
↳found in centr_TC
2021-04-23 15:54:10,747 - climada.engine.impact - INFO - Calculating damage for 50
↳assets (>0) and 216 events.

```

[2]: <matplotlib.legend.Legend at 0x7f8f71b5c8e0>





```

[3]: # effect of exp_region_id
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard
from climada.engine import Impact

from climada.util import HAZ_DEMO_H5, EXP_DEMO_H5

# define measure
meas = Measure()
meas.name = 'Building code'
meas.haz_type = 'TC'
meas.color_rgb = np.array([1, 1, 1])
meas.cost = 5000000000
meas.hazard_freq_cutoff = 0.00455
meas.exp_region_id = [1] # apply measure to points close to exposures with region_id=1

# impact functions
impf_tc = ImpfTropCyclone()
impf_tc.set_emanuel_usa()
impf_all = ImpactFuncSet()
impf_all.append(impf_tc)

# Hazard
haz = Hazard('TC')
haz.read_hdf5(HAZ_DEMO_H5)
haz.check()

# Exposures
exp = Exposures()
exp.read_hdf5(EXP_DEMO_H5)
#exp['region_id'] = np.ones(exp.shape[0])
exp.check()
# all exposures have region_id=1
exp.plot_hexbin(buffer=1.0)

# new hazard
new_exp, new_impfs, new_haz = meas.apply(exp, impf_all, haz)
# the cutoff has been applied only in the region of the exposures
haz.plot_intensity(0)
new_haz.plot_intensity(0)

# the exceedance frequency has only been computed for the selected exposures before
# →doing the cutoff.
# since we have removed the hazard of the places with exposure, the new exceedance
# →frequency curve is zero.
imp = Impact()
imp.calc(exp, impf_all, haz)
imp.calc_freq_curve().plot()

new_imp = Impact()
new_imp.calc(new_exp, new_impfs, new_haz)

```

(continues on next page)

(continued from previous page)

```

new_imp.calc_freq_curve().plot()

2021-04-23 15:54:12,871 - climada.hazard.base - INFO - Reading /Users/zeliestalhanske/
↳ climada/demo/data/tc_fl_1990_2004.h5
2021-04-23 15:54:12,905 - climada.entity.exposures.base - INFO - meta set to default_
↳ value {}
2021-04-23 15:54:12,905 - climada.entity.exposures.base - INFO - tag set to default_
↳ value File:
Description:
2021-04-23 15:54:12,906 - climada.entity.exposures.base - INFO - ref_year set to default_
↳ value 2018
2021-04-23 15:54:12,906 - climada.entity.exposures.base - INFO - value_unit set to_
↳ default value USD
2021-04-23 15:54:12,907 - climada.entity.exposures.base - INFO - crs set to default_
↳ value: EPSG:4326
2021-04-23 15:54:12,920 - climada.entity.exposures.base - INFO - Reading /Users/
↳ zeliestalhanske/climada/demo/data/exp_demo_today.h5
2021-04-23 15:54:12,937 - climada.entity.exposures.base - INFO - meta set to default_
↳ value {}
2021-04-23 15:54:12,938 - climada.entity.exposures.base - INFO - tag set to default_
↳ value File:
Description:
2021-04-23 15:54:12,938 - climada.entity.exposures.base - INFO - ref_year set to default_
↳ value 2018
2021-04-23 15:54:12,938 - climada.entity.exposures.base - INFO - value_unit set to_
↳ default value USD
2021-04-23 15:54:12,944 - climada.entity.exposures.base - INFO - crs set to default_
↳ value: EPSG:4326
2021-04-23 15:54:12,955 - climada.entity.exposures.base - INFO - centr_ not set.

/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be_
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
return _prepare_from_string(" ".join(pjargs))
/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳ 221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now_
↳ deprecated and will not be supported in the future.
self.gdf.crs = self.meta['crs']

2021-04-23 15:54:13,667 - climada.entity.exposures.base - INFO - meta set to default_
↳ value {}
2021-04-23 15:54:13,668 - climada.entity.exposures.base - INFO - tag set to default_
↳ value File:
Description:
2021-04-23 15:54:13,668 - climada.entity.exposures.base - INFO - ref_year set to default_
↳ value 2018
2021-04-23 15:54:13,670 - climada.entity.exposures.base - INFO - value_unit set to_
↳ default value USD
2021-04-23 15:54:13,673 - climada.entity.exposures.base - INFO - Matching 50 exposures_
↳ with 2500 centroids.
2021-04-23 15:54:13,682 - climada.engine.impact - INFO - Calculating damage for 50_
↳ assets (>0) and 216 events.

```

(continues on next page)

(continued from previous page)

```

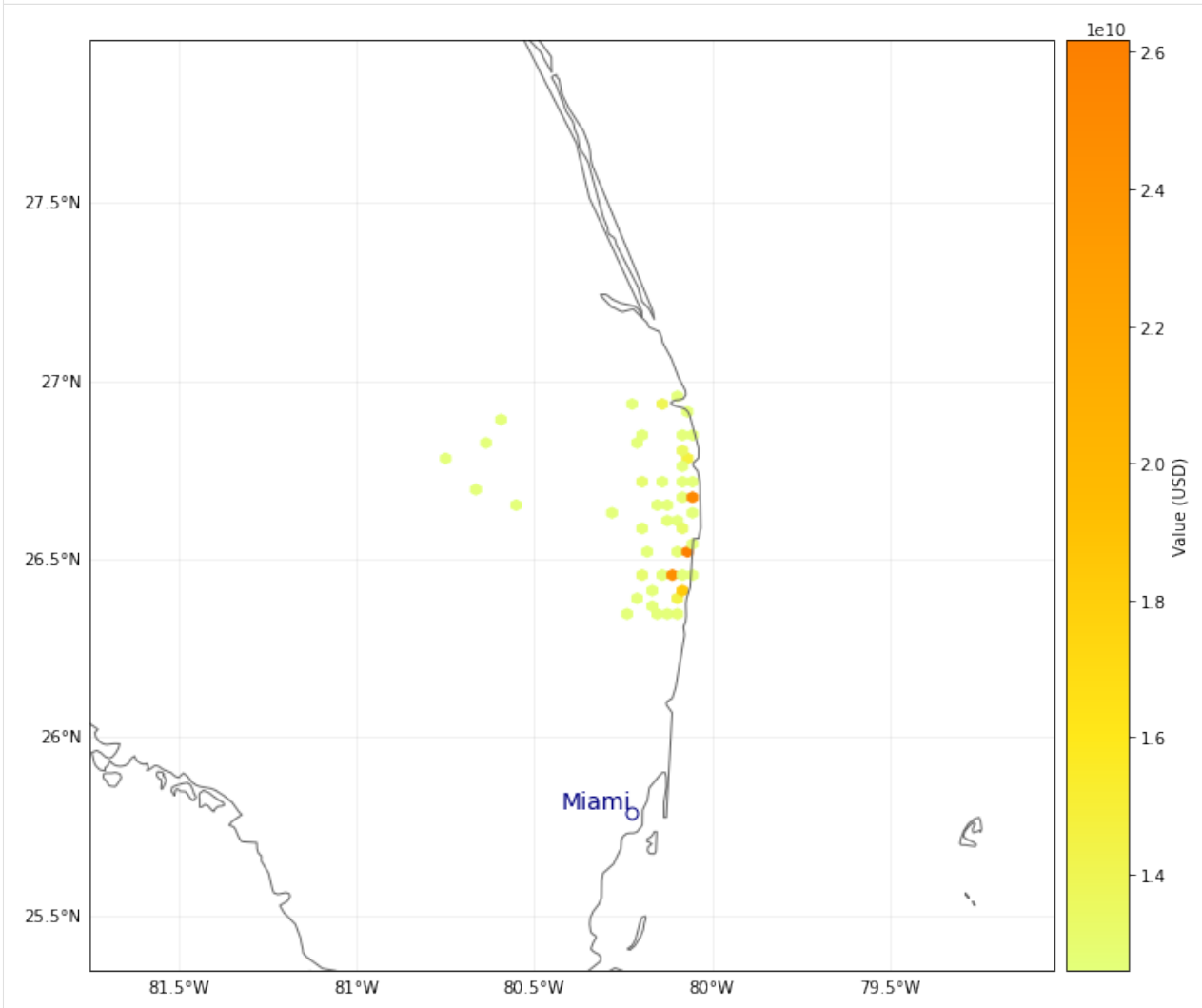
2021-04-23 15:54:13,699 - climada.entity.exposures.base - INFO - Matching 50 exposures_
↳with 2500 centroids.
2021-04-23 15:54:14,779 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-23 15:54:14,780 - climada.engine.impact - INFO - Calculating damage for 50_
↳assets (>0) and 216 events.
2021-04-23 15:54:14,814 - climada.entity.exposures.base - INFO - Matching 50 exposures_
↳with 2500 centroids.
2021-04-23 15:54:14,825 - climada.engine.impact - INFO - Calculating damage for 50_
↳assets (>0) and 216 events.

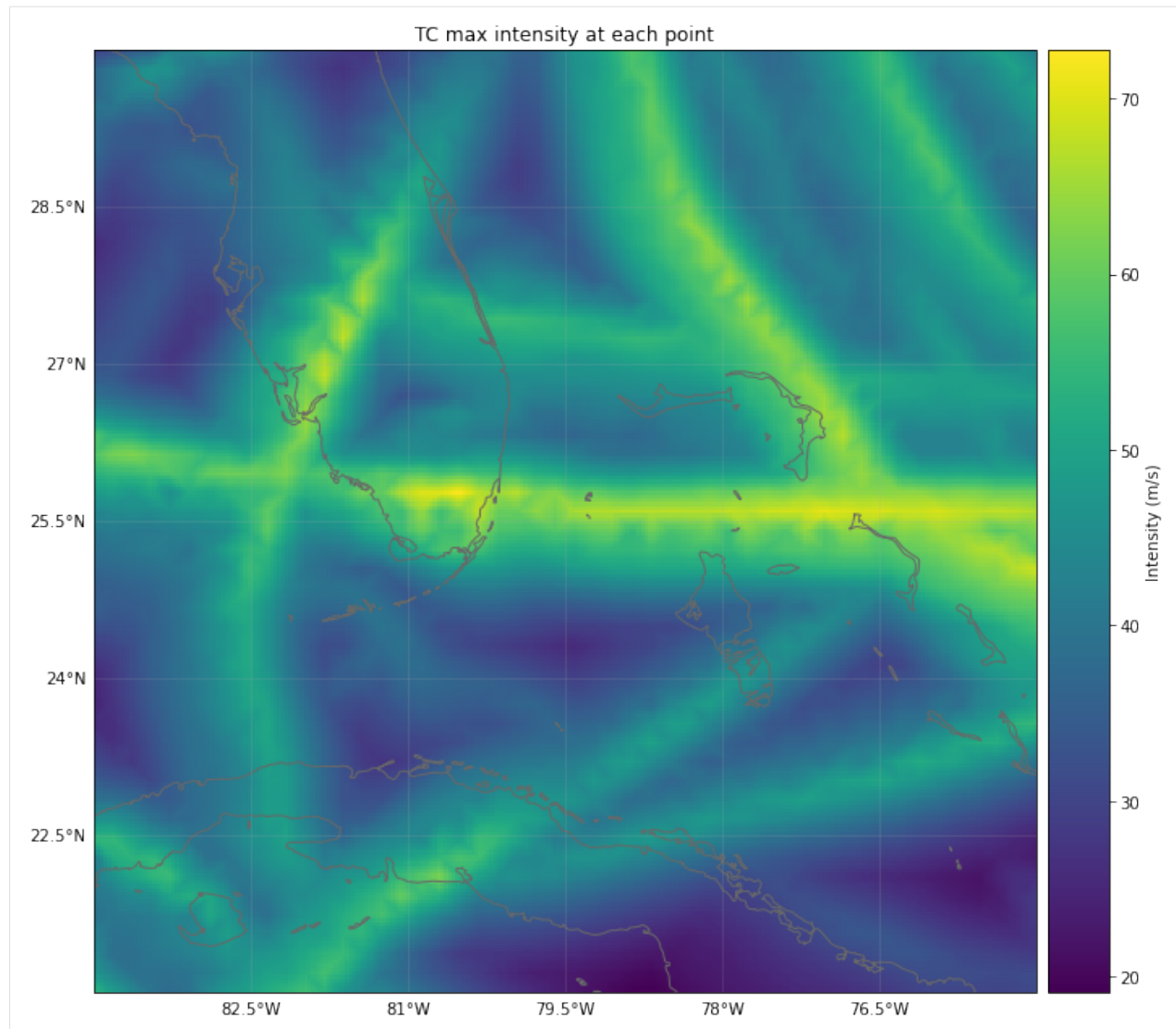
```

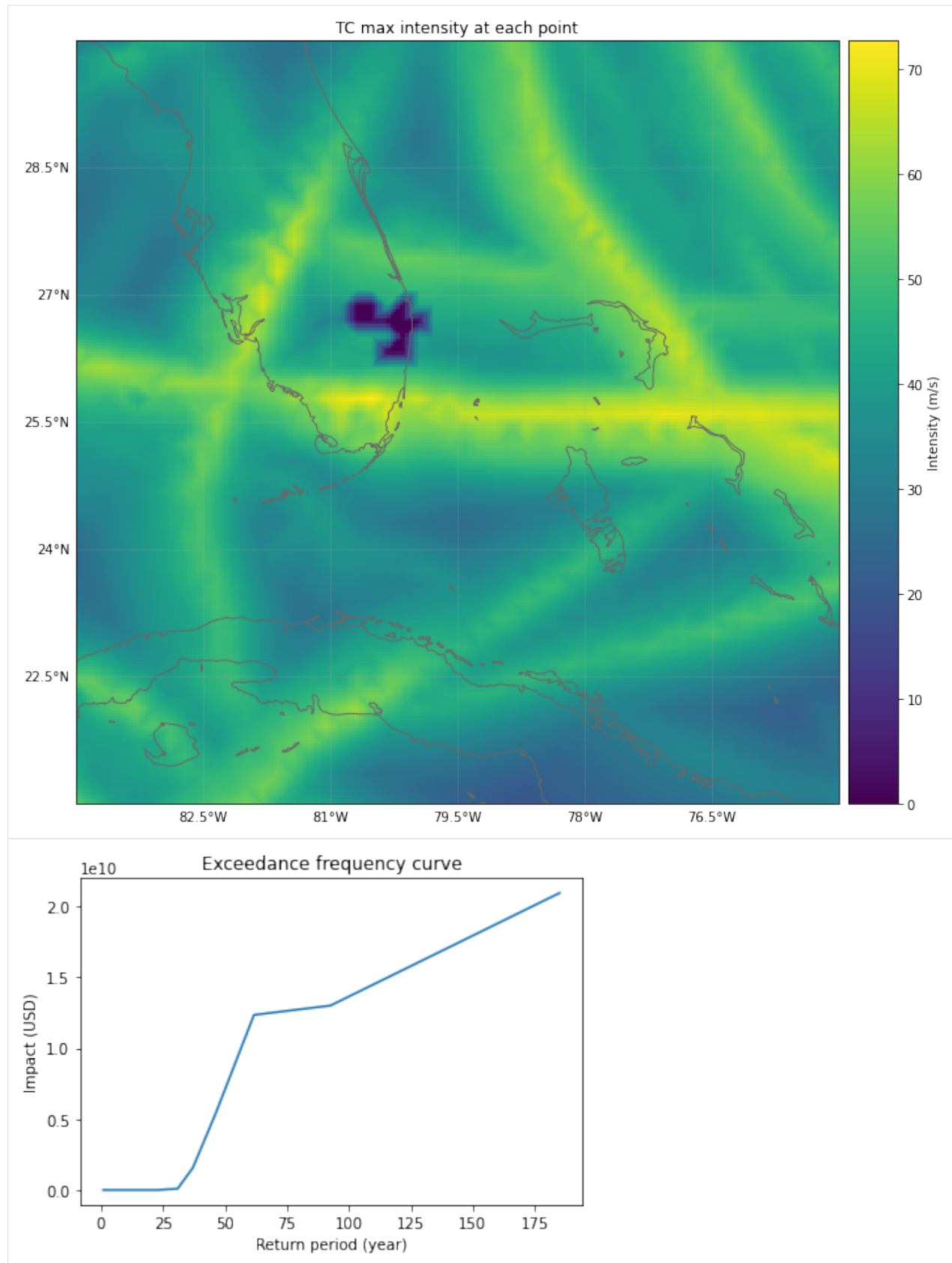
```

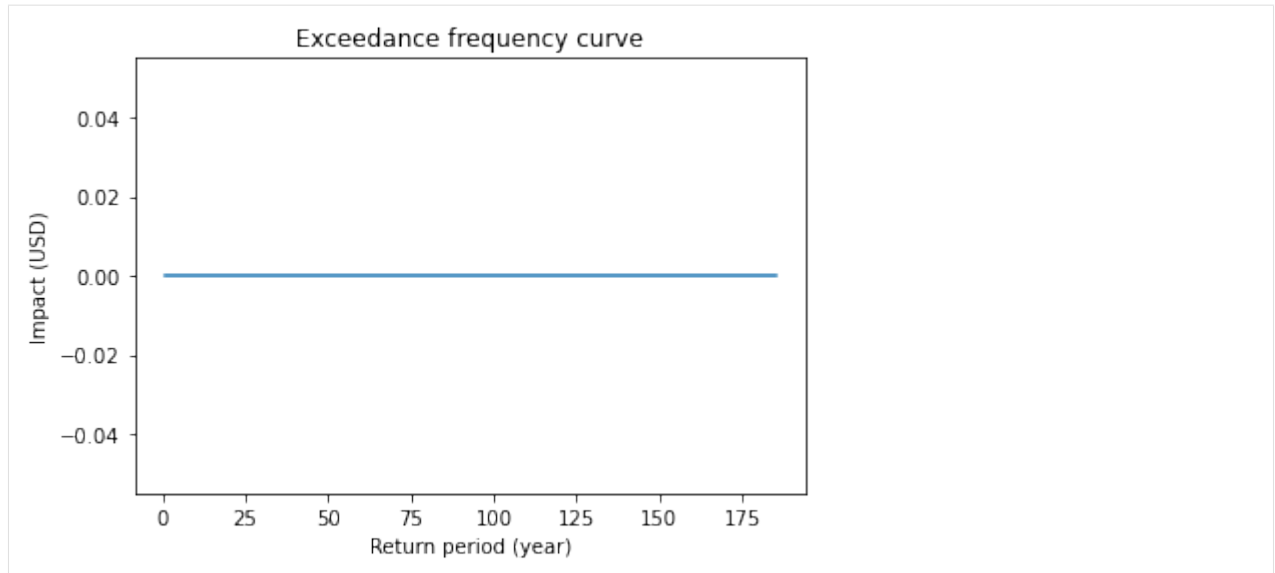
[3]: <AxesSubplot:title={'center':'Exceedance frequency curve'}, xlabel='Return period (year)
↳', ylabel='Impact (USD)'\>

```









```
[4]: # effect of risk_transf_attach and risk_transf_cover
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard
from climada.engine import Impact

from climada.util import HAZ_DEMO_H5, EXP_DEMO_H5

# define measure
meas = Measure()
meas.name = 'Insurance'
meas.haz_type = 'TC'
meas.color_rgb = np.array([1, 1, 1])
meas.cost = 5000000000
meas.risk_transf_attach = 5.0e8
meas.risk_transf_cover = 1.0e9

# impact functions
impf_tc = ImpfTropCyclone()
impf_tc.set_emanuel_usa()
impf_all = ImpactFuncSet()
impf_all.append(impf_tc)

# Hazard
haz = Hazard('TC')
haz.read_hdf5(HAZ_DEMO_H5)
haz.check()

# Exposures
exp = Exposures()
exp.read_hdf5(EXP_DEMO_H5)
exp.check()
```

(continues on next page)

(continued from previous page)

```

# impact before
imp = Impact()
imp.calc(exp, impf_all, haz)
imp.calc_freq_curve().plot()

# impact after. risk_transf will be added to the cost of the measure
imp_new, risk_transf = meas.calc_impact(exp, impf_all, haz)
imp_new.calc_freq_curve().plot()
print('risk_transfer {:.3}'.format(risk_transf.aai_agg))

2021-04-23 15:54:19,137 - climada.hazard.base - INFO - Reading /Users/zeliestalhanske/
↳ climada/demo/data/tc_fl_1990_2004.h5
2021-04-23 15:54:19,169 - climada.entity.exposures.base - INFO - meta set to default_
↳ value {}
2021-04-23 15:54:19,169 - climada.entity.exposures.base - INFO - tag set to default_
↳ value File:
Description:
2021-04-23 15:54:19,170 - climada.entity.exposures.base - INFO - ref_year set to default_
↳ value 2018
2021-04-23 15:54:19,170 - climada.entity.exposures.base - INFO - value_unit set to_
↳ default value USD
2021-04-23 15:54:19,171 - climada.entity.exposures.base - INFO - crs set to default_
↳ value: EPSG:4326
2021-04-23 15:54:19,183 - climada.entity.exposures.base - INFO - Reading /Users/
↳ zeliestalhanske/climada/demo/data/exp_demo_today.h5
2021-04-23 15:54:19,201 - climada.entity.exposures.base - INFO - meta set to default_
↳ value {}
2021-04-23 15:54:19,202 - climada.entity.exposures.base - INFO - tag set to default_
↳ value File:
Description:
2021-04-23 15:54:19,202 - climada.entity.exposures.base - INFO - ref_year set to default_
↳ value 2018
2021-04-23 15:54:19,203 - climada.entity.exposures.base - INFO - value_unit set to_
↳ default value USD
2021-04-23 15:54:19,209 - climada.entity.exposures.base - INFO - crs set to default_
↳ value: EPSG:4326
2021-04-23 15:54:19,221 - climada.entity.exposures.base - INFO - centr_ not set.
2021-04-23 15:54:19,223 - climada.entity.exposures.base - INFO - Matching 50 exposures_
↳ with 2500 centroids.
2021-04-23 15:54:19,231 - climada.engine.impact - INFO - Calculating damage for 50_
↳ assets (>0) and 216 events.
2021-04-23 15:54:19,255 - climada.engine.impact - INFO - Exposures matching centroids_
↳ found in centr_TC
2021-04-23 15:54:19,258 - climada.engine.impact - INFO - Calculating damage for 50_
↳ assets (>0) and 216 events.
risk_transfer 2.7e+07

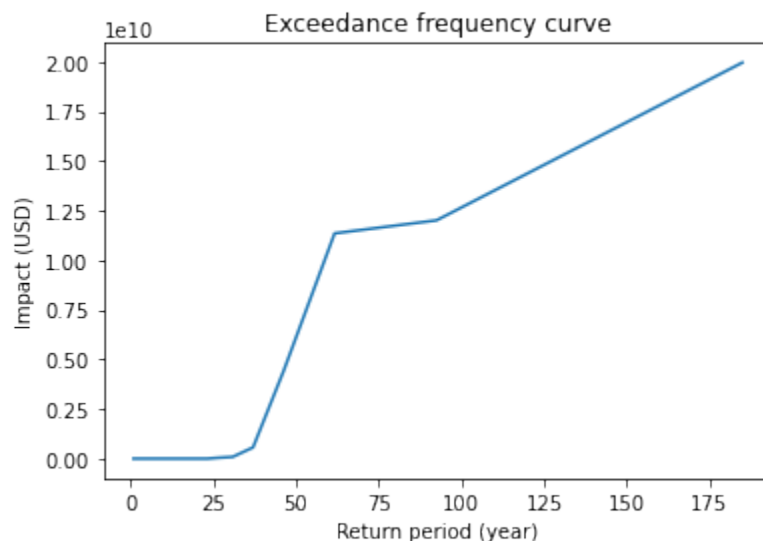
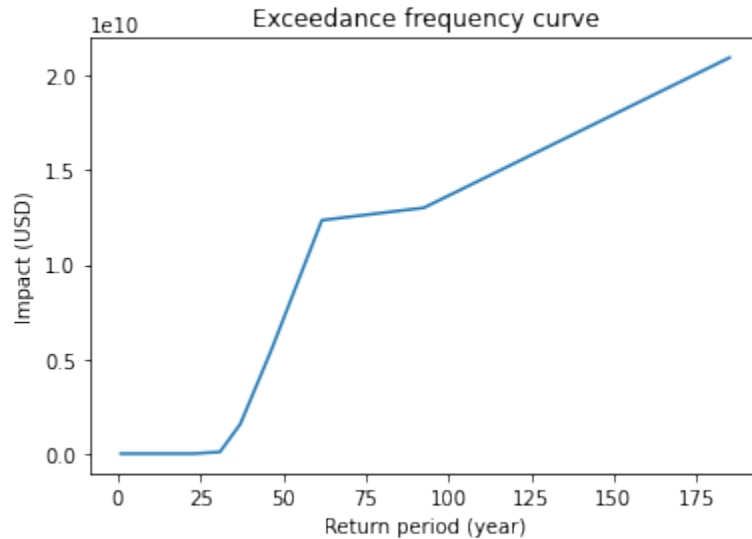
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be_
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
return _prepare_from_string(" ".join(pjargs))

```

(continues on next page)

(continued from previous page)

```
/Users/zeliestalhanske/python_projects/climada_python/climada/entity/exposures/base.py:
↳221: FutureWarning: Assigning CRS to a GeoDataFrame without a geometry column is now
↳deprecated and will not be supported in the future.
self.gdf.crs = self.meta['crs']
```



5.7.2 MeasureSet class

Similarly to the `ImpactFuncSet`, `MeasureSet` is a container which handles `Measure` instances through the methods `append()`, `extend()`, `remove_measure()` and `get_measure()`. Use the `check()` method to make sure all the measures have been properly set.

`MeasureSet` contains the attribute `tag` of type `Tag`, which stores information about the data: the original file name and a description.

```
[5]: from climada.entity import MeasureSet
help(MeasureSet)
```

Help on class MeasureSet in module climada.entity.measures.measure_set:

```
class MeasureSet(builtins.object)
|   Contains measures of type Measure. Loads from
|   files with format defined in FILE_EXT.
|
|   Attributes:
|       tag (Tag): information about the source data
|       _data (dict): contains Measure classes. It's not supposed to be
|       directly accessed. Use the class methods instead.
|
|   Methods defined here:
|
|   __init__(self)
|       Empty initialization.
|
|   Examples:
|       Fill MeasureSet with values and check consistency data:
|
|       >>> act_1 = Measure()
|       >>> act_1.name = 'Seawall'
|       >>> act_1.color_rgb = np.array([0.1529, 0.2510, 0.5451])
|       >>> act_1.hazard_intensity = (1, 0)
|       >>> act_1.mdd_impact = (1, 0)
|       >>> act_1.paa_impact = (1, 0)
|       >>> meas = MeasureSet()
|       >>> meas.append(act_1)
|       >>> meas.tag.description = "my dummy MeasureSet."
|       >>> meas.check()
|
|       Read measures from file and checks consistency data:
|
|       >>> meas = MeasureSet()
|       >>> meas.read_excel(ENT_TEMPLATE_XLS)
|
|   append(self, meas)
|       Append an Measure. Override if same name and haz_type.
|
|       Parameters:
|           meas (Measure): Measure instance
|
|       Raises:
|           ValueError
|
|   check(self)
|       Check instance attributes.
|
|       Raises:
|           ValueError
|
|   clear(self)
|       Reinitialize attributes.
```

(continues on next page)

(continued from previous page)

```

| extend(self, meas_set)
|     Extend measures of input MeasureSet to current
|     MeasureSet. Overwrite Measure if same name and haz_type.
|
|     Parameters:
|         impact_funcs (MeasureSet): ImpactFuncSet instance to extend
|
|     Raises:
|         ValueError
|
| get_hazard_types(self, meas=None)
|     Get measures hazard types contained for the name provided.
|     Return all hazard types if no input name.
|
|     Parameters:
|         name (str, optional): measure name
|
|     Returns:
|         list(str)
|
| get_measure(self, haz_type=None, name=None)
|     Get ImpactFunc(s) of input hazard type and/or id.
|     If no input provided, all impact functions are returned.
|
|     Parameters:
|         haz_type (str, optional): hazard type
|         name (str, optional): measure name
|
|     Returns:
|         Measure (if haz_type and name),
|         list(Measure) (if haz_type or name),
|         {Measure.haz_type: {Measure.name : Measure}} (if None)
|
| get_names(self, haz_type=None)
|     Get measures names contained for the hazard type provided.
|     Return all names for each hazard type if no input hazard type.
|
|     Parameters:
|         haz_type (str, optional): hazard type from which to obtain the names
|
|     Returns:
|         list(Measure.name) (if haz_type provided),
|         {Measure.haz_type : list(Measure.name)} (if no haz_type)
|
| read_excel(self, file_name, description='', var_names={'sheet_name': 'measures',
| → 'col_name': {'name': 'name', 'color': 'color', 'cost': 'cost', 'haz_int_a': 'hazard_
| → intensity impact a', 'haz_int_b': 'hazard intensity impact b', 'haz_frq': 'hazard high_
| → frequency cutoff', 'haz_set': 'hazard event set', 'mdd_a': 'MDD impact a', 'mdd_b':
| → 'MDD impact b', 'paa_a': 'PAA impact a', 'paa_b': 'PAA impact b', 'fun_map':
| → 'damagefunctions map', 'exp_set': 'assets file', 'exp_reg': 'Region_ID', 'risk_att':
| → 'risk transfer attachment', 'risk_cov': 'risk transfer cover', 'risk_fact': 'risk_
| → transfer cost factor', 'haz': 'peril_ID'}})

```

(continues on next page)

(continued from previous page)

```

|         Read excel file following template and store variables.
|
|         Parameters:
|             file_name (str): absolute file name
|             description (str, optional): description of the data
|             var_names (dict, optional): name of the variables in the file
|
|         read_mat(self, file_name, description='', var_names={'sup_field_name': 'entity',
|         ↪ 'field_name': 'measures', 'var_name': {'name': 'name', 'color': 'color', 'cost': 'cost
|         ↪ ', 'haz_int_a': 'hazard_intensity_impact_a', 'haz_int_b': 'hazard_intensity_impact_b',
|         ↪ 'haz_freq': 'hazard_high_frequency_cutoff', 'haz_set': 'hazard_event_set', 'mdd_a':
|         ↪ 'MDD_impact_a', 'mdd_b': 'MDD_impact_b', 'paa_a': 'PAA_impact_a', 'paa_b': 'PAA_impact_
|         ↪ b', 'fun_map': 'damagefunctions_map', 'exp_set': 'assets_file', 'exp_reg': 'Region_ID',
|         ↪ 'risk_att': 'risk_transfer_attachement', 'risk_cov': 'risk_transfer_cover', 'haz':
|         ↪ 'peril_ID'}})
|         Read MATLAB file generated with previous MATLAB CLIMADA version.
|
|         Parameters:
|             file_name (str): absolute file name
|             description (str, optional): description of the data
|             var_names (dict, optional): name of the variables in the file
|
|         remove_measure(self, haz_type=None, name=None)
|             Remove impact function(s) with provided hazard type and/or id.
|             If no input provided, all impact functions are removed.
|
|         Parameters:
|             haz_type (str, optional): all impact functions with this hazard
|             name (str, optional): measure name
|
|         size(self, haz_type=None, name=None)
|             Get number of measures contained with input hazard type and
|             /or id. If no input provided, get total number of impact functions.
|
|         Parameters:
|             haz_type (str, optional): hazard type
|             name (str, optional): measure name
|
|         Returns:
|             int
|
|         write_excel(self, file_name, var_names={'sheet_name': 'measures', 'col_name': {'name
|         ↪ ': 'name', 'color': 'color', 'cost': 'cost', 'haz_int_a': 'hazard intensity impact a',
|         ↪ 'haz_int_b': 'hazard intensity impact b', 'haz_freq': 'hazard high frequency cutoff',
|         ↪ 'haz_set': 'hazard event set', 'mdd_a': 'MDD impact a', 'mdd_b': 'MDD impact b', 'paa_a
|         ↪ ': 'PAA impact a', 'paa_b': 'PAA impact b', 'fun_map': 'damagefunctions map', 'exp_set
|         ↪ ': 'assets file', 'exp_reg': 'Region_ID', 'risk_att': 'risk transfer attachement',
|         ↪ 'risk_cov': 'risk transfer cover', 'risk_fact': 'risk transfer cost factor', 'haz':
|         ↪ 'peril_ID'}})
|         Write excel file following template.
|
|         Parameters:

```

(continues on next page)

(continued from previous page)

```

|         file_name (str): absolute file name to write
|         var_names (dict, optional): name of the variables in the file
|
| -----
| Data descriptors defined here:
|
| __dict__
|     dictionary for instance variables (if defined)
|
| __weakref__
|     list of weak references to the object (if defined)

```

```

[6]: # build measures
import numpy as np
import matplotlib.pyplot as plt
from climada.entity.measures import Measure, MeasureSet

meas_1 = Measure()
meas_1.haz_type = 'TC'
meas_1.name = 'Mangrove'
meas_1.color_rgb = np.array([1, 1, 1])
meas_1.cost = 500000000
meas_1.mdd_impact = (1, 2)
meas_1.paa_impact = (1, 2)
meas_1.hazard_inten_imp = (1, 2)
meas_1.risk_transf_cover = 500

meas_2 = Measure()
meas_2.haz_type = 'TC'
meas_2.name = 'Sandbags'
meas_2.color_rgb = np.array([1, 1, 1])
meas_2.cost = 220000000
meas_2.mdd_impact = (1, 2)
meas_2.paa_impact = (1, 3)
meas_2.hazard_inten_imp = (1, 2)
meas_2.exp_region_id = 2

# gather all measures
meas_set = MeasureSet()
meas_set.append(meas_1)
meas_set.append(meas_2)
meas_set.check()

# select one measure
meas_sel = meas_set.get_measure(name='Sandbags')
print(meas_sel[0].name, meas_sel[0].cost)

Sandbags 220000000

```

5.7.3 Read measures of an Excel file

Measures defined in an excel file following the template provided in sheet measures of `climada_python/data/system/entity_template.xlsx` can be ingested directly using the method `read_excel()`.

```
[7]: from climada.entity.measures import MeasureSet
     from climada.util import ENT_TEMPLATE_XLS

     # Fill DataFrame from Excel file
     file_name = ENT_TEMPLATE_XLS # provide absolute path of the excel file
     meas_set = MeasureSet()
     meas_set.read_excel(file_name)
     print('Read file:', meas_set.tag.file_name)

Read file: /Users/zeliestalhanske/climada/data/entity_template.xlsx
```

5.7.4 Write measures

Measures can be written in Excel format using `write_excel()` method.

```
[8]: from climada.entity.measures import MeasureSet
     from climada.util import ENT_TEMPLATE_XLS

     # Fill DataFrame from Excel file
     file_name = ENT_TEMPLATE_XLS # provide absolute path of the excel file
     meas_set = MeasureSet()
     meas_set.read_excel(file_name)

     # write file
     meas_set.write_excel('results/tutorial_meas_set.xlsx')
```

Pickle can always be used as well:

```
[9]: from climada.util.save import save
     # this generates a results folder in the current path and stores the output there
     save('tutorial_meas_set.p', meas_set)

2021-04-23 15:54:19,930 - climada.util.save - INFO - Written file /Users/zeliestalhanske/
python_projects/climada_python/doc/tutorial/results/tutorial_meas_set.p
```

5.8 Hazard class

5.8.1 What is a hazard?

A hazard describes weather events such as storms, floods, droughts, or heat waves both in terms of probability of occurrence as well as physical intensity.

5.8.2 How are hazards embedded in the CLIMADA architecture?

Hazards are defined by the base class `Hazard` which gathers the required attributes that enable the impact computation (such as centroids, frequency per event, and intensity per event and centroid) and common methods such as readers and visualization functions. Each hazard class collects historical data or model simulations and transforms them, if necessary, in order to construct a coherent event database. Stochastic events can be generated taking into account the frequency and main intensity characteristics (such as local water depth for floods or gust speed for storms) of historical events, producing an ensemble of probabilistic events for each historical event. CLIMADA provides therefore an event-based probabilistic approach which does not depend on a hypothesis of a priori general probability distribution choices. The source of the historical data (e.g. inventories or satellite images) or model simulations (e.g. synthetic tropical cyclone tracks) and the methodologies used to compute the hazard attributes and its stochastic events depend on each hazard type and are defined in its corresponding Hazard-derived class (e.g. `TropCyclone` for tropical cyclones, explained in the tutorial [TropCyclone](#)). This procedure provides a solid and homogeneous methodology to compute impacts worldwide. In the case where the risk analysis comprises a specific region where good quality data or models describing the hazard intensity and frequency are available, these can be directly ingested by the platform through the reader functions, skipping the hazard modelling part (in total or partially), and allowing us to easily and seamlessly combine CLIMADA with external sources. Hence the impact model can be used for a wide variety of applications, e.g. deterministically to assess the impact of a single (past or future) event or to quantify risk based on a (large) set of probabilistic events. Note that since the `Hazard` class is not an abstract class, any hazard that is not defined in CLIMADA can still be used by providing the Hazard attributes.

5.8.3 What do hazards look like in CLIMADA?

A `Hazard` contains events of some hazard type defined at **centroids**. There are certain variables in a `Hazard` instance that *are needed* to compute the impact, while others are *descriptive* and can therefore be set with default values. The full list of looks like this:

Mandatory variables	Data Type	Description
tag	<code>TagHazard()</code>	information about the source
units	(str)	units of the intensity
centroids	<code>Centroids()</code>	centroids of the events
event_id	(np.array)	id (>0) of each event
frequency	(np.array)	frequency of each event in years
intensity	(sparse.csr_matrix)	intensity of the events at centroids
fraction	(sparse.csr_matrix)	fraction of affected exposures for each event at each centroid

Descriptive variables	Data Type	Description
date	(np.array)	integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)
orig	(np.array)	flags indicating historical events (True) or probabilistic (False)
event_name	(list(str))	name of each event (default: event_id)

Note that `intensity` and `fraction` are `scipy.sparse` matrices of size `num_events` x `num_centroids`. The `Centroids` class contains the geographical coordinates where the hazard is defined. A `Centroids` instance provides the coordinates either as points or raster data together with their Coordinate Reference System (CRS). The default CRS used in `climada` is the usual EPSG:4326. `Centroids` provides moreover methods to compute centroids areas, on land mask, country iso mask or distance to coast.

How is this tutorial structured?

****Part 1:**** Read hazards from raster data

****Part 2:**** Read hazards from other data

****Part 3:**** Define hazards manually

****Part 4:**** Analyse hazards

****Part 5:**** Visualize hazards

****Part 6:**** Write (=save) hazards

Part 1: Read hazards from raster data

Raster data can be read in any format accepted by `rasterio` using Hazard's `set_raster()` method. The raster information might refer to the intensity or fraction of the hazard. Different configuration options such as transforming the coordinates, changing the CRS and reading only a selected area or band are available through the `set_raster()` arguments as follows:

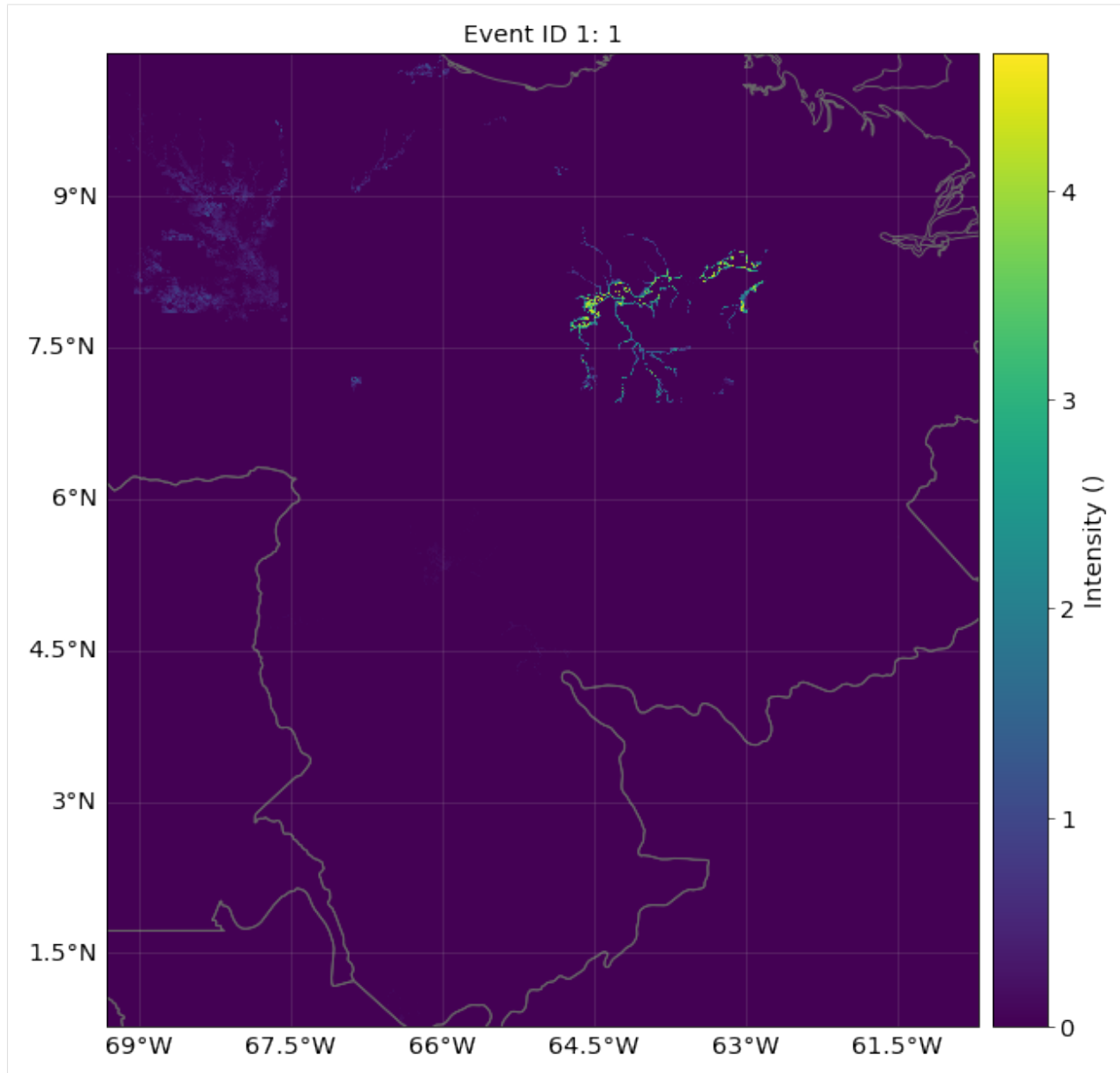
```
[1]: %matplotlib inline
import numpy as np
from climada.hazard import Hazard
from climada.util.constants import HAZ_DEMO_FL

haz_ven = Hazard('FL')
# read intensity from raster file HAZ_DEMO_FL and set frequency for the contained event
haz_ven.set_raster([HAZ_DEMO_FL], attrs={'frequency': np.ones(1)/2})
haz_ven.check()

# The masked values of the raster are set to 0
# Sometimes the raster file does not contain all the information, as in this case the
↳ mask value -9999
# We mask it manually and plot it using plot_intensity()
haz_ven.intensity[haz_ven.intensity == -9999] = 0
haz_ven.plot_intensity(1, smooth=False) # if smooth=True (default value) is used, the
↳ computation time might increase

# per default the following attributes have been set
print('event_id: ', haz_ven.event_id)
print('event_name: ', haz_ven.event_name)
print('date: ', haz_ven.date)
print('frequency: ', haz_ven.frequency)
print('orig: ', haz_ven.orig)
print('min, max fraction: ', haz_ven.fraction.min(), haz_ven.fraction.max())

2021-06-04 17:07:27,902 - climada.util.coordinates - INFO - Reading /Users/
↳ zeliestalhanske/climada/demo/data/SC22000_VE__M1.grd.gz
event_id: [1]
event_name: ['1']
date: [1.]
frequency: [0.5]
orig: [ True]
min, max fraction: 0.0 1.0
```

**EXERCISE:**

1. Read raster data in EPSG 2201 Coordinate Reference System (CRS)
2. Read raster data in its given CRS and transform it to the affine transformation `Affine(0.0090000000000000341, 0.0, -69.33714959699981, 0.0, -0.0090000000000000341, 10.42822096697894)`, `height=500`, `width=501`)
3. Read raster data in window `Window(10, 10, 20, 30)`

[2]: *# Put your code here*

```
[3]: # Solution:

# 1. The CRS can be reprojected using dst_crs option
haz = Hazard('FL')
haz.set_raster([HAZ_DEMO_FL], dst_crs={'init':'epsg:2201'})
haz.check()
print('\n Solution 1:')
print('centroids CRS:', haz.centroids.crs)
print('raster info:', haz.centroids.meta)

# 2. Transformations of the coordinates can be set using the transform option and Affine
from rasterio import Affine
haz = Hazard('FL')
haz.set_raster([HAZ_DEMO_FL], transform=Affine(0.0090000000000000341, 0.0, -69.
↪ 33714959699981, \
                                0.0, -0.0090000000000000341, 10.
↪ 42822096697894), height=500, width=501)
haz.check()
print('\n Solution 2:')
print('raster info:', haz.centroids.meta)
print('intensity size:', haz.intensity.shape)

# 3. A partial part of the raster can be loaded using the window or geometry
from rasterio.windows import Window
haz = Hazard('FL')
haz.set_raster([HAZ_DEMO_FL], window=Window(10, 10, 20, 30))
haz.check()
print('\n Solution 3:')
print('raster info:', haz.centroids.meta)
print('intensity size:', haz.intensity.shape)

2021-06-04 17:07:36,789 - climada.util.coordinates - INFO - Reading /Users/
↪ zeliestalhanske/climada/demo/data/SC22000_VE__M1.grd.gz

Solution 1:
centroids CRS: {'init': 'epsg:2201'}
raster info: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.701410009187828e+38,
↪ 'width': 978, 'height': 1091, 'count': 1, 'crs': {'init': 'epsg:2201'}, 'transform':
↪ Affine(1011.5372910988809, 0.0, 1120744.5486664253,
        0.0, -1011.5372910988809, 1189133.7652687666)}
2021-06-04 17:07:40,873 - climada.util.coordinates - INFO - Reading /Users/
↪ zeliestalhanske/climada/demo/data/SC22000_VE__M1.grd.gz

Solution 2:
raster info: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.701410009187828e+38,
↪ 'width': 501, 'height': 500, 'count': 1, 'crs': CRS.from_epsg(4326), 'transform':
↪ Affine(0.0090000000000000341, 0.0, -69.33714959699981,
        0.0, -0.0090000000000000341, 10.42822096697894)}
intensity size: (1, 250500)
2021-06-04 17:07:44,486 - climada.util.coordinates - INFO - Reading /Users/
↪ zeliestalhanske/climada/demo/data/SC22000_VE__M1.grd.gz

Solution 3:
```

(continues on next page)

(continued from previous page)

```
raster info: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.701410009187828e+38,
→ 'width': 20, 'height': 30, 'count': 1, 'crs': CRS.from_epsg(4326), 'transform':
→ Affine(0.00900000000000000341, 0.0, -69.2471495969998,
0.0, -0.00900000000000000341, 10.338220966978936)}
intensity size: (1, 600)
```

Part 2: Read hazards from other data

- excel: Hazards can be read from Excel files following the template in `climada_python/data/system/hazard_template.xlsx` using the `read_excel()` method.
- MATLAB: Hazards generated with CLIMADA's MATLAB version (.mat format) can be read using `read_mat()`.
- vector data: Use Hazard's `set_vector()` to read shape data (all formats supported by `fiona`).
- hdf5: Hazards generated with the CLIMADA in Python (.h5 format) can be read using `read_hdf5()`.

```
[4]: from climada.hazard import Hazard
from climada.util import HAZ_DEMO_H5 # CLIMADA's Python file
# Hazard needs to know the acronym of the hazard type to be constructed!!! Use 'NA' if
→ not known.
haz_tc_fl = Hazard('TC')
haz_tc_fl.read_hdf5(HAZ_DEMO_H5) # Historic tropical cyclones in Florida from 1990 to
→ 2004
haz_tc_fl.check() # Use always the check() method to see if the hazard has been loaded
→ correctly
```

```
2021-06-04 17:07:44,529 - climada.hazard.base - INFO - Reading /Users/zeliestalhanske/
→ climada/demo/data/tc_fl_1990_2004.h5
```

```
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
→ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
→ <authority>:<code>' is the preferred initialization method. When making the change, be
→ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
→ #axis-order-changes-in-proj-6
return _prepare_from_string(" ".join(pjargs))
```

Part 3: Define hazards manually A Hazard can be defined by filling its values one by one, as follows:

```
[5]: # setting points
import numpy as np
from scipy import sparse

lat = np.array([26.933899, 26.957203, 26.783846, 26.645524, 26.897796, 26.925359, \
26.914768, 26.853491, 26.845099, 26.82651 , 26.842772, 26.825905, \
26.80465 , 26.788649, 26.704277, 26.71005 , 26.755412, 26.678449, \
26.725649, 26.720599, 26.71255 , 26.6649 , 26.664699, 26.663149, \
26.66875 , 26.638517, 26.59309 , 26.617449, 26.620079, 26.596795, \
26.577049, 26.524585, 26.524158, 26.523737, 26.520284, 26.547349, \
26.463399, 26.45905 , 26.45558 , 26.453699, 26.449999, 26.397299, \
26.4084 , 26.40875 , 26.379113, 26.3809 , 26.349068, 26.346349, \
26.348015, 26.347957])

lon = np.array([-80.128799, -80.098284, -80.748947, -80.550704, -80.596929, \
```

(continues on next page)

(continued from previous page)

```

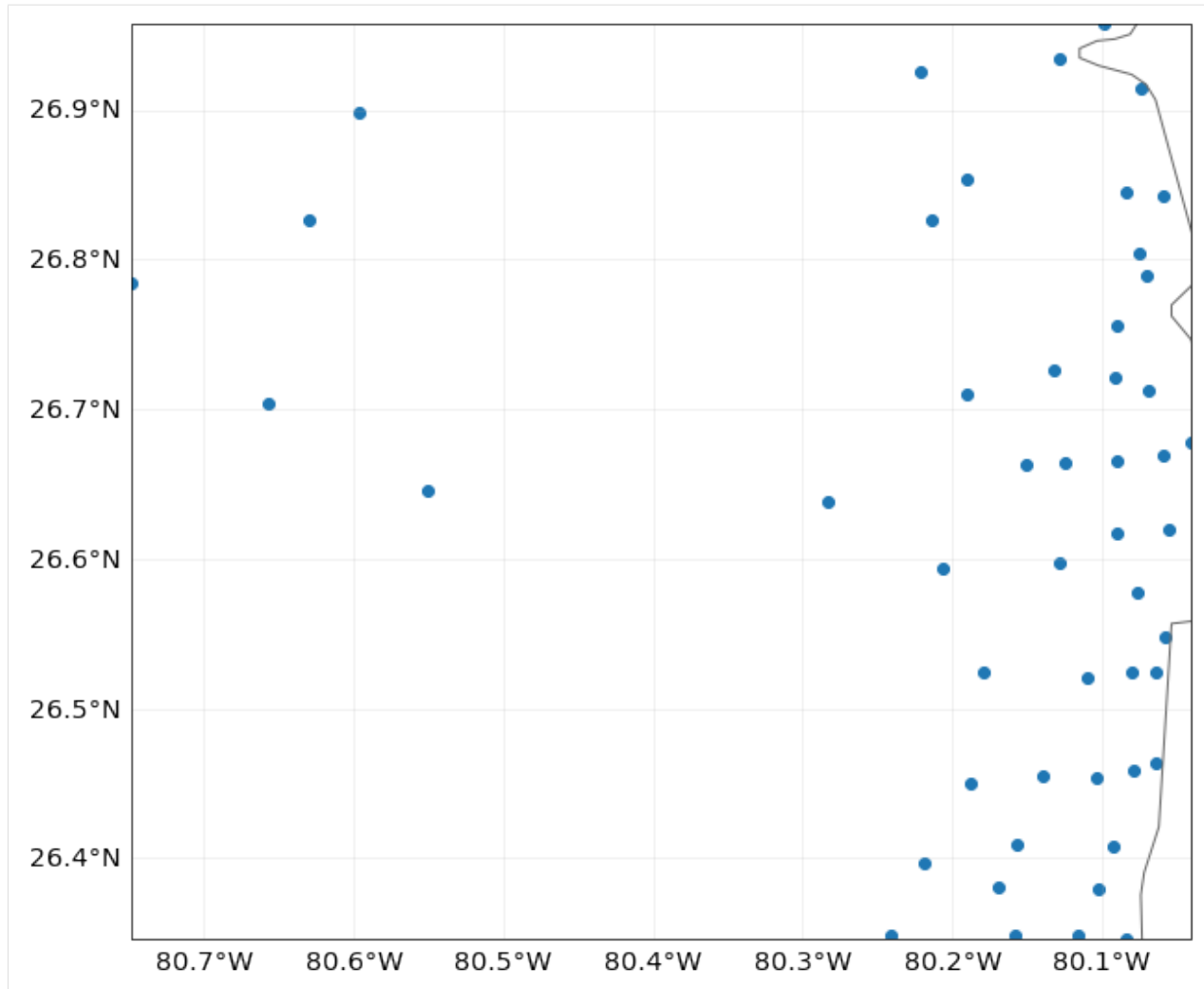
-80.220966, -80.07466 , -80.190281, -80.083904, -80.213493, \
-80.0591 , -80.630096, -80.075301, -80.069885, -80.656841, \
-80.190085, -80.08955 , -80.041179, -80.1324 , -80.091746, \
-80.068579, -80.090698, -80.1254 , -80.151401, -80.058749, \
-80.283371, -80.206901, -80.090649, -80.055001, -80.128711, \
-80.076435, -80.080105, -80.06398 , -80.178973, -80.110519, \
-80.057701, -80.064251, -80.07875 , -80.139247, -80.104316, \
-80.188545, -80.21902 , -80.092391, -80.1575 , -80.102028, \
-80.16885 , -80.116401, -80.08385 , -80.241305, -80.158855])

n_cen = lon.size # number of centroids
n_ev = 10 # number of events

haz = Hazard('TC')
haz.centroids.set_lat_lon(lat, lon) # default crs used
haz.intensity = sparse.csr_matrix(np.random.random((n_ev, n_cen)))
haz.units = 'm'
haz.event_id = np.arange(n_ev, dtype=int)
haz.event_name = ['ev_12', 'ev_21', 'Maria', 'ev_35', 'Irma', 'ev_16', 'ev_15', 'Edgar',
↳ 'ev_1', 'ev_9']
haz.date = [721166, 734447, 734447, 734447, 721167, 721166, 721167, 721200, 721166,
↳ 721166]
haz.orig = np.zeros(n_ev, bool)
haz.frequency = np.ones(n_ev)/n_ev
haz.fraction = haz.intensity.copy()
haz.fraction.data.fill(1)
haz.check()
haz.centroids.plot()

```

[5]: <GeoAxesSubplot:>



```
[6]: # setting raster
import numpy as np
from scipy import sparse

# raster info:
# border upper left corner (of the pixel, not of the center of the pixel)
xf_lat = 22
xo_lon = -72
# resolution in lat and lon
d_lat = -0.5 # negative because starting in upper corner
d_lon = 0.5 # same step as d_lat
# number of points
n_lat = 50
n_lon = 40

n_ev = 10 # number of events

haz = Hazard('TC')
haz.centroids.set_raster_from_pix_bounds(xf_lat, xo_lon, d_lat, d_lon, n_lat, n_lon) #
↳ default crs used
```

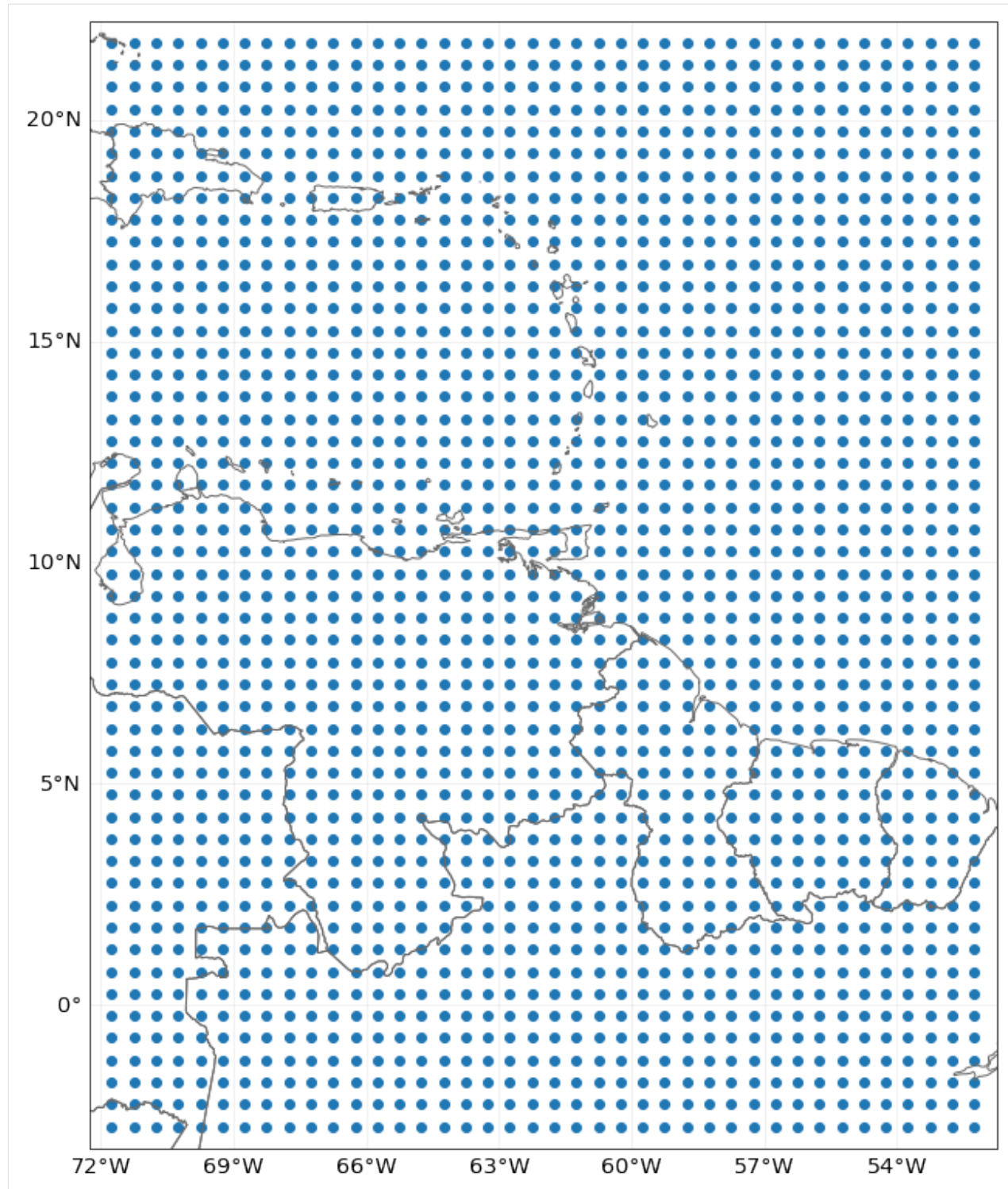
(continues on next page)

(continued from previous page)

```
haz.intensity = sparse.csr_matrix(np.random.random((n_ev, haz.centroids.size)))
haz.units = 'm'
haz.event_id = np.arange(n_ev, dtype=int)
haz.event_name = ['ev_12', 'ev_21', 'Maria', 'ev_35', 'Irma', 'ev_16', 'ev_15', 'Edgar',
↳ 'ev_1', 'ev_9']
haz.date = [721166, 734447, 734447, 734447, 721167, 721166, 721167, 721200, 721166,
↳ 721166]
haz.orig = np.zeros(n_ev, bool)
haz.frequency = np.ones(n_ev)/n_ev
haz.fraction = haz.intensity.copy()
haz.fraction.data.fill(1)
haz.check()
print('Check centroids borders:', haz.centroids.total_bounds)
haz.centroids.plot()

# using set_raster_from_pnt_bounds, the bounds refer to the bounds of the center of the
↳ pixel
left, bottom, right, top = xo_lon, -3.0, -52.0, xf_lat
haz.centroids.set_raster_from_pnt_bounds((left, bottom, right, top), 0.5) # default crs
↳ used
print('Check centroids borders:', haz.centroids.total_bounds)

Check centroids borders: (-72.0, -3.0, -52.0, 22.0)
Check centroids borders: (-72.25, -3.25, -51.75, 22.25)
```



Part 4: Analyse Hazards

The following methods can be used to analyse the data in Hazard:

- `calc_year_set()` method returns a dictionary with all the historical (not synthetic) event ids that happened at each year.

- `get_event_date()` returns strings of dates in ISO format.
- To obtain the relation between event ids and event names, two methods can be used `get_event_name()` and `get_event_id()`.

Other methods to handle one or several Hazards are: - the property `size` returns the number of events contained. - `append()` is used to expand events with data from another Hazard (and same centroids). - `select()` returns a new hazard with the selected region, date and/or synthetic or historical filter. - `remove_duplicates()` removes events with same name and date. - `local_exceedance_inten()` returns a matrix with the exceedance frequency at every frequency and provided return periods. This is the one used in `plot_rp_intensity()`. - `reproject_raster()`, `reproject_vector()`, `raster_to_vector()`, `vector_to_raster()` are methods to change centroids' CRS and between raster and vector data.

Centroids methods: - centroids properties such as area per pixel, distance to coast, country ISO code, on land mask or elevation are available through different `set_XX()` methods. - `set_lat_lon_to_meta()` computes the raster meta dictionary from present lat and lon. `set_meta_to_lat_lon()` computes lat and lon of the center of the pixels described in attribute meta. The raster meta information contains at least: width, height, crs and transform data (use `help(Centroids)` for more info). Using raster centroids can increase computing performance for several computations. - when using lats and lons (vector data) the `geopandas.GeoSeries` geometry attribute contains the CRS information and can be filled with point shapes to perform different computation. The geometry points can be then released using `empty_geometry_points()`.

EXERCISE:

Using the previous hazard `haz_tc_fl` answer these questions: 1. How many synthetic events are contained? 2. Generate a hazard with historical hurricanes occurring between 1995 and 2001. 3. How many historical hurricanes occurred in 1999? Which was the year with most hurricanes between 1995 and 2001? 4. What is the number of centroids with distance to coast smaller than 1km?

```
[7]: # Put your code here:
```

```
[8]: #help(hist_tc.centroids)
```

```
[9]: # SOLUTION:
```

```
# 1.How many synthetic events are contained?
print('Number of total events:', haz_tc_fl.size)
print('Number of synthetic events:', np.logical_not(haz_tc_fl.orig).astype(int).sum())

# 2. Generate a hazard with historical hurricanes occurring between 1995 and 2001.
hist_tc = haz_tc_fl.select(date=('1995-01-01', '2001-12-31'), orig=True)
print('Number of historical events between 1995 and 2001:', hist_tc.size)

# 3. How many historical hurricanes occurred in 1999? Which was the year with most_
↳hurricanes between 1995 and 2001?
ev_per_year = hist_tc.calc_year_set() # events ids per year
print('Number of events in 1999:', ev_per_year[1999].size)
max_year = 1995
max_ev = ev_per_year[1995].size
for year, ev in ev_per_year.items():
    if ev.size > max_ev:
        max_year = year
```

(continues on next page)

(continued from previous page)

```

print('Year with most hurricanes between 1995 and 2001:', max_year)

# 4. What is the number of centroids with distance to coast smaller than 1km?
hist_tc.centroids.set_dist_coast()
num_cen_coast = np.argwhere(hist_tc.centroids.dist_coast < 1000).size
print('Number of centroids close to coast: ', num_cen_coast)

Number of total events: 216
Number of synthetic events: 0
Number of historical events between 1995 and 2001: 109
Number of events in 1999: 16
Year with most hurricanes between 1995 and 2001: 1995
2021-06-04 17:07:50,281 - climada.hazard.centroids.centri - INFO - Convert centroids to
↳ GeoSeries of Point shapes.
2021-06-04 17:07:51,542 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617 (1/
↳ 2)

/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
    return _prepare_from_string(" ".join(pjargs))
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
    return _prepare_from_string(" ".join(pjargs))

2021-06-04 17:07:53,271 - climada.util.coordinates - INFO - dist_to_coast: UTM 32618 (2/
↳ 2)

/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
    return _prepare_from_string(" ".join(pjargs))
/Users/zeliestalhanske/miniconda3/envs/climada_env/lib/python3.8/site-packages/pyproj/
↳ crs/crs.py:53: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '
↳ <authority>:<code>' is the preferred initialization method. When making the change, be
↳ mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html
↳ #axis-order-changes-in-proj-6
    return _prepare_from_string(" ".join(pjargs))

Number of centroids close to coast: 41

```

Part 5: Visualize Hazards

There are three different plot functions: `plot_intensity()`, `plot_fraction()` and `plot_rp_intensity()`. Depending on the inputs, different properties can be visualized. Check the documentation of the functions:

```

[10]: help(haz_tc_fl.plot_intensity)
      help(haz_tc_fl.plot_rp_intensity)

```

Help on method plot_intensity in module climada.hazard.base:

plot_intensity(event=None, centr=None, smooth=True, axis=None, adapt_fontsize=True,
 ↳**kwargs) method of climada.hazard.base.Hazard instance
 Plot intensity values for a selected event or centroid.

Parameters:

event (int or str, optional): If event > 0, plot intensities of event with id = event. If event = 0, plot maximum intensity in each centroid. If event < 0, plot abs(event)-largest event. If event is string, plot events with that name.
 centr (int or tuple, optional): If centr > 0, plot intensity of all events at centroid with id = centr. If centr = 0, plot maximum intensity of each event. If centr < 0, plot abs(centr)-largest centroid where higher intensities are reached. If tuple with (lat, lon) plot intensity of nearest centroid.
 smooth (bool, optional): Rescale data to RESOLUTIONxRESOLUTION pixels (see
 ↳constant in module `climada.util.plot`)
 axis (matplotlib.axes._subplots.AxesSubplot, optional): axis to use
 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 (optional): arguments for pcolormesh matplotlib function used in event plots or for plot function used in centroids plots

Returns:

matplotlib.axes._subplots.AxesSubplot

Raises:

ValueError

Help on method plot_rp_intensity in module climada.hazard.base:

plot_rp_intensity(return_periods=(25, 50, 100, 250), smooth=True, axis=None, figsize=(9,
 ↳13), adapt_fontsize=True, **kwargs) method of climada.hazard.base.Hazard instance
 Compute and plot hazard exceedance intensity maps for different return periods. Calls local_exceedance_inten.

Parameters:

return_periods (tuple(int), optional): return periods to consider
 smooth (bool, optional): smooth plot to plot.RESOLUTIONxplot.RESOLUTION
 axis (matplotlib.axes._subplots.AxesSubplot, optional): axis to use
 figsize (tuple, optional): figure size for plt.subplots
 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 (optional): arguments for pcolormesh matplotlib function used in event plots

(continues on next page)

(continued from previous page)

Returns:

```
matplotlib.axes._subplots.AxesSubplot,
np.ndarray (return_periods.size x num_centroids)
```

```
[11]: # 1. intensities of the largest event (defined as greater sum of intensities):
# all events:
haz_tc_fl.plot_intensity(event=-1) # largest historical event: 1992230N11325 hurricane_
↳ ANDREW

# 2. maximum intensities at each centroid:
haz_tc_fl.plot_intensity(event=0)

# 3. intensities of hurricane 1998295N12284:
haz_tc_fl.plot_intensity(event='1998295N12284', cmap='BuGn') # setting color map

# 4. tropical cyclone intensities maps for the return periods [10, 50, 75, 100]
_, res = haz_tc_fl.plot_rp_intensity([10, 50, 75, 100])

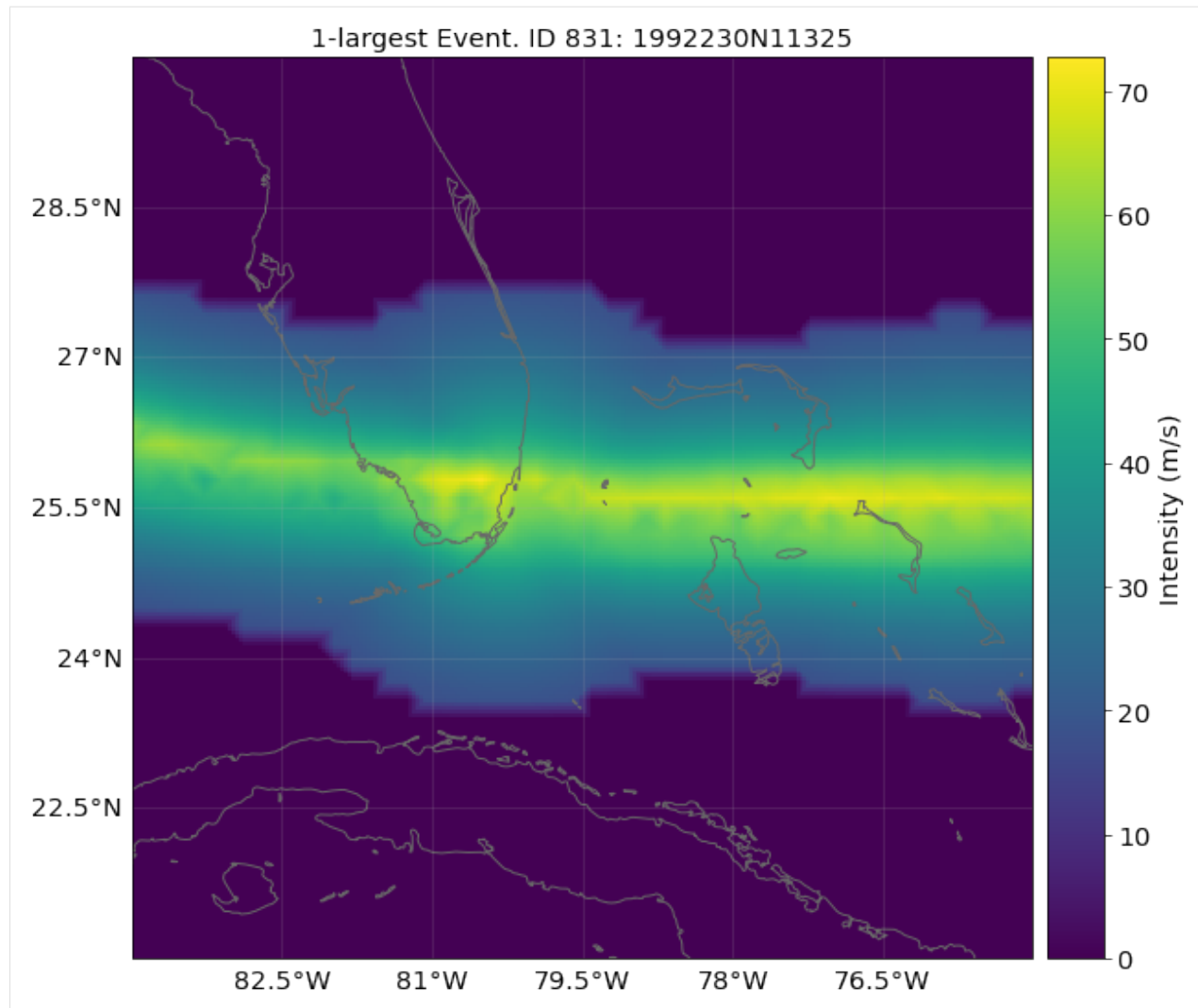
# 5. intensities of all the events in centroid with id 50
haz_tc_fl.plot_intensity(centr=50)

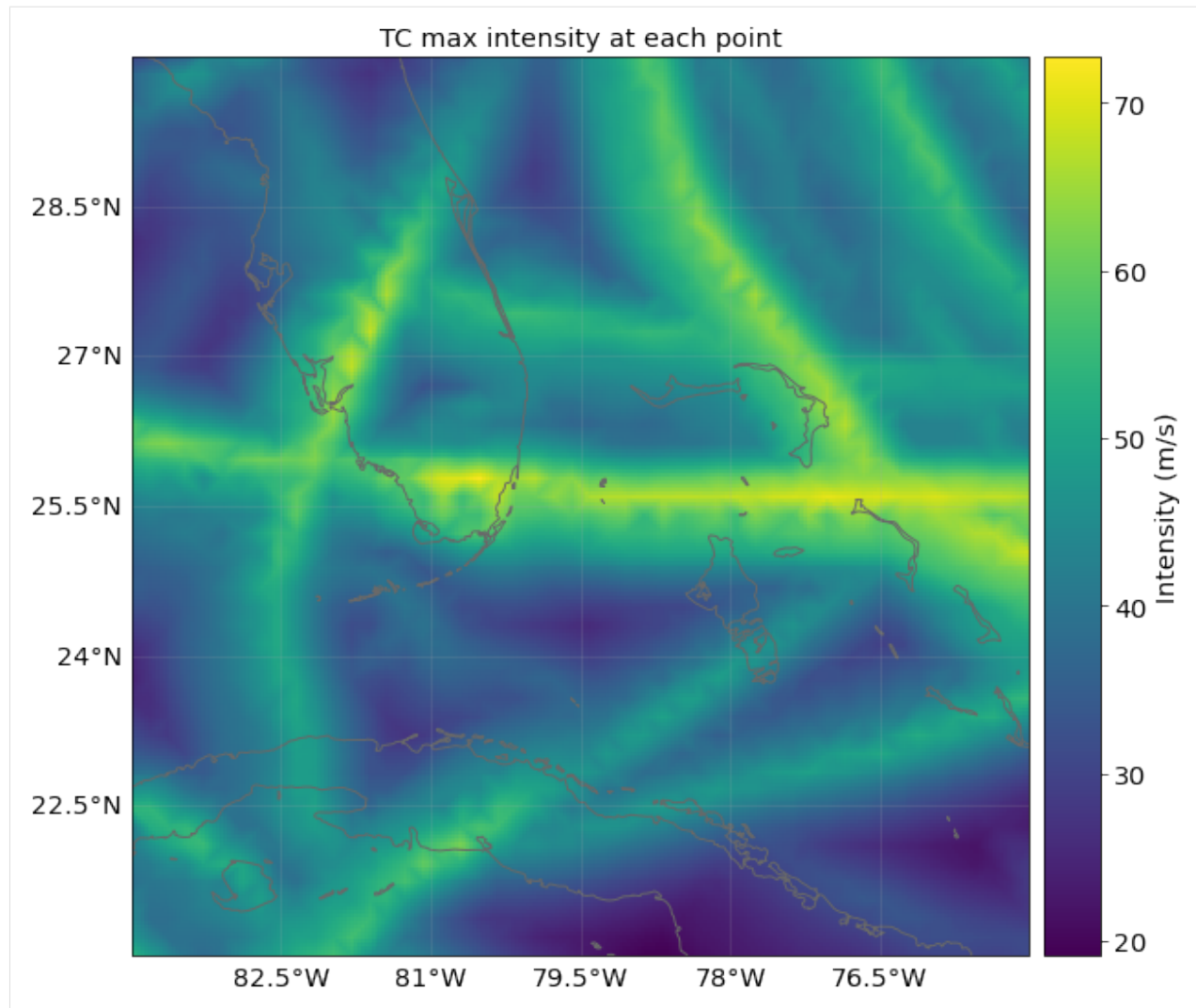
# 6. intensities of all the events in centroid closest to lat, lon = (26.5, -81)
haz_tc_fl.plot_intensity(centr=(26.5, -81));

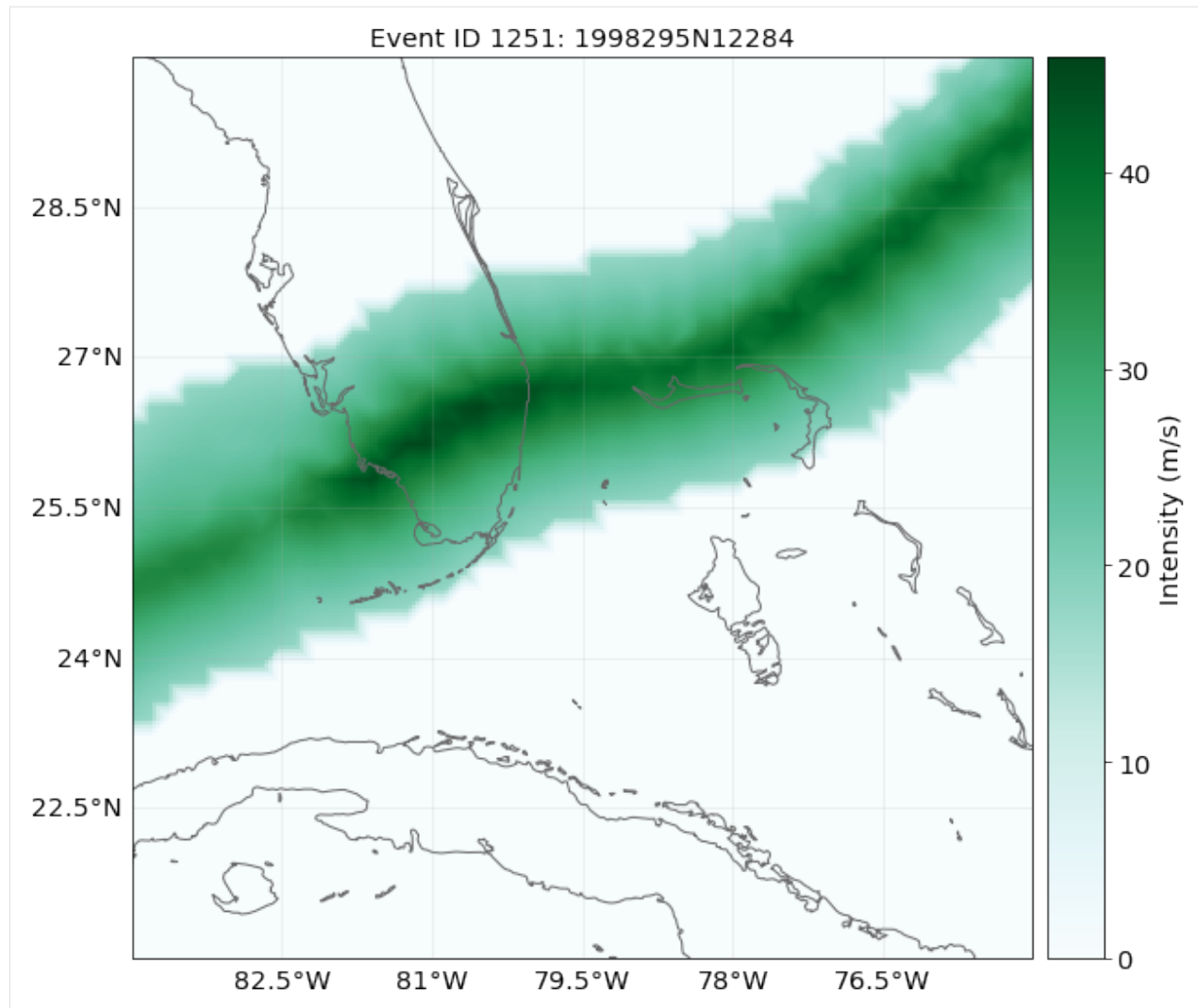
2021-06-04 17:07:56,644 - climada.hazard.base - INFO - Computing exceedance intensitiy_
↳ map for return periods: [ 10  50  75 100]
2021-06-04 17:07:57,198 - climada.hazard.base - WARNING - Exceedance intensitiy values_
↳ below 0 are set to 0. Reason: no negative intensity values were_
↳ found in hazard.

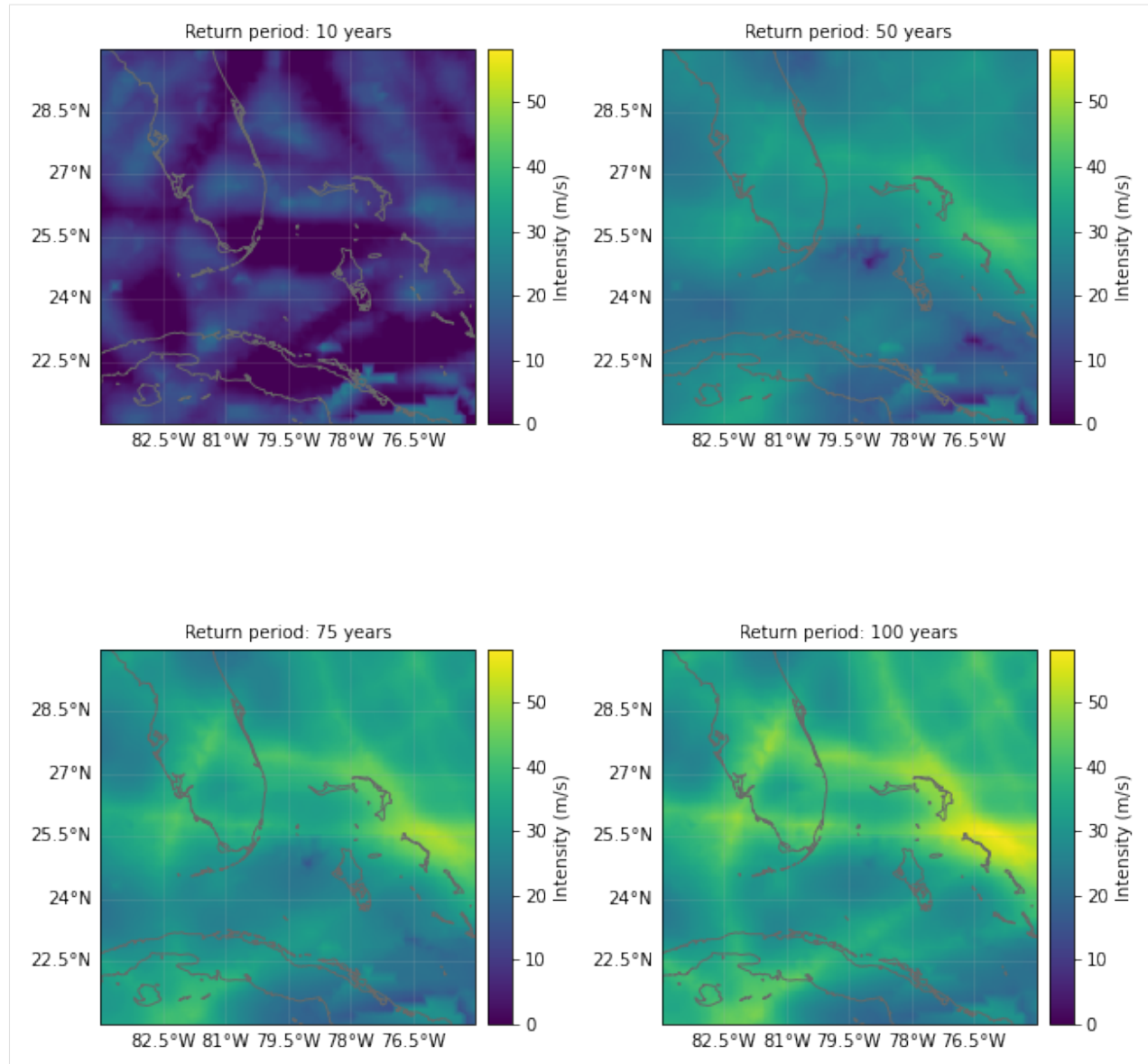
/Users/zeliestalhanske/python_projects/climada_python/climada/hazard/centroids/centr.py:
↳ 571: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely_
↳ incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before_
↳ this operation.

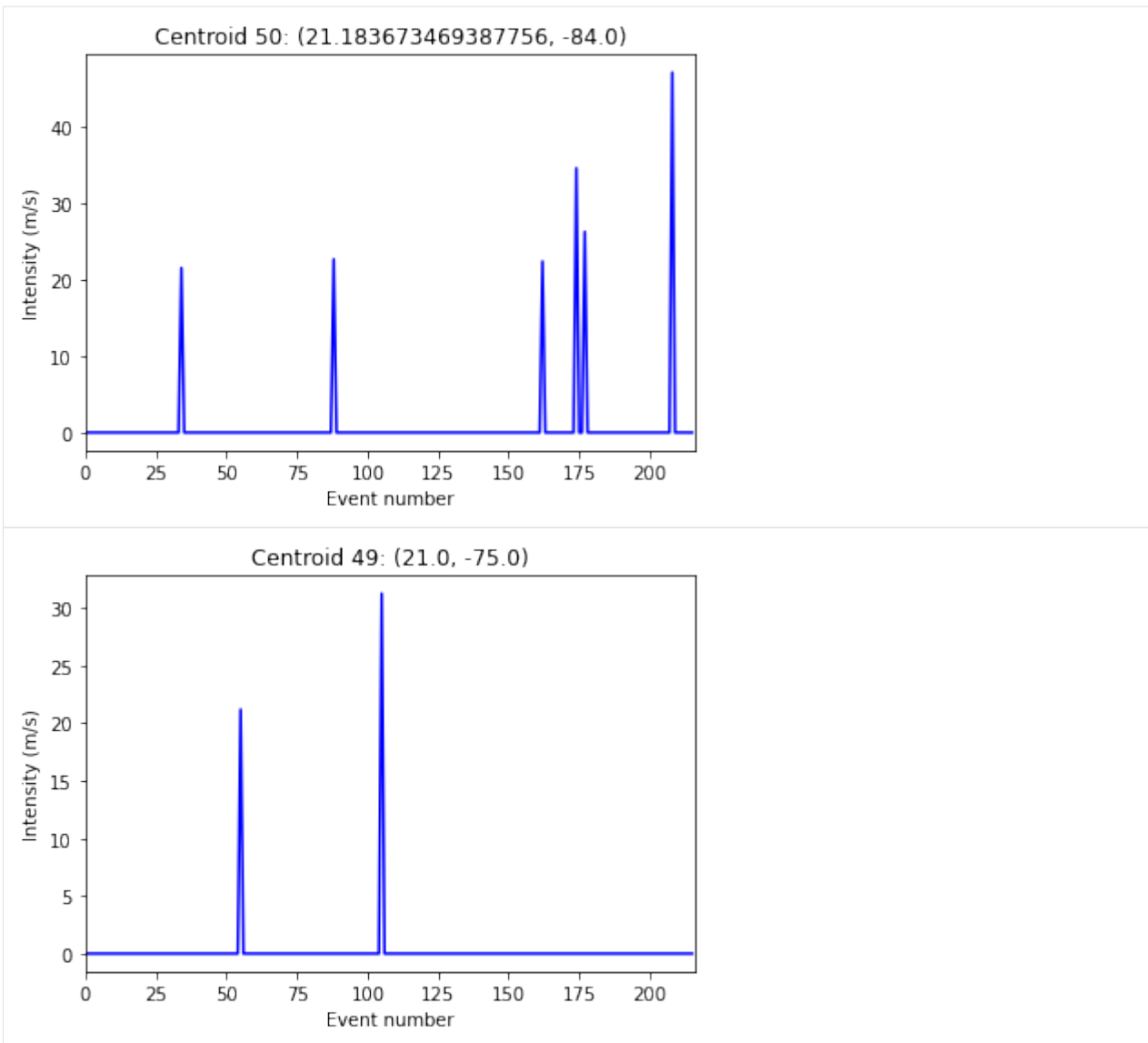
close_idx = self.geometry.distance(Point(x_lon, y_lat)).values.argmin()
```











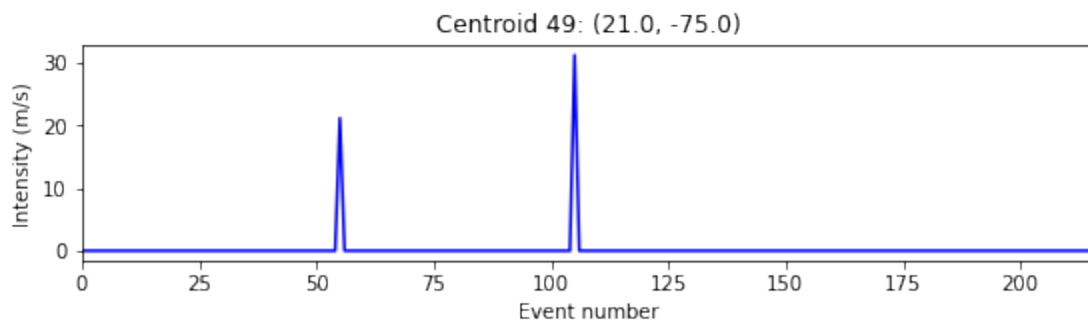
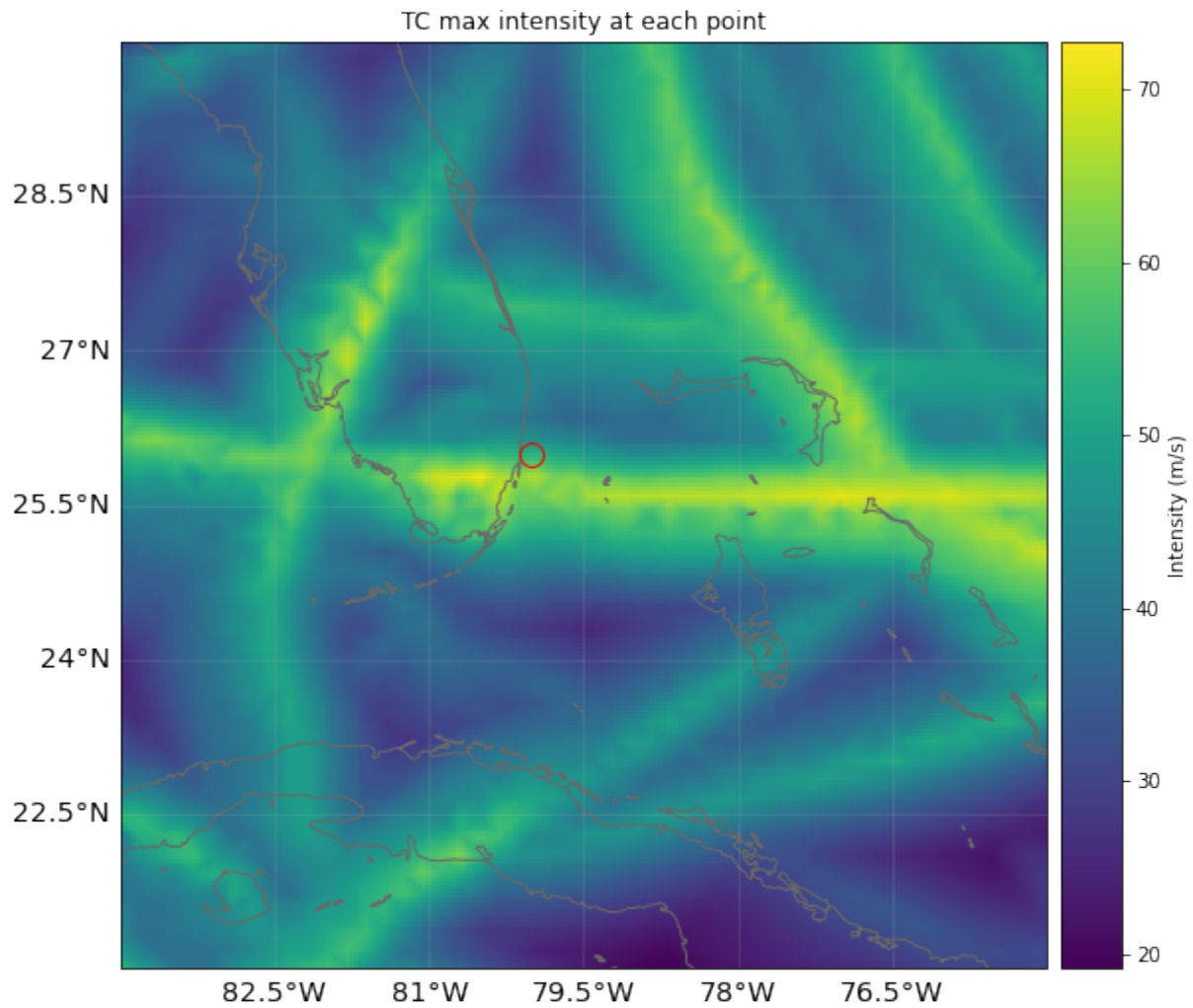
```
[12]: # 7. one figure with two plots: maximum intensities and selected centroid with all
      ↪ intensities:
      from climada.util.plot import make_map
      import matplotlib.pyplot as plt
      plt.ioff()
      fig, ax1, fontsize = make_map(1) # map
      ax2 = fig.add_subplot(2, 1, 2) # add regular axes
      haz_tc_fl.plot_intensity(axis=ax1, event=0) # plot original resolution
      ax1.plot(-80, 26, 'or', mfc='none', markersize=12)
      haz_tc_fl.plot_intensity(axis=ax2, centr=(26, -80))
      fig.subplots_adjust(hspace=6.5)

/Users/zeliestalhanske/python_projects/climada_python/climada/hazard/centroids/centr.py:
↪ 571: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely
↪ incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
↪ this operation.
```

(continues on next page)

(continued from previous page)

```
close_idx = self.geometry.distance(Point(x_lon, y_lat)).values.argmax()
```



```
## Part 6: Write (=save) hazards
```

Hazards can be written and read in hdf5 format as follows:

```
[13]: haz_tc_fl.write_hdf5('results/haz_tc_fl.h5')

haz = Hazard('TC')
haz.read_hdf5('results/haz_tc_fl.h5')
haz.check()

2021-06-04 17:08:17,623 - climada.hazard.base - INFO - Writing results/haz_tc_fl.h5
2021-06-04 17:08:17,707 - climada.hazard.base - INFO - Reading results/haz_tc_fl.h5
```

GeoTiff data is generated using `write_raster()`:

```
[14]: haz_ven.write_raster('results/haz_ven.tif') # each event is a band of the tif file

2021-06-04 17:08:17,739 - climada.util.coordinates - INFO - Writting results/haz_ven.tif
```

Pickle will work as well:

```
[15]: from climada.util.save import save
# this generates a results folder in the current path and stores the output there
save('tutorial_haz_tc_fl.p', haz_tc_fl)

2021-06-04 17:08:17,817 - climada.util.save - INFO - Written file /Users/zeliestalhanske/
↳python_projects/climada_python/doc/tutorial/results/tutorial_haz_tc_fl.p
```

```
[ ]:
```

5.9 Hazard: Tropical cyclones

Tropical cyclones tracks are gathered in the class `TCTracks` and then provided to the hazard `TropCyclone` which computes the wind gusts at each centroid. `TropCyclone` inherits from `Hazard` and has an associated hazard type `TC`.

5.9.1 What do tropical cyclones look like in CLIMADA?

`TCTracks` reads and handles historical tropical cyclone tracks of the [IBTrACS](#) repository or synthetic tropical cyclone tracks simulated using fully statistical or coupled statistical-dynamical modeling approaches. It also generates synthetic tracks from the historical ones using Wiener processes.

The tracks are stored in the attribute data, which is a list of `xarray`'s `Dataset` (see `xarray.Dataset`). Each `Dataset` contains the following variables:

Coordinates
time
latitude
longitude

Descriptive variables
time_step
radius_max_wind
max_sustained_wind
central_pressure
environmental_pressure

Attributes
max_sustained_wind_unit
central_pressure_unit
sid
name
orig_event_flag
data_provider
basin
id_no
category

How is this tutorial structured?

*****Part 1: Load TC tracks*****

*****a) Load TC tracks from historical records*****

*****b) Generate probabilistic events*****

*****c) ECMWF Forecast Tracks*****

*****d) Load TC tracks from other sources*****

*****Part 2: `TropCyclone()` class***** <#Part2>`__

*****a) Default hazard generation for tropical cyclones*****

*****b) Implementing climate change*****

*****c) Multiprocessing - improving performance for big computations*****

*****d) Making videos*****

Part 1: Load TC tracks

Records of historical TCs are very limited and therefore the database to study this natural hazard remains sparse. Only a small fraction of the TCs make landfall every year and reliable documentation of past TC landfalling events has just started in the 1950s (1980s - satellite era). The generation of synthetic storm tracks is an important tool to overcome this spatial and temporal limitation. Synthetic dataset are much larger and thus allow to estimate the risk of much rarer events. Here we show the most prominent tools in CLIMADA to load TC tracks from historical records *a)*, generate a probabilistic dataset thereof *b)*, and work with model simulations *c)*.

a) Load TC tracks from historical records

The best-track historical data from the International Best Track Archive for Climate Stewardship ([IBTrACS](#)) can easily be loaded into CLIMADA to study the historical records of TC events. The method `read_ibtracs_netcdf()` generates the Datasets for tracks selected by [IBTrACS](#) id, or by basin and year range. To achieve this, it downloads the first time the [IBTrACS data v4 in netcdf format](#) and stores it in `climada_python/data/system`. The tracks can be accessed later either using the attribute data or using `get_track()`, which allows to select tracks by its name or id. Use the method `append()` to extend the data list.

If you get an error downloading the IBTrACS data, try to manually access <ftp://eclipse.ncdc.noaa.gov/pub/ibtracs/v04r00/provisional/netcdf/>, connect as a *Guest* and copy the file IBTrACS.ALL.v04r00.nc to climada_python/data/system.

To visualize the tracks use `plot()`.

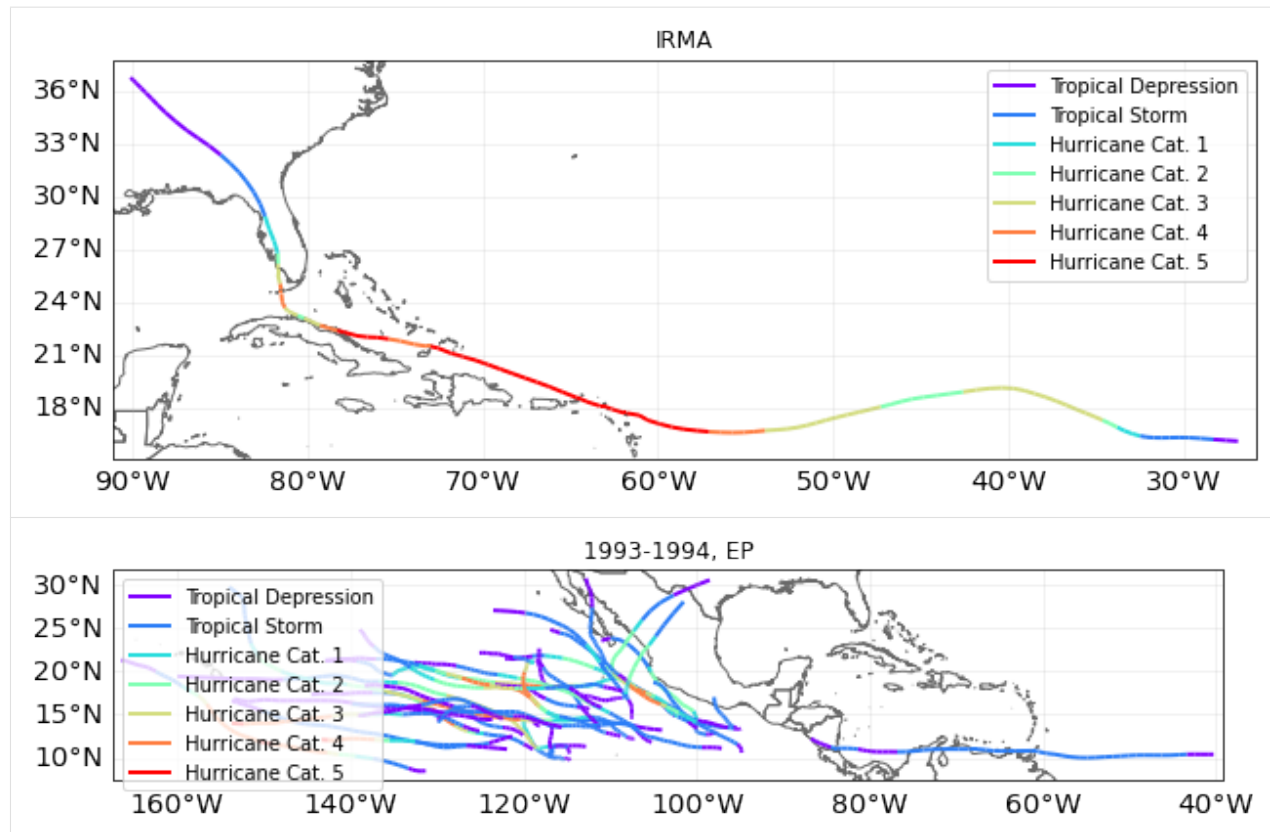
```
[1]: %matplotlib inline
from climada.hazard import TCTracks

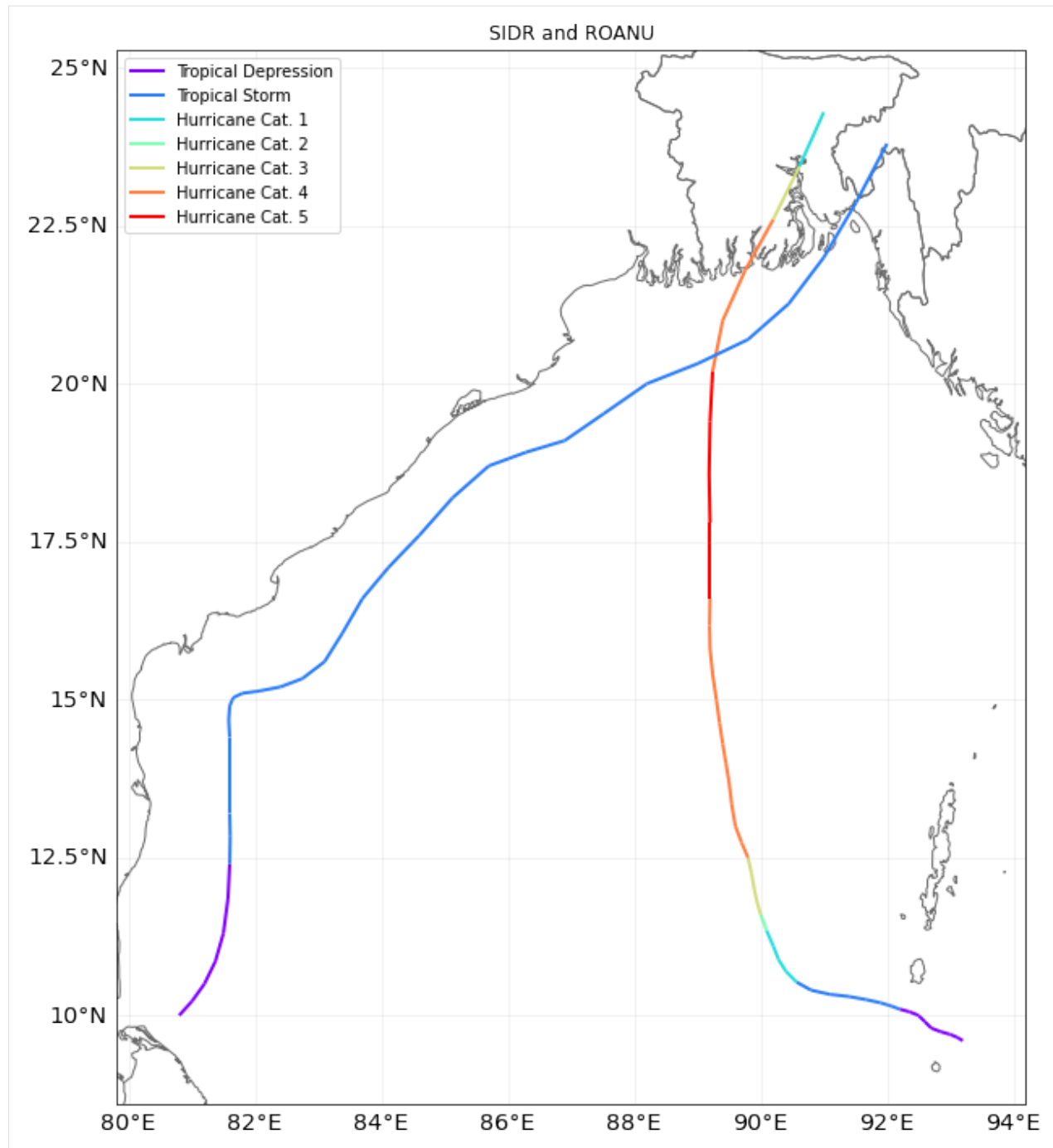
tr_irma = TCTracks()
tr_irma.read_ibtracs_netcdf(provider='usa', storm_id='2017242N16333') # IRMA 2017
ax = tr_irma.plot()
ax.set_title('IRMA') # set title

# other ibtracs selection options
from climada.hazard import TCTracks
sel_ibtracs = TCTracks()
# years 1993 and 1994 in basin EP.
# correct_pres ignores tracks with not enough data. For statistics (frequency of events),
# → these should be considered as well
sel_ibtracs.read_ibtracs_netcdf(provider='usa', year_range=(1993, 1994), basin='EP',
# → correct_pres=False)
print('Number of tracks:', sel_ibtracs.size)
ax = sel_ibtracs.plot()
ax.get_legend()._loc = 2 # correct legend location
ax.set_title('1993-1994, EP') # set title

track1 = TCTracks()
track1.read_ibtracs_netcdf(provider='usa', storm_id='2007314N10093') # SDR 2007
track2 = TCTracks()
track2.read_ibtracs_netcdf(provider='usa', storm_id='2016138N10081') # ROANU 2016
track1.append(track2.data) # put both tracks together
ax = track1.plot()
ax.get_legend()._loc = 2 # correct legend location
ax.set_title('SDR and ROANU'); # set title

2021-06-04 17:07:33,515 - climada.hazard.tc_tracks - INFO - Progress: 100%
2021-06-04 17:07:35,833 - climada.hazard.tc_tracks - WARNING - 19 storm events are
→discarded because no valid wind/pressure values have been found: 1993178N14265,
→1993221N12216, 1993223N07185, 1993246N16129, 1993263N11168, ...
2021-06-04 17:07:35,940 - climada.hazard.tc_tracks - INFO - Progress: 11%
2021-06-04 17:07:36,028 - climada.hazard.tc_tracks - INFO - Progress: 23%
2021-06-04 17:07:36,119 - climada.hazard.tc_tracks - INFO - Progress: 35%
2021-06-04 17:07:36,218 - climada.hazard.tc_tracks - INFO - Progress: 47%
2021-06-04 17:07:36,312 - climada.hazard.tc_tracks - INFO - Progress: 58%
2021-06-04 17:07:36,399 - climada.hazard.tc_tracks - INFO - Progress: 70%
2021-06-04 17:07:36,493 - climada.hazard.tc_tracks - INFO - Progress: 82%
2021-06-04 17:07:36,585 - climada.hazard.tc_tracks - INFO - Progress: 94%
2021-06-04 17:07:36,612 - climada.hazard.tc_tracks - INFO - Progress: 100%
Number of tracks: 33
2021-06-04 17:07:38,825 - climada.hazard.tc_tracks - INFO - Progress: 100%
2021-06-04 17:07:39,974 - climada.hazard.tc_tracks - INFO - Progress: 100%
```



```
[2]: tr_irma.get_track('2017242N16333')
```

```
[2]: <xarray.Dataset>
Dimensions:                (time: 123)
Coordinates:
  * time                    (time) datetime64[ns] 2017-08-30 ... 2017-09-13T1...
    lat                    (time) float32 16.1 16.15 16.2 ... 36.2 36.5 36.8
    lon                    (time) float32 -26.9 -27.59 -28.3 ... -89.79 -90.1
Data variables:
```

(continues on next page)

(continued from previous page)

```

time_step          (time) float64 3.0 3.0 3.0 3.0 ... 3.0 3.0 3.0 3.0
radius_max_wind    (time) float32 60.0 60.0 60.0 ... 60.0 60.0 60.0
radius_oci         (time) float32 180.0 180.0 180.0 ... 350.0 350.0
max_sustained_wind (time) float32 30.0 32.0 35.0 ... 15.0 15.0 15.0
central_pressure   (time) float32 1.008e+03 1.007e+03 ... 1.005e+03
environmental_pressure (time) float64 1.012e+03 1.012e+03 ... 1.008e+03
basin              (time) <U2 'NA' 'NA' 'NA' 'NA' ... 'NA' 'NA' 'NA'

```

Attributes:

```

max_sustained_wind_unit: kn
central_pressure_unit:  mb
name:                   IRMA
sid:                    2017242N16333
orig_event_flag:       True
data_provider:         ibtracs_usa
id_no:                 2017242016333.0
category:              5

```

b) Generate probabilistic events

Once tracks are present in TCTracks, one can generate synthetic tracks for each present track based on directed random walk. Note that the tracks should be interpolated to use the same timestep **before** generation of probabilistic events.

`calc_perturbed_trajectories()` generates an ensemble of “nb_synth_tracks” numbers of synthetic tracks is computed for every track. The methodology perturbs the tracks locations, and if decay is True it additionally includes decay of wind speed and central pressure drop after landfall. No other track parameter is perturbed.

```

[3]: # here we use tr_irma retrieved from IBTrACS with the function above
# select number of synthetic tracks (nb_synth_tracks) to generate per present tracks.
tr_irma.equal_timestep()
tr_irma.calc_perturbed_trajectories(nb_synth_tracks=5)
tr_irma.plot()
# see more configuration options (e.g. amplitude of max random starting point shift in_
↳ decimal degree; max_shift_ini)

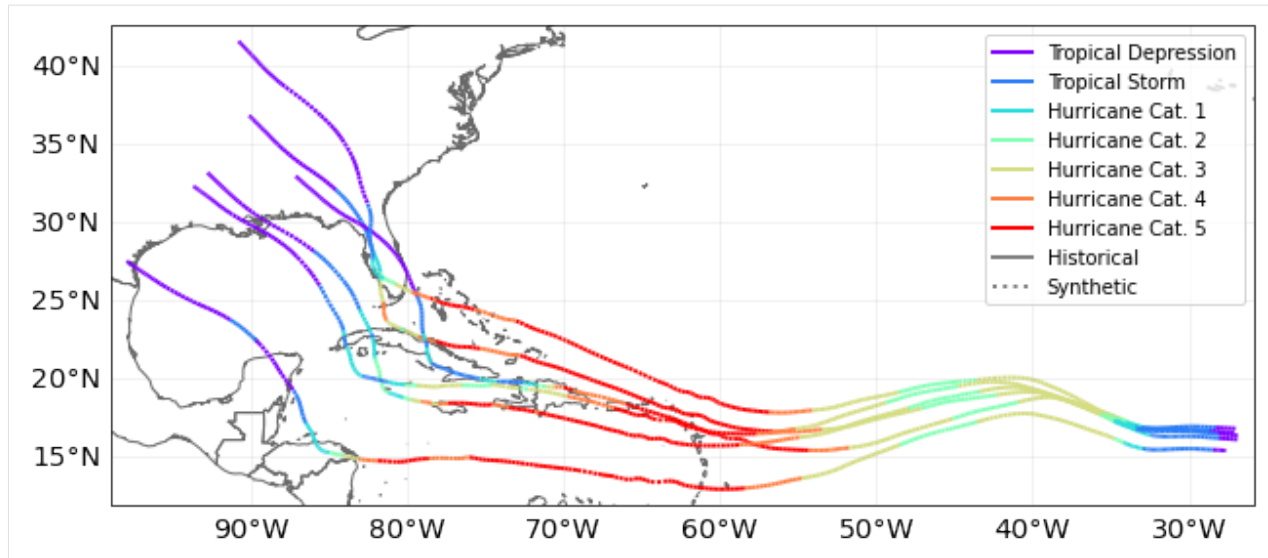
2021-06-04 17:07:49,459 - climada.hazard.tc_tracks - INFO - Interpolating 1 tracks to 1h_
↳ time steps.
2021-06-04 17:07:51,019 - climada.hazard.tc_tracks_synth - INFO - Computing 5 synthetic_
↳ tracks.

```

```

[3]: <GeoAxesSubplot:>

```



```
[4]: tr_irma.data[-1] # last synthetic track. notice the value of orig_event_flag and name
```

```
[4]: <xarray.Dataset>
Dimensions:
      (time: 349)
Coordinates:
      * time      (time) datetime64[ns] 2017-08-30 ... 2017-09-13T1...
        lon      (time) float64 -27.64 -27.8 -27.96 ... -97.81 -97.93
        lat      (time) float64 15.39 15.41 15.42 ... 27.41 27.49
Data variables:
    time_step      (time) float64 1.0 1.0 1.0 1.0 ... 1.0 1.0 1.0 1.0
    radius_max_wind (time) float64 60.0 60.0 60.0 ... 60.0 60.0 60.0
    radius_oci      (time) float64 180.0 180.0 180.0 ... 350.0 350.0
    max_sustained_wind (time) float64 30.0 30.67 31.33 ... 15.0 14.82 14.47
    central_pressure (time) float64 1.008e+03 1.008e+03 ... 1.004e+03
    environmental_pressure (time) float64 1.012e+03 1.012e+03 ... 1.008e+03
    basin          (time) <U2 'NA' 'NA' 'NA' 'NA' ... 'NA' 'NA' 'NA'
    on_land        (time) bool False False False ... False True True
    dist_since_lf  (time) float64 nan nan nan nan ... nan 7.605 22.71
Attributes:
    max_sustained_wind_unit:  kn
    central_pressure_unit:   mb
    name:                    IRMA_gen5
    sid:                     2017242N16333_gen5
    orig_event_flag:         False
    data_provider:           ibtracs_usa
    id_no:                   2017242016333.05
    category:                5
```

EXERCISE

Using the first synthetic track generated,

1. Which is the time frequency of the data?
2. Compute the maximum sustained wind for each day.

```
[5]: # Put your code here
```

```
[6]: # SOLUTION:
import numpy as np
# select the track
tc_syn = tr_irma.get_track('2017242N16333_gen1')

# 1. Which is the time frequency of the data?
# The values of a DataArray are numpy.arrays.
# The numpy.ediff1d computes the different between elements in an array
diff_time_ns = np.ediff1d(tc_syn.time)
diff_time_h = diff_time_ns.astype(int)/1000/1000/1000/60/60
print('Mean time frequency in hours:', diff_time_h.mean())
print('Std time frequency in hours:', diff_time_h.std())
print()

# 2. Compute the maximum sustained wind for each day.
print('Daily max sustained wind:', tc_syn.max_sustained_wind.groupby('time.day').max())

Mean time frequency in hours: 1.0
Std time frequency in hours: 0.0

Daily max sustained wind: <xarray.DataArray 'max_sustained_wind' (day: 15)>
array([[100.          , 100.          , 100.          , 123.33333333,
        155.          , 155.          , 150.          , 138.          ,
        66.86201366,  68.45663763,  39.45663763,   8.99807756,
         4.13170454,  54.          ,  99.          ]])
Coordinates:
  * day      (day) int64 1 2 3 4 5 6 7 8 9 10 11 12 13 30 31
```

c) ECMWF Forecast Tracks

ECMWF publishes tropical cyclone forecast tracks free of charge as part of the [WMO essentials](#). These tracks are detected automatically in the ENS and HRES models. The non-supervised nature of the model may lead to artefacts.

The `tc_fcast` trackset below inherits from `TCTracks`, but contains some additional metadata that follows ECMWF's definitions. Try plotting these tracks and compare them to the official [cones of uncertainty](#)! The example track at `tc_fcast.data[0]` shows the data structure.

```
[ ]: from climada.hazard import TCForecast

tc_fcast = TCForecast()
tc_fcast.fetch_ecmwf()
```

(continues on next page)

(continued from previous page)

```
print(tc_fcast.data[0])
```

d) Load TC tracks from other sources

In addition to the *historical records of TCs (IBTrACS)*, the *probabilistic extension* of these tracks, and the *ECMWF Forecast tracks*, CLIMADA also features functions to read in synthetic TC tracks from other sources. These include synthetic storm tracks from Kerry Emanuel's coupled statistical-dynamical model (Emanuel et al., 2006 as used in Geiger et al., 2016), synthetic storm tracks from a second coupled statistical-dynamical model (CHAZ) (as described in Lee et al., 2018), and synthetic storm tracks from a fully statistical model (STORM) Bloemendaal et al., 2020). However, these functions are partly under development and/or targeted at advanced users of CLIMADA in the context of very specific use cases. They are thus not covered in this tutorial.

Part 2: TropCyclone() class

The TropCyclone class is a derived class of *Hazard*. As such, it contains all the attributes and methods of a Hazard. Additionally, it comes with the constructor method `from_tracks` to model tropical cyclones from tracks contained in a TCTracks instance.

When setting tropical cyclones from tracks, the centroids where to map the wind gusts (the hazard intensity) can be provided. If no centroids are provided, the global centroids `GLB_NatID_grid_0360as_adv_2.mat` are used.

From the track properties the 1 min sustained peak gusts are computed in each centroid as the sum of a circular wind field (following Holland, 2008) and the translational wind speed that arises from the storm movement. We incorporate the decline of the translational component from the cyclone centre by multiplying it by an attenuation factor. See *CLIMADA v1* and references therein for more information.

a) Default hazard generation for tropical cyclones

```
[8]: from climada.hazard import Centroids, TropCyclone
```

```
# construct centroids
min_lat, max_lat, min_lon, max_lon = 16.99375, 21.95625, -72.48125, -61.66875
cent = Centroids()
cent.set_raster_from_pnt_bounds((min_lon, min_lat, max_lon, max_lat), res=0.12)
cent.check()
cent.plot()

# construct tropical cyclones
tc_irma = TropCyclone.from_tracks(tr_irma, centroids=cent)
# tc_irma = TropCyclone.from_tracks(tr_irma) # try without given centroids
tc_irma.check()
tc_irma.plot_intensity('2017242N16333') # IRMA
tc_irma.plot_intensity('2017242N16333_gen2') # IRMA's synthetic track 2
```

```
2021-06-04 17:10:50,049 - climada.hazard.centroids.cent - INFO - Convert centroids to
↳ GeoSeries of Point shapes.
```

```
2021-06-04 17:10:51,347 - climada.util.coordinates - INFO - dist_to_coast: UTM 32618 (1/
↳ 3)
```

```
2021-06-04 17:10:51,852 - climada.util.coordinates - INFO - dist_to_coast: UTM 32619 (2/
↳ 3)
```

```
2021-06-04 17:10:53,131 - climada.util.coordinates - INFO - dist_to_coast: UTM 32620 (3/
↳ 3)
```

```
2021-06-04 17:10:54,019 - climada.hazard.trop_cyclone - INFO - Mapping 6 tracks to 3822
↳ coastal centroids.
```

```
2021-06-04 17:10:54,294 - climada.hazard.trop_cyclone - INFO - Progress: 16%
```

(continues on next page)

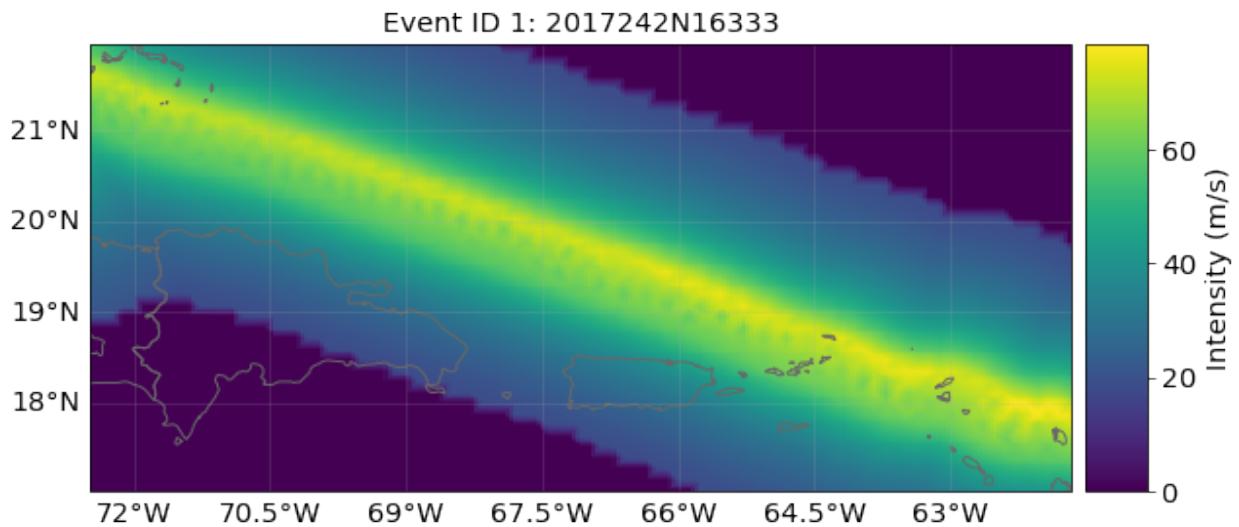
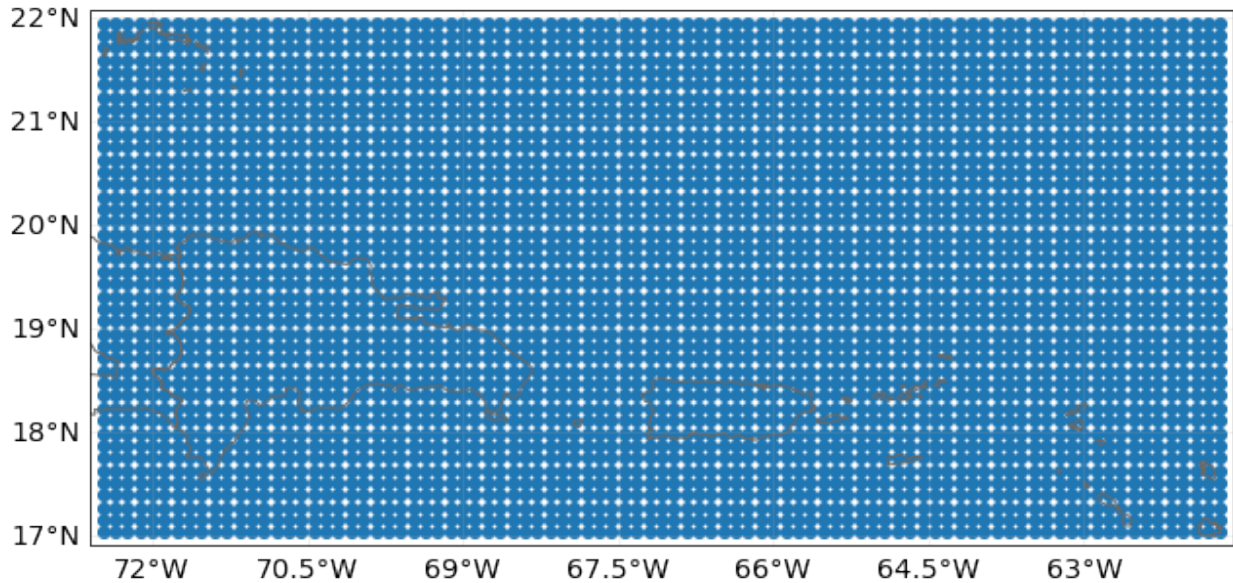
(continued from previous page)

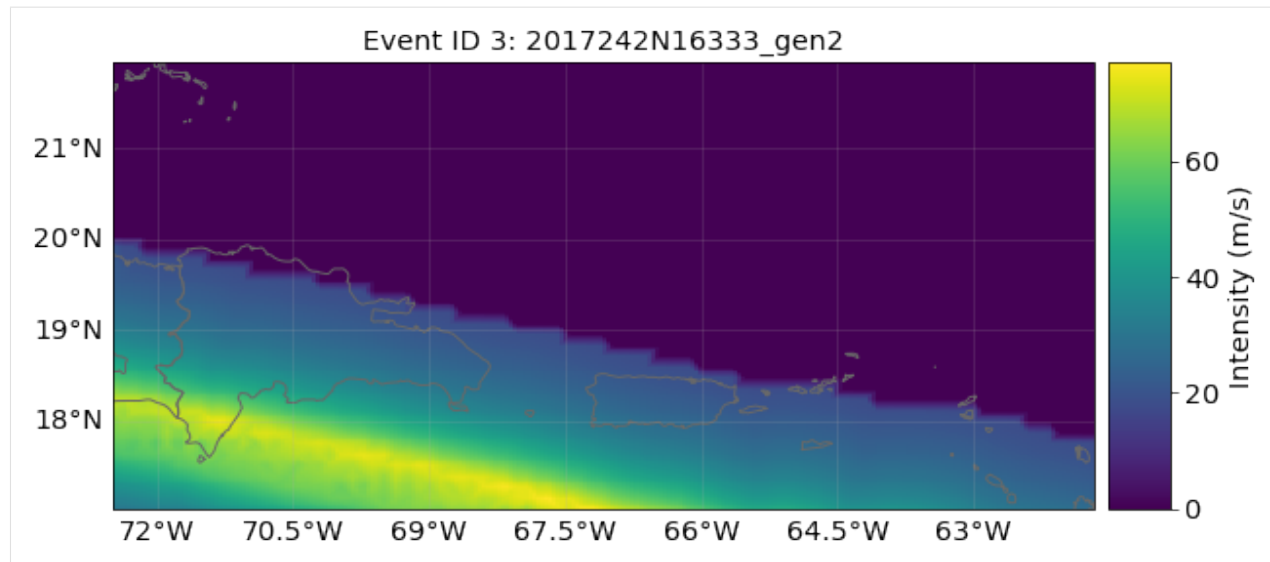
```

2021-06-04 17:10:54,586 - climada.hazard.trop_cyclone - INFO - Progress: 33%
2021-06-04 17:10:54,831 - climada.hazard.trop_cyclone - INFO - Progress: 50%
2021-06-04 17:10:55,109 - climada.hazard.trop_cyclone - INFO - Progress: 66%
2021-06-04 17:10:55,364 - climada.hazard.trop_cyclone - INFO - Progress: 83%
2021-06-04 17:10:55,524 - climada.hazard.trop_cyclone - INFO - Progress: 100%

```

[8]: <GeoAxesSubplot:title={'center':'Event ID 3: 2017242N16333_gen2'}>





b) Implementing climate change

`set_climate_scenario_knu` implements the changes on intensity and frequency due to climate change described in *Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios* of Knutson et al 2015. Other RCP scenarios are approximated from the RCP 4.5 values by interpolating them according to their relative radiative forcing.

[9]: `# an Irma event-like in 2055 under RCP 4.5:`

```
tc_irma = TropCyclone.from_tracks(tr_irma, centroids=cent)
tc_irma_cc = tc_irma.set_climate_scenario_knu(ref_year=2055, rcp_scenario=45)
tc_irma_cc.plot_intensity('2017242N16333')
```

```
2021-06-04 17:11:01,178 - climada.hazard.trop_cyclone - INFO - Mapping 6 tracks to 3822_
↪coastal centroids.
```

```
2021-06-04 17:11:01,509 - climada.hazard.trop_cyclone - INFO - Progress: 16%
```

```
2021-06-04 17:11:01,792 - climada.hazard.trop_cyclone - INFO - Progress: 33%
```

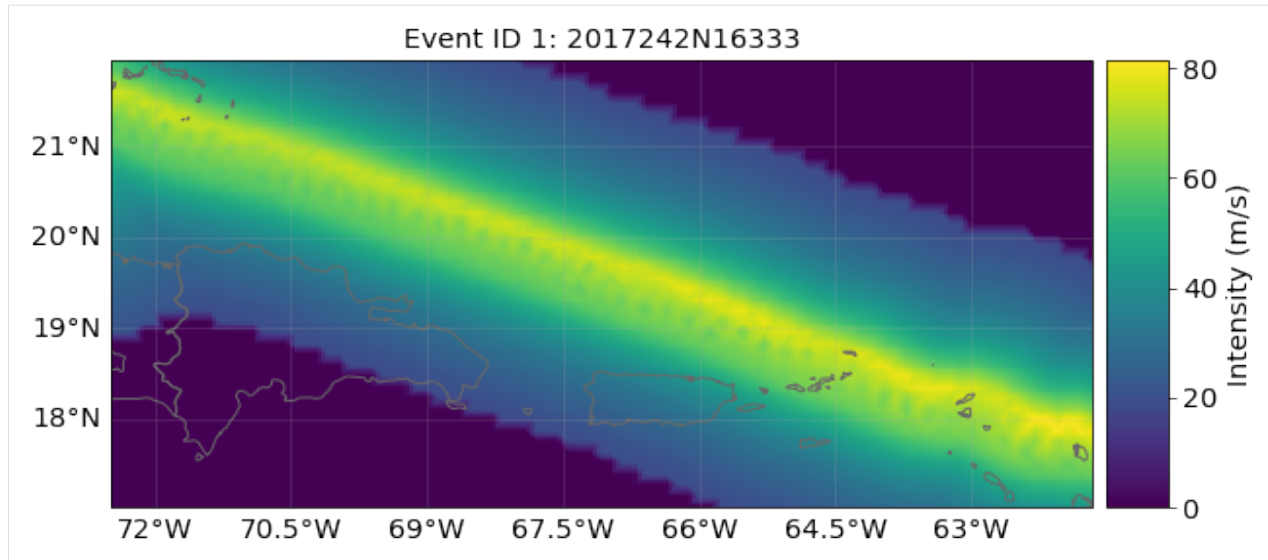
```
2021-06-04 17:11:02,068 - climada.hazard.trop_cyclone - INFO - Progress: 50%
```

```
2021-06-04 17:11:02,351 - climada.hazard.trop_cyclone - INFO - Progress: 66%
```

```
2021-06-04 17:11:02,651 - climada.hazard.trop_cyclone - INFO - Progress: 83%
```

```
2021-06-04 17:11:02,802 - climada.hazard.trop_cyclone - INFO - Progress: 100%
```

[9]: `<GeoAxesSubplot:title={'center': 'Event ID 1: 2017242N16333'}>`



Note: this method to implement climate change is simplified and does only take into account changes in TC frequency and intensity. However, how hurricane damage changes with climate remains challenging to assess. Records of hurricane damage exhibit widely fluctuating values because they depend on rare, landfalling events which are substantially more volatile than the underlying basin-wide TC characteristics. For more accurate future projections of how a warming climate might shape TC characteristics, there is a two-step process needed. First, the understanding of how climate change affects critical environmental factors (like SST, humidity, etc.) that shape TCs is required. Second, the means of simulating how these changes impact TC characteristics (such as intensity, frequency, etc.) are necessary. Statistical-dynamical models (Emanuel et al., 2006 and Lee et al., 2018) are physics-based and allow for such climate change studies. However, this goes beyond the scope of this tutorial.

c) Multiprocessing - improving performance for big computations

WARNING: Uncomment and execute these lines in a console, outside Jupyter Notebook. Multiprocessing is implemented in the tropical cyclones. Simply provide pool in the constructor. When dealing with a big amount of data, you might consider using it as follows:

```
[10]: #from climada.hazard import TCTracks, Centroids, TropCyclone
#from pathos.pools import ProcessPool as Pool

#pool = Pool() # start a pathos pool

#tc_track = TCTracks(pool) # provide the pool in the constructor
#tc_track.read_ibtracs_netcdf(provider='usa', year_range=(1992, 1994), basin='EP')
#tc_track.calc_perturbed_trajectories() # OPTIONAL: if you want to generate a
↳probabilistic set of TC tracks.
#tc_track.equal_timestep()

#lon_min, lat_min, lon_max, lat_max = -160, 10, -100, 36
#centr = Centroids()
#centr.set_raster_from_pnt_bounds((lon_min, lat_min, lon_max, lat_max), 0.1)

#tc_haz = TropCyclone.from_tracks(tc_track, centroids=centr, pool=pool) # provide the
↳pool in the constructor
#tc_haz.check()

#pool.close()
```

(continues on next page)

(continued from previous page)

`#pool.join()`

d) Making videos **WARNING:** Uncomment and execute these lines in a console, outside Jupyter Notebook.

Videos of a tropical cyclone hitting specific centroids are done automatically using the method `video_intensity()`.

```
[ ]: #lon_min, lat_min, lon_max, lat_max = -83.5, 24.4, -79.8, 29.6
#centr_video = Centroids()
#centr_video.set_raster_from_pnt_bounds((lon_min, lat_min, lon_max, lat_max), 0.04)
#centr_video.check()

#track_name = '2017242N16333' # '2016273N13300' # '1992230N11325'
#tc_video = TropCyclone()
# use file_name="" to not to write the video
#tc_list, tr_coord = tc_video.video_intensity(track_name, tr_irma, centr_video, file_
↪name='./results/irma_tc_fl.gif')
# tc_list contains a list with TropCyclone instances plotted at each time step
# tr_coord contains a list with the track path coordinates plotted at each time step

# mp4 occupies much less space! To use it:
# conda install ffmpeg
# in code:
# plt.rcParams['animation.ffmpeg_path']='path/to/climada_env/bin/ffmpeg'
# writer=animation.FFMpegWriter(bitrate=500)
# tc_list, tr_coord = tc_video.video_intensity(track_name, tr_irma, centr_video, file_
↪name='./results/irma_tc_fl.gif', writer=writer)
```

REFERENCES:

- Bloemendaal, N., Haigh, I. D., de Moel, H., Muis, S., Haarsma, R. J., & Aerts, J. C. J. H. (2020). Generation of a global synthetic tropical cyclone hazard dataset using STORM. *Scientific Data*, 7(1). <https://doi.org/10.1038/s41597-020-0381-2>
- Emanuel, K., S. Ravela, E. Vivant, and C. Risi, 2006: A Statistical Deterministic Approach to Hurricane Risk Assessment. *Bull. Amer. Meteor. Soc.*, 87, 299–314, <https://doi.org/10.1175/BAMS-87-3-299>.
- Geiger, T., Frieler, K., & Levermann, A. (2016). High-income does not protect against hurricane losses. *Environmental Research Letters*, 11(8). <https://doi.org/10.1088/1748-9326/11/8/084012>
- Knutson, T. R., Sirutis, J. J., Zhao, M., Tuleya, R. E., Bender, M., Vecchi, G. A., ... Chavas, D. (2015). Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios. *Journal of Climate*, 28(18), 7203–7224. <https://doi.org/10.1175/JCLI-D-15-0129.1>
- Lee, C. Y., Tippet, M. K., Sobel, A. H., & Camargo, S. J. (2018). An environmentally forced tropical cyclone hazard model. *Journal of Advances in Modeling Earth Systems*, 10(1), 223–241. <https://doi.org/10.1002/2017MS001186>

```
[11]: from climada.entity.exposures import LitPop

exp_litpop = LitPop()
exp_litpop.set_country('Puerto Rico', res_arcsec = 120) # We'll go lower resolution than_
↪default to keep it simple
exp_litpop.set_geometry_points() # Set geodataframe geometries from lat lon data
```

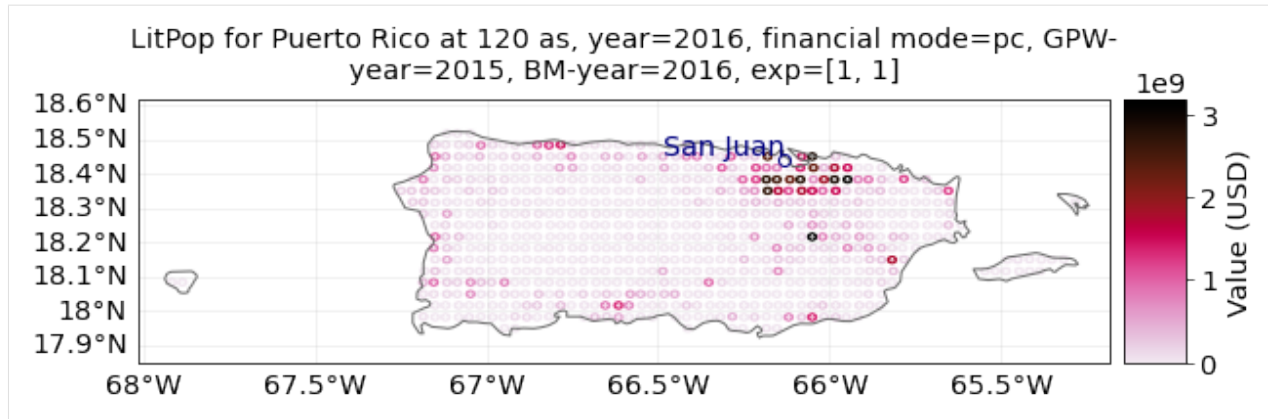
(continues on next page)

(continued from previous page)

```
exp_litpop.plot_hexbin(pop_name=True, linewidth=4, buffer=0.1)
```

```
2021-06-04 17:11:04,971 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-06-04 17:11:04,972 - climada.entity.exposures.base - INFO - tag set to default.
↳value File:
Description:
2021-06-04 17:11:04,976 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-06-04 17:11:04,978 - climada.entity.exposures.base - INFO - value_unit set to
↳default value USD
2021-06-04 17:11:04,982 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-06-04 17:11:06,788 - climada.entity.exposures.litpop - INFO - Generating LitPop.
↳data at a resolution of 120 arcsec.
2021-06-04 17:11:10,508 - climada.entity.exposures.gpw_import - INFO - Reference year:
↳2016. Using nearest available year for GPW population data: 2015
2021-06-04 17:11:10,509 - climada.entity.exposures.gpw_import - INFO - GPW Version v4.11
2021-06-04 17:11:37,893 - climada.util.finance - WARNING - No data available for country.
↳ Using non-financial wealth instead
2021-06-04 17:11:38,462 - climada.util.finance - INFO - GDP PRI 2016: 1.043e+11.
2021-06-04 17:11:38,467 - climada.util.finance - WARNING - No data for country, using
↳mean factor.
2021-06-04 17:11:38,622 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-06-04 17:11:38,622 - climada.entity.exposures.base - INFO - tag set to default.
↳value File:
Description:
2021-06-04 17:11:38,623 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-06-04 17:11:38,623 - climada.entity.exposures.base - INFO - value_unit set to
↳default value USD
2021-06-04 17:11:38,626 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-06-04 17:11:38,649 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-06-04 17:11:38,653 - climada.entity.exposures.base - INFO - Hazard type not set in
↳impf_
2021-06-04 17:11:38,654 - climada.entity.exposures.base - INFO - category_id not set.
2021-06-04 17:11:38,655 - climada.entity.exposures.base - INFO - cover not set.
2021-06-04 17:11:38,656 - climada.entity.exposures.base - INFO - deductible not set.
2021-06-04 17:11:38,657 - climada.entity.exposures.base - INFO - geometry not set.
2021-06-04 17:11:38,658 - climada.entity.exposures.base - INFO - centr_ not set.
2021-06-04 17:11:38,675 - climada.util.coordinates - INFO - Setting geometry points.
```

```
[11]: <GeoAxesSubplot:title={'center': 'LitPop for Puerto Rico at 120 as, year=2016, financial.
↳mode=pc, GPW-\nyear=2015, BM-year=2016, exp=[1, 1]'}>
```



5.10 Hazard: winter windstorms / extratropical cyclones in Europe

5.10.1 Or: The StormEurope hazard subclass of CLIMADA

Auth: Jan Hartman & Thomas Rösli

Date: 2018-04-26 & 2020-03-03

This notebook will give a quick tour of the capabilities of the StormEurope hazard class. This includes functionalities to apply probabilistic alterations to historical storms.

```
[2]: %matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [15, 10]
```

5.10.2 Reading Data

StormEurope was written under the presumption that you'd start out with WISC storm footprint data in netCDF format. This notebook works with a demo dataset. If you would like to work with the real data: (1) Please follow the link and download the file C3S_WISC_FOOTPRINT_NETCDF_0100.tgz from the Copernicus Windstorm Information Service, (2) unzip it (3) uncomment the last two lines in the following codeblock and (4) adjust the variable "WISC_files".

We first construct an instance and then point the reader at a directory containing compatible .nc files. Since there are other files in there, we must be explicit and use a globbing pattern; supplying incompatible files will make the reader fail.

The reader actually calls `climada.util.files_handler.get_file_names`, so it's also possible to hand it an explicit list of filenames, or a dirname, or even a list of glob patterns or directories.

```
[3]: from climada.hazard import StormEurope
from climada.util.constants import WS_DEMO_NC

storm_instance = StormEurope()
storm_instance.read_footprints(WS_DEMO_NC)

# WISC_files = '/path/to/folder/C3S_WISC_FOOTPRINT_NETCDF_0100/fp_era[!er5]*_0.nc'
# storm_instance.read_footprints(WISC_files)
```

```

2020-03-05 10:29:24,582 - climada - DEBUG - Loading default config file: C:\shortpaths\
↳GitHub\climada_python\climada\conf\defaults.conf
2020-03-05 10:29:25,946 - climada - DEBUG - Loading user config file: C:\shortpaths\
↳GitHub\climada.conf
2020-03-05 10:29:29,788 - climada.hazard.storm_europe - INFO - Constructing centroids_
↳from C:\shortpaths\GitHub\climada_python\data\demo\fp_lothar_crop-test.nc
2020-03-05 10:29:29,821 - climada.hazard.centroids.centri - INFO - Setting geometry_
↳points.
2020-03-05 10:29:29,898 - climada.hazard.centroids.centri - DEBUG - Setting area_pixel_
↳9944 points.
2020-03-05 10:29:31,113 - climada.hazard.centroids.centri - DEBUG - Setting on_land 9944_
↳points.
2020-03-05 10:29:31,624 - climada.hazard.storm_europe - INFO - Commencing to iterate_
↳over netCDF files.
2020-03-05 10:29:31,723 - climada.util.checker - DEBUG - Hazard.ssi not set.
2020-03-05 10:29:31,724 - climada.util.checker - DEBUG - Hazard.ssi_full_area not set.
2020-03-05 10:29:31,777 - climada.util.checker - DEBUG - Hazard.ssi not set.
2020-03-05 10:29:31,778 - climada.util.checker - DEBUG - Hazard.ssi_full_area not set.
2020-03-05 10:29:31,786 - climada.hazard.base - DEBUG - Resetting event_id.

```

5.10.3 Introspection

Let's quickly see what attributes this class brings with it:

```
[4]: storm_instance?
```

You could also try listing all permissible methods with `dir(storm_instance)`, but since that would include the methods from the Hazard base class, you wouldn't know what's special. The best way is to read the source: uncomment the following statement to read more.

```
[5]: # StormEurope??
```

5.10.4 Into the Storm Severity Index (SSI)

The SSI, according to [Dawkins et al. 2016](#) or [Lamb and Frydendahl, 1991](#), can be set using `set_ssi`. For demonstration purposes, I show the default arguments. (Check also the defaults using `storm_instance.calc_ssi?`, the method for which `set_ssi` is a wrapper.)

We won't be using the `plot_ssi` functionality just yet, because we only have two events; the graph really isn't informative. After this, we'll generate some more storms to make that plot more aesthetically pleasing.

```

[6]: storm_instance.set_ssi(
    method = 'wind_gust',
    intensity = storm_instance.intensity,
    # the above is just a more explicit way of passing the default
    on_land = True,
    threshold = 25,
    sel_cen = None
    # None is default. sel_cen could be used to subset centroids
)

```

5.10.5 Probabilistic Storms

This class allows generating probabilistic storms from historical ones according to a method outlined in [Schwierz et al. 2010](#). This means that per historical event, we generate 29 new ones with altered intensities. Since it's just a bunch of vector operations, this is pretty fast.

However, we should not return the entire probabilistic dataset in-memory: in trials, this used up 60 GB of RAM, thus requiring a great amount of swap space. Instead, we must select a country by setting the `reg_id` parameter to an ISO_N3 country code used in the [Natural Earth](#) dataset. It is also possible to supply a list of ISO codes. If your machine is up for the job of handling the whole dataset, set the `reg_id` parameter to `None`.

Since assigning each centroid a country ID is a rather inefficient affair, you may need to wait a minute or two for the entire WISC dataset to be processed. For the small demo dataset, it runs pretty quickly.

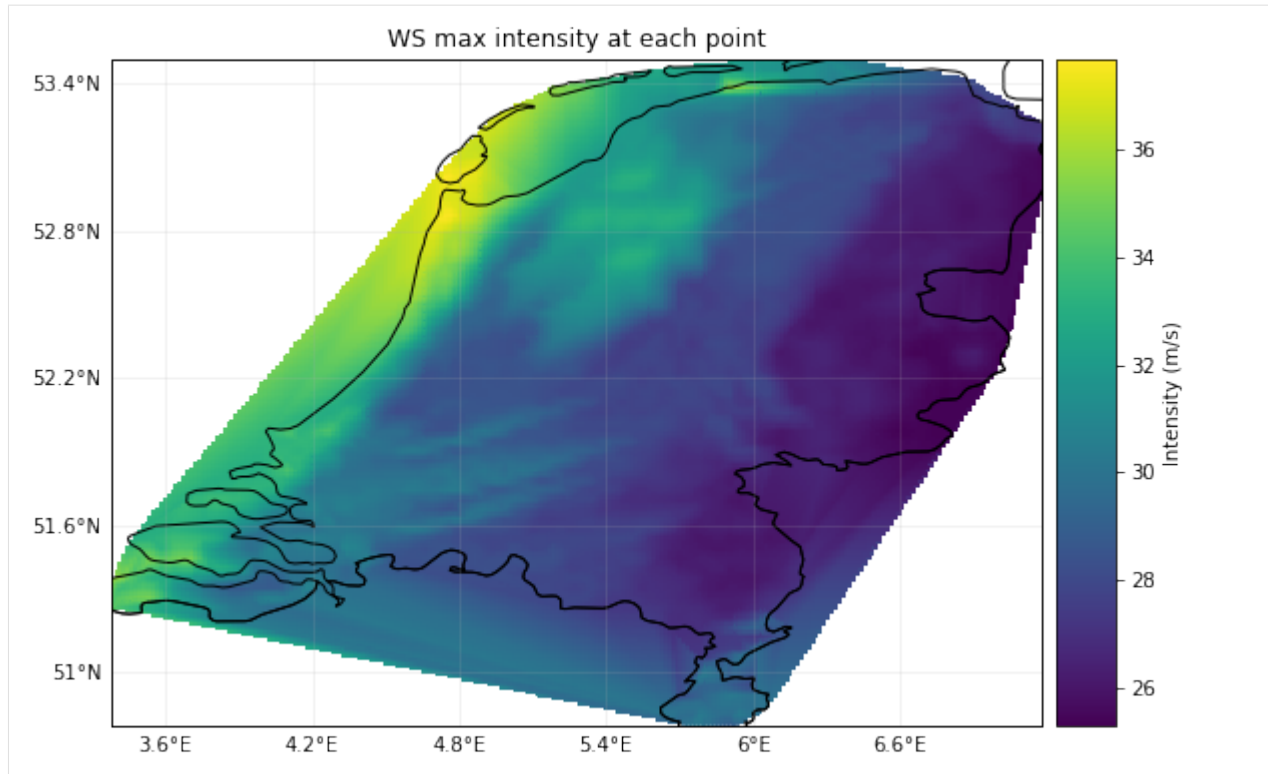
```
[7]: %%time
storm_prob = storm_instance.generate_prob_storms(reg_id=528)
storm_prob.plot_intensity(0)

2020-03-05 10:29:31,845 - climada.hazard.centroids.cent - INFO - Setting geometry_
↳points.
2020-03-05 10:29:32,248 - climada.hazard.centroids.cent - DEBUG - Setting region_id_
↳9944 points.
2020-03-05 10:29:32,466 - climada.util.coordinates - DEBUG - Setting region_id 9944_
↳points.
2020-03-05 10:29:33,506 - climada.hazard.storm_europe - INFO - Commencing probabilistic_
↳calculations
2020-03-05 10:29:33,620 - climada.hazard.storm_europe - INFO - Generating new_
↳StormEurope instance
2020-03-05 10:29:33,663 - climada.util.checker - DEBUG - Hazard.ssi not set.
2020-03-05 10:29:33,664 - climada.util.checker - DEBUG - Hazard.ssi_wisc not set.
2020-03-05 10:29:33,665 - climada.util.checker - DEBUG - Hazard.event_name not set._
↳Default values set.

C:\shortpaths\GitHub\climada_python\climada\util\plot.py:311: UserWarning: Tight layout_
↳not applied. The left and right margins cannot be made large enough to accommodate all_
↳axes decorations.
fig.tight_layout()

Wall time: 2.24 s

[7]: <cartopy.mpl.geoaxes.GeoAxesSubplot at 0x1dafba69940>
```



We can get much more fancy in our calls to `generate_prob_storms`; the keyword arguments after `ssi_args` are passed on to `_hist2prob`, allowing us to tweak the probabilistic permutations.

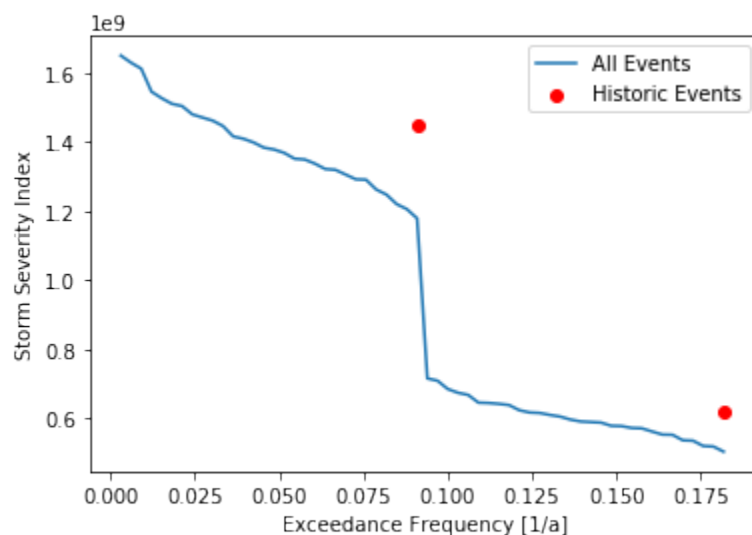
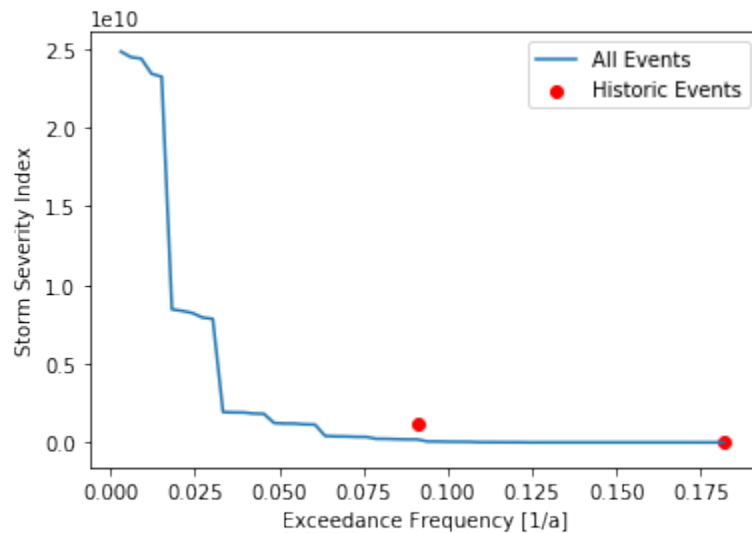
```
[9]: ssi_args = {
    'on_land': True,
    'threshold': 25,
}

storm_prob_extreme = storm_instance.generate_prob_storms(
    reg_id=[56, 528], # BEL and NLD
    spatial_shift=2,
    ssi_args=ssi_args,
    power=1.5,
    scale=0.3
)

2020-03-05 10:29:36,628 - climada.hazard.storm_europe - INFO - Commencing probabilistic_
↳ calculations
2020-03-05 10:29:36,738 - climada.hazard.storm_europe - INFO - Generating new_
↳ StormEurope instance
2020-03-05 10:29:36,807 - climada.util.checker - DEBUG - Hazard.ssi not set.
2020-03-05 10:29:36,808 - climada.util.checker - DEBUG - Hazard.ssi_wisc not set.
2020-03-05 10:29:36,808 - climada.util.checker - DEBUG - Hazard.event_name not set._
↳ Default values set.
```

We can now check out the SSI plots of both these calculations. The comparison between the historic and probabilistic ssi values, only makes sense for the full dataset.

```
[10]: storm_prob_xtreme.plot_ssi(full_area=True)
storm_prob.plot_ssi(full_area=True)
```



```
[10]: (<Figure size 432x288 with 1 Axes>,
<matplotlib.axes._subplots.AxesSubplot at 0x1dafba99128>)
```

5.11 END-TO-END IMPACT CALCULATION

5.11.1 Goal of this tutorial

The goal of this tutorial is to show a full end-to-end impact computation. Note that this tutorial exemplifies the work flow, but does not explore all possible features.

5.11.2 What is an Impact?

The impact is the combined effect of hazard events on a set of exposures mediated by a set of impact functions. By computing the impact for each event (historical and synthetic) and for each exposure value at each geographical location, the Impact class provides different risk measures, such as the expected annual impact per exposure, the probable maximum impact for different return periods, and the total average annual impact.

5.11.3 Impact class data structure

The impact class does not require any attributes to be defined by the user. All attributes are set by the method `impact.calc()`. This method requires three attributes: an `Exposure`, a `Hazard`, and an `ImpactFuncSet`. After calling `impact.calc(Exposure, ImpactFuncSet, Hazard, save_mat=False)`, the Impact object has the following attributes:

Attributes from input	Data Type	Description
tag	(dict)	dictionary storing the tags of the inputs (Exposure.tag, ImpactFuncSet.tag Hazard.tag)
even_id	list(int)	id (>0) of each hazard event (Hazard.event_id)
event_name	(list(str))	name of each event (Hazard.event_name)
date	np.array	date of events (Hazard.date)
coord_exp	np.array	exposures coordinates [lat, lon] (in degrees) (Exposure.gdf.latitudes, Exposure.gdf.longitudes)
frequency	np.array	annual frequency of events (Hazard.frequency)
unit	str	value unit used (Exposure.value_unit)
csr	str	unit system for Exposure and Hazard geographical data (Exposure.csr)

Computed attributes	Data Type	Description
at_event	np.array	impact for each hazard event summed over all locations
eai_exp	np.array	expected annual impact for each locations, summed over all events weighted by frequency
aai_agg	float	total annual average aggregated impact value (summed over events and locations)
impt_mat	sparse.csr_matrix	matrix (num_events X num_exp) with impact values (only filled if save_mat is True).
tot_value	float	total exposure value affected (sum of value all exposures locations affected by at least one hazard event)

All other methods compute values from the attributes set by `Impact.calc()`. For example, one can compute the frequency exceedance curve, plot impact data, or compute traditional risk transfer over impact.

5.11.4 How do I compute an impact in CLIMADA?

In CLIMADA, impacts are computed using the `Impact` class. To computation of the impact requires an `Exposure`, an `ImpactFuncSet`, and a `Hazard` object. For details about how to define `Exposures` <climada_entity_Exposures.ipynb>`__`, `Hazard` <climada_hazard_Hazard.ipynb>`__`, `Impact Functions` <climada_entity_ImpactFuncSet.ipynb>`__` see the respective tutorials.

The steps of an impact calculations are typically:

- Set exposure
- Set hazard and hazard centroids
- Set impact functions in impact function set
- Compute impact
- Visualize, save, use impact output

Hints: Before computing the impact of a given `Exposure` and `Hazard`, it is important to correctly match the `Exposures`' coordinates with the `Hazard` Centroids. Try to have similar resolutions in `Exposures` and `Hazard`. By the impact calculation the nearest neighbor for each `Exposure` to the `Hazard`'s Centroids is searched.

Hint: Set first the `Exposures` and use its coordinates information to set a matching `Hazard`.

Hint: The configurable parameter `max_matrix_size` defined in the `configuration file` <../climada/conf/defaults.conf>`__` (located at `/climada/conf/defaults.conf`) controls the maximum matrix size contained in a chunk. You can decrease its value if you are having memory issues when using the `Impact.calc()` method. A high value will make the computation fast, but increase the memory use.

5.11.5 Structure of the tutorial

We begin with one very detailed example, and later present in quick and dirty examples.

`Part1` <#part1>`__`: Detailed impact calculation with `Litpop` and `TropCyclone`

`Part2` <#part2>`__`: Quick examples: raster and point exposures/hazards

`Part3` <#part3>`__`: Visualization methods

Detailed Impact calculation - LitPop + TropCyclone

We present a detailed example for the hazard `Tropical Cyclones` <climada_hazard_TropCyclone.ipynb>`__` and the exposures from `LitPop` <climada_entity_LitPop.ipynb>`__`.

Define the exposure

Reminder: The exposures must be defined according to your problem either using CLIMADA exposures such as `BlackMarble` <climada_entity_BlackMarble.ipynb>`__`, `LitPop` <climada_entity_LitPop.ipynb>`__`, `OSM` <climada_entity_openstreetmap.ipynb>`__`, extracted from external sources (imported via csv, excel, api, ...) or directly user defined. As a reminder, exposures are `geopandas` dataframes with at least columns 'latitude', 'longitude' and 'value' of exposures. For impact calculations, for each exposure value the corresponding impact function to use (defined by the column `impf_`) and the associated hazard centroids must be defined. This is done after defining the impact function(s) and the hazard(s). See tutorials on `Exposures` <climada_entity_Exposures.ipynb>`__`, `Hazard` <climada_hazard_Hazard.ipynb>`__`, `ImpactFuncSet` <climada_entity_ImpactFuncSet.ipynb>`__` for more details.

Exposures are either defined as a series of (latitude/longitude) points or as a raster of (latitude/longitude) points. Fundamentally, this changes nothing for the impact computations. Note that for larger number of points, consider using a raster which might be more efficient (computationally). For a low number of points, avoid using a raster if this adds a lot of exposures values equal to 0.

We shall here use a raster example.

```
[1]: # Exposure from the module Litpop
# Note that the file gpw_v4_population_count_rev11_2015_30_sec.tif must be downloaded.
# (do not forget to unzip) if
# you want to execute this cell on your computer.

%matplotlib inline
import numpy as np
from climada.entity import LitPop

# Cuba with resolution 10km and financial_mode = income group.
exp_lp = LitPop()
exp_lp.set_country(countries=['CUB'], res_arcsec=300, fin_mode='income_group')
exp_lp.check()

2021-04-30 13:10:51,006 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-30 13:10:51,010 - climada.entity.exposures.base - INFO - tag set to default.
↳value File:
Description:
2021-04-30 13:10:51,012 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-04-30 13:10:51,014 - climada.entity.exposures.base - INFO - value_unit set to.
↳default value USD
2021-04-30 13:10:51,016 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-04-30 13:10:52,560 - climada.entity.exposures.litpop - INFO - Generating LitPop.
↳data at a resolution of 300.0 arcsec.
2021-04-30 13:10:55,947 - climada.entity.exposures.gpw_import - INFO - Reference year:.
↳2016. Using nearest available year for GPW population data: 2015
2021-04-30 13:10:55,949 - climada.entity.exposures.gpw_import - INFO - GPW Version v4.11
2021-04-30 13:11:12,426 - climada.util.finance - INFO - GDP CUB 2016: 9.137e+10.
2021-04-30 13:11:12,577 - climada.util.finance - INFO - Income group CUB 2016: 3.
2021-04-30 13:11:12,836 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-30 13:11:12,838 - climada.entity.exposures.base - INFO - tag set to default.
↳value File:
Description:
2021-04-30 13:11:12,839 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-04-30 13:11:12,840 - climada.entity.exposures.base - INFO - value_unit set to.
↳default value USD
2021-04-30 13:11:12,844 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-04-30 13:11:12,874 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-30 13:11:12,880 - climada.entity.exposures.base - INFO - Hazard type not set in.
↳impf_
```

(continues on next page)

(continued from previous page)

```

2021-04-30 13:11:12,882 - climada.entity.exposures.base - INFO - category_id not set.
2021-04-30 13:11:12,884 - climada.entity.exposures.base - INFO - cover not set.
2021-04-30 13:11:12,889 - climada.entity.exposures.base - INFO - deductible not set.
2021-04-30 13:11:12,891 - climada.entity.exposures.base - INFO - geometry not set.
2021-04-30 13:11:12,892 - climada.entity.exposures.base - INFO - centr_ not set.
2021-04-30 13:11:12,914 - climada.entity.exposures.base - INFO - Hazard type not set in_
↳ impf_
2021-04-30 13:11:12,916 - climada.entity.exposures.base - INFO - category_id not set.
2021-04-30 13:11:12,918 - climada.entity.exposures.base - INFO - cover not set.
2021-04-30 13:11:12,919 - climada.entity.exposures.base - INFO - deductible not set.
2021-04-30 13:11:12,923 - climada.entity.exposures.base - INFO - geometry not set.
2021-04-30 13:11:12,924 - climada.entity.exposures.base - INFO - centr_ not set.

```

```
[2]: exp_lp.gdf.head()
```

```

[2]:      value  latitude  longitude  region_id  impf_
0  9.818338e+05  21.875000 -84.875000        192      1
1  1.110009e+06  21.875000 -84.791667        192      1
2  0.000000e+00  21.958333 -84.708333        192      1
3  1.026952e+06  21.958333 -84.625000        192      1
4  1.109359e+06  21.958333 -84.541667        192      1

```

```

[3]: # not needed for impact calculations
      # visualize the define exposure
      exp_lp.plot_raster()
      print('\n Raster properties exposures:', exp_lp.meta)

```

```

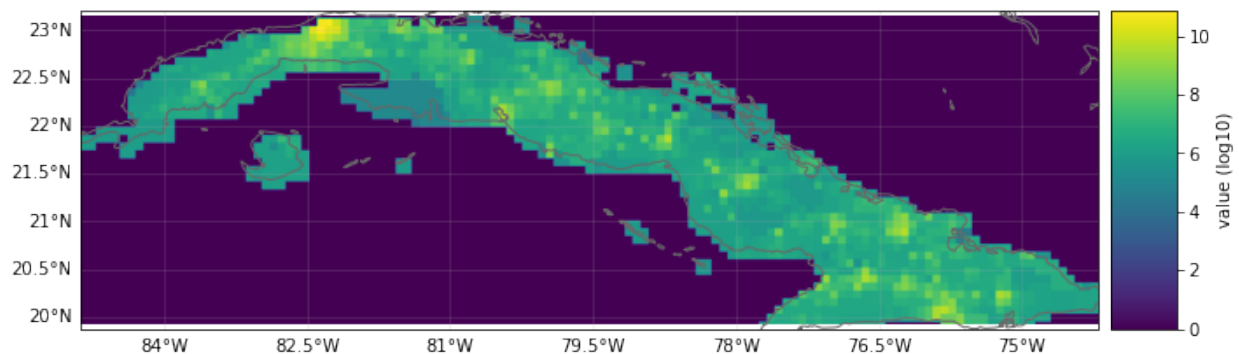
2021-04-30 13:11:13,034 - climada.util.coordinates - INFO - Raster from resolution 0.
↳ 0.08333333333333286 to 0.08333333333333286.

```

```

Raster properties exposures: {'width': 129, 'height': 41, 'crs': 'EPSG:4326', 'transform
↳ ': Affine(0.08333333333333286, 0.0, -84.91666666666669,
0.0, 0.08333333333333286, 19.833333333333336)}

```



Define the hazard

Let us define a tropical cyclone hazard using the TropCyclone and TCTracks modules.

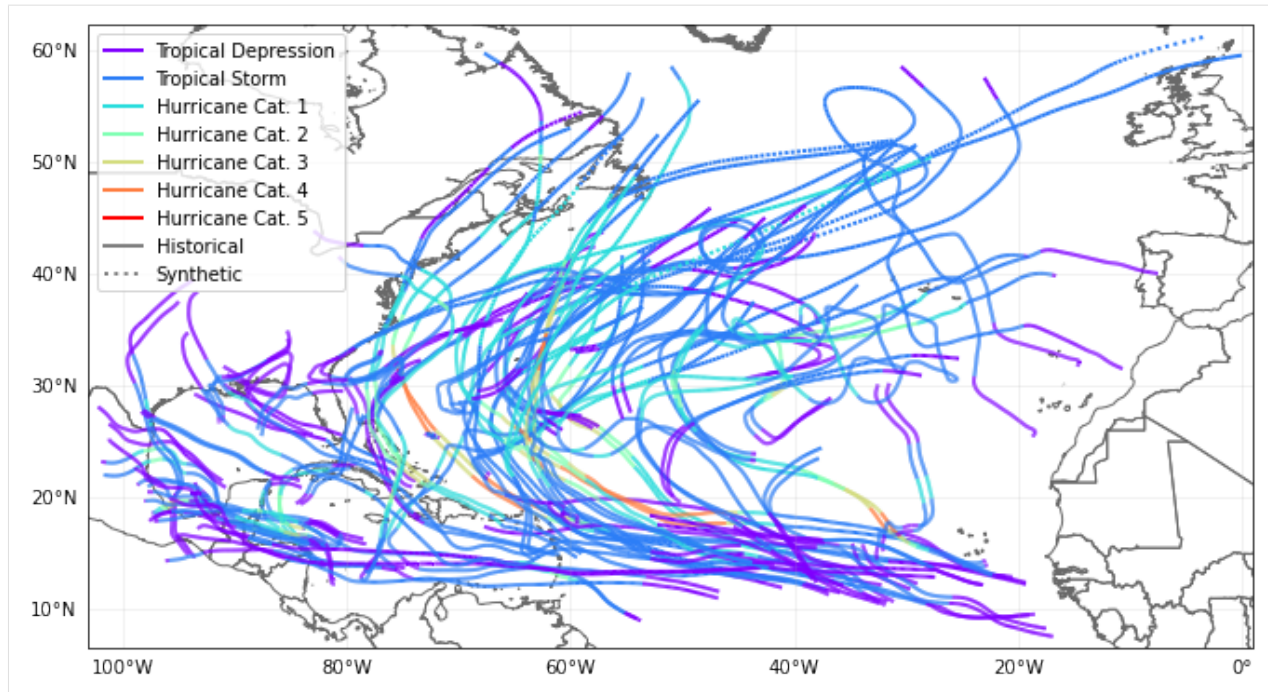
```
[4]: from climada.hazard import TCTracks, TropCyclone, Centroids

# Load historical tropical cyclone tracks from ibtracks over the North Atlantic basin
# between 2010-2012
ibtracks_na = TCTracks()
ibtracks_na.read_ibtracks_netcdf(provider='usa', basin='NA', year_range=(2010, 2012),
# correct_pres=True)
print('num tracks hist:', ibtracks_na.size)

ibtracks_na.equal_timestep(0.5) # Interpolation to make the track smooth and to allow
# applying calc_perturbed_trajectories
# Add randomly generated tracks using the calc_perturbed_trajectories method (1 per
# historical track)
ibtracks_na.calc_perturbed_trajectories(nb_synth_tracks=1)
print('num tracks hist+syn:', ibtracks_na.size)

2021-04-30 13:11:20,100 - climada.hazard.tc_tracks - WARNING - `correct_pres` is
# deprecated. Use `estimate_missing` instead.
2021-04-30 13:11:21,255 - climada.hazard.tc_tracks - INFO - Progress: 10%
2021-04-30 13:11:21,380 - climada.hazard.tc_tracks - INFO - Progress: 20%
2021-04-30 13:11:21,516 - climada.hazard.tc_tracks - INFO - Progress: 30%
2021-04-30 13:11:21,646 - climada.hazard.tc_tracks - INFO - Progress: 40%
2021-04-30 13:11:21,780 - climada.hazard.tc_tracks - INFO - Progress: 50%
2021-04-30 13:11:21,922 - climada.hazard.tc_tracks - INFO - Progress: 60%
2021-04-30 13:11:22,063 - climada.hazard.tc_tracks - INFO - Progress: 70%
2021-04-30 13:11:22,207 - climada.hazard.tc_tracks - INFO - Progress: 80%
2021-04-30 13:11:22,333 - climada.hazard.tc_tracks - INFO - Progress: 90%
2021-04-30 13:11:22,468 - climada.hazard.tc_tracks - INFO - Progress: 100%
num tracks hist: 60
2021-04-30 13:11:22,490 - climada.hazard.tc_tracks - INFO - Interpolating 60 tracks to 0.
# 5h time steps.
2021-04-30 13:11:25,842 - climada.hazard.tc_tracks_synth - INFO - Computing 60 synthetic
# tracks.
num tracks hist+syn: 120
```

```
[5]: # not needed for calculations
# visualize tracks
ax = ibtracks_na.plot()
ax.get_legend()._loc = 2
```



From the tracks, we generate the hazards (the tracks are only the coordinates of the center of the cyclones, the full cyclones however affects a region around the tracks).

First thing we define the set of centroids which are geographical points where the hazard has a defined value. In our case, we want to define windspeeds from the tracks.

Remember: In the impact computations, for each exposure geographical point, one must assign a centroid from the hazard. By default, each exposure is assigned to the closest centroid from the hazard. But one can also define manually which centroid is assigned to which exposure point.

Examples: - Define the exposures from a given source (e.g., raster of asset values from LitPop). Define the hazard centroids from the exposures' geolocations (e.g. compute Tropical Cyclone windspeed at each raster point and assign centroid to each raster point). - Define the exposures from a given source (e.g. houses position and value). Define the hazard from a given source (e.g. where landslides occur). Use a metric to assign to each exposures point a hazard centroid (all houses in a radius of 5km around the landslide are assigned to this centroid, if a house is within 5km of two landslides, choose the closest one). - Define a geographical raster. Define the exposures value on this raster. Define the hazard centroids on the geographical raster.

We shall pursue with the first case (Litpop + TropicalCyclone)

Hint: computing the wind speeds in many locations for many tc tracks is a computationally costly operation. Thus, we should define centroids only where we also have an exposure.

[6]: *# Define the centroids from the exposures position*

```
centrs = Centroids()
lat = exp_lp.gdf['latitude'].values
lon = exp_lp.gdf['longitude'].values
centrs.set_lat_lon(lat, lon)
centrs.check()
```

[7]: *# Using the tracks, compute the windspeed at the location of the centroids*

```
tc = TropCyclone.from_tracks(ibtracks_na, centroids=centrs)
tc.check()
```

```

2021-04-30 13:12:04,245 - climada.hazard.centroids.centroid - INFO - Convert centroids to
↳ GeoSeries of Point shapes.
2021-04-30 13:12:05,392 - climada.util.coordinates - INFO - dist_to_coast: UTM 32616 (1/
↳ 3)
2021-04-30 13:12:05,800 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617 (2/
↳ 3)
2021-04-30 13:12:07,010 - climada.util.coordinates - INFO - dist_to_coast: UTM 32618 (3/
↳ 3)
2021-04-30 13:12:07,695 - climada.hazard.trop_cyclone - INFO - Mapping 120 tracks to
↳ 1387 coastal centroids.
2021-04-30 13:12:08,106 - climada.hazard.trop_cyclone - INFO - Progress: 10%
2021-04-30 13:12:08,306 - climada.hazard.trop_cyclone - INFO - Progress: 20%
2021-04-30 13:12:08,798 - climada.hazard.trop_cyclone - INFO - Progress: 30%
2021-04-30 13:12:09,322 - climada.hazard.trop_cyclone - INFO - Progress: 40%
2021-04-30 13:12:09,873 - climada.hazard.trop_cyclone - INFO - Progress: 50%
2021-04-30 13:12:10,004 - climada.hazard.trop_cyclone - INFO - Progress: 60%
2021-04-30 13:12:10,322 - climada.hazard.trop_cyclone - INFO - Progress: 70%
2021-04-30 13:12:10,519 - climada.hazard.trop_cyclone - INFO - Progress: 80%
2021-04-30 13:12:10,973 - climada.hazard.trop_cyclone - INFO - Progress: 90%
2021-04-30 13:12:11,377 - climada.hazard.trop_cyclone - INFO - Progress: 100%

```

Hint: The operation of computing the windspeed in different location is in general computationally expensive. Hence, if you have a lot of tropical cyclone tracks, you should first make sure that all your tropical cyclones actually affect your exposure (remove those that don't). Then, be careful when defining the centroids. For a large country like China, there is no need for centroids 500km inland (no tropical cyclones get so far).

Impact function

For Tropical Cyclones, some calibrated default impact functions exist. Here we will use the one from Emanuel (2011).

```

[8]: from climada.entity import ImpactFuncSet, IFTropCyclone
     # impact function TC
     impf_tc= IFTropCyclone()
     impf_tc.set_emanuel_usa()

     # add the impact function to an Impact function set
     impf_set = ImpactFuncSet()
     impf_set.append(impf_tc)
     impf_set.check()

2021-04-30 13:12:11,789 - 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.

```

Recall that the exposures, hazards and impact functions must be matched in the impact calculations. Here it is simple, since there is a single impact function for all the hazards. We must simply make sure that the exposure is assigned this impact function through renaming the `impf_` column from the hazard type of the impact function in the impact function set and set the values of the column to the id of the impact function.

```

[9]: # Get the hazard type and hazard id
     [haz_type] = impf_set.get_hazard_types()
     [haz_id] = impf_set.get_ids()[haz_type]
     print(f'hazard type: {haz_type}, hazard id: {haz_id}')

```



```
hazard type: TC, hazard id: 1
```

```
[10]: # Exposures: rename column and assign id
exp_lp.gdf.rename(columns={"impf_": "impf_" + haz_type}, inplace=True)
exp_lp.gdf['impf_' + haz_type] = haz_id
exp_lp.check()
exp_lp.gdf.head()

2021-04-30 13:12:11,822 - climada.entity.exposures.base - INFO - category_id not set.
2021-04-30 13:12:11,824 - climada.entity.exposures.base - INFO - cover not set.
2021-04-30 13:12:11,826 - climada.entity.exposures.base - INFO - deductible not set.
2021-04-30 13:12:11,828 - climada.entity.exposures.base - INFO - geometry not set.
2021-04-30 13:12:11,830 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
[10]:
```

	value	latitude	longitude	region_id	impf_TC
0	9.818338e+05	21.875000	-84.875000	192	1
1	1.110009e+06	21.875000	-84.791667	192	1
2	0.000000e+00	21.958333	-84.708333	192	1
3	1.026952e+06	21.958333	-84.625000	192	1
4	1.109359e+06	21.958333	-84.541667	192	1

Impact computation

We are finally ready for the impact computation. This is the simplest step. Just give the exposure, impact function and hazard to the `Impact.calc()` method.

Note: we did not specifically assign centroids to the exposures. Hence, the default is used - each exposure is associated with the closest centroids. Since we defined the centroids from the exposures, this is a one-to-one mapping.

Note: we did not define an Entity in this impact calculations. Recall that Entity is a container class for ``Exposures <climada_entity_Exposures.ipynb>``, ``Impact Functions <climada_entity_ImpactFuncSet.ipynb>``, ``Discount Rates <climada_entity_DiscRates.ipynb>`` and ``Measures <climada_entity_MeasureSet.ipynb>``. Since we had only one Exposure and one Impact Function, the container would not have added any value, but for more complex projects, the Entity class is very useful.

```
[11]: # Compute impact
from climada.engine import Impact
imp = Impact()
imp.calc(exp_lp, impf_set, tc, save_mat=False) #Do not save the results geographically,
↳resolved (only aggregate values)

2021-04-30 13:12:11,894 - climada.entity.exposures.base - INFO - Matching 1387 exposures,
↳with 1387 centroids.
2021-04-30 13:12:11,899 - climada.engine.impact - INFO - Calculating damage for 1373,
↳assets (>0) and 120 events.
```

```
[12]: exp_lp.gdf
```

	value	latitude	longitude	region_id	impf_TC	centr_TC
0	9.818338e+05	21.875000	-84.875000	192	1	0
1	1.110009e+06	21.875000	-84.791667	192	1	1
2	0.000000e+00	21.958333	-84.708333	192	1	2
3	1.026952e+06	21.958333	-84.625000	192	1	3
4	1.109359e+06	21.958333	-84.541667	192	1	4

(continues on next page)

(continued from previous page)

```

...      ...      ...      ...      ...      ...
1382  4.015236e+05  20.291667 -74.291667      192      1      1382
1383  2.292229e+06  20.208333 -74.291667      192      1      1383
1384  2.293454e+06  20.125000 -74.291667      192      1      1384
1385  2.290998e+06  20.291667 -74.208333      192      1      1385
1386  2.292228e+06  20.208333 -74.208333      192      1      1386

[1387 rows x 6 columns]

```

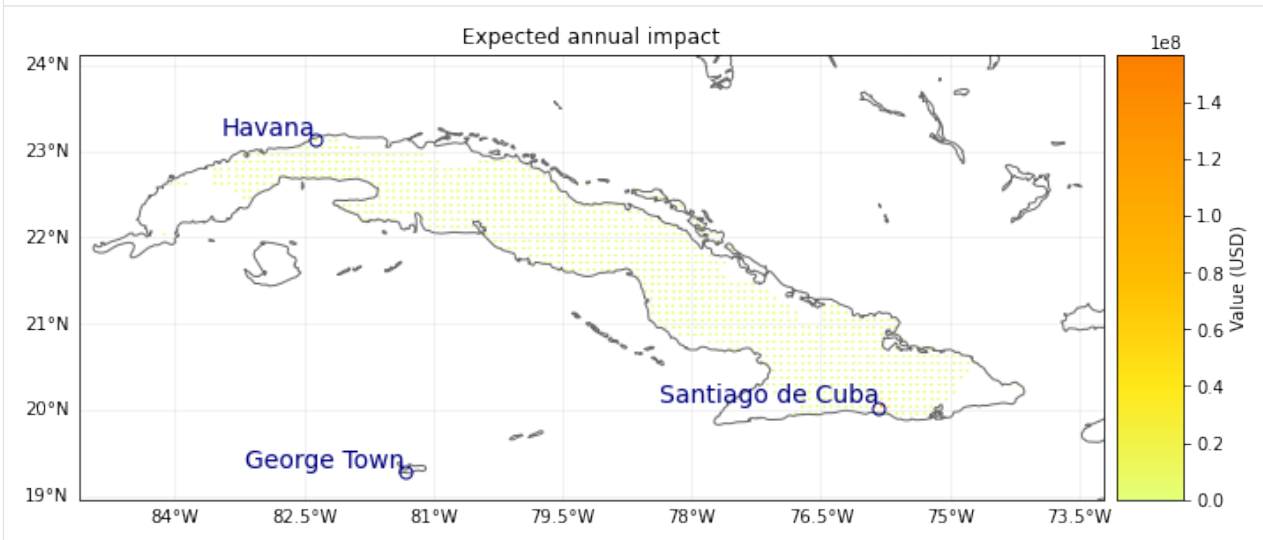
For example we can now obtain the aggregated average annual impact or plot the average annual impact in each exposure location.

```
[13]: print(f"Aggregated average annual impact: {round(imp.aai_agg,0)} $")
```

```
Aggregated average annual impact: 424196666.0 $
```

```
[14]: imp.plot_hexbin_eai_exposure(buffer=1)
```

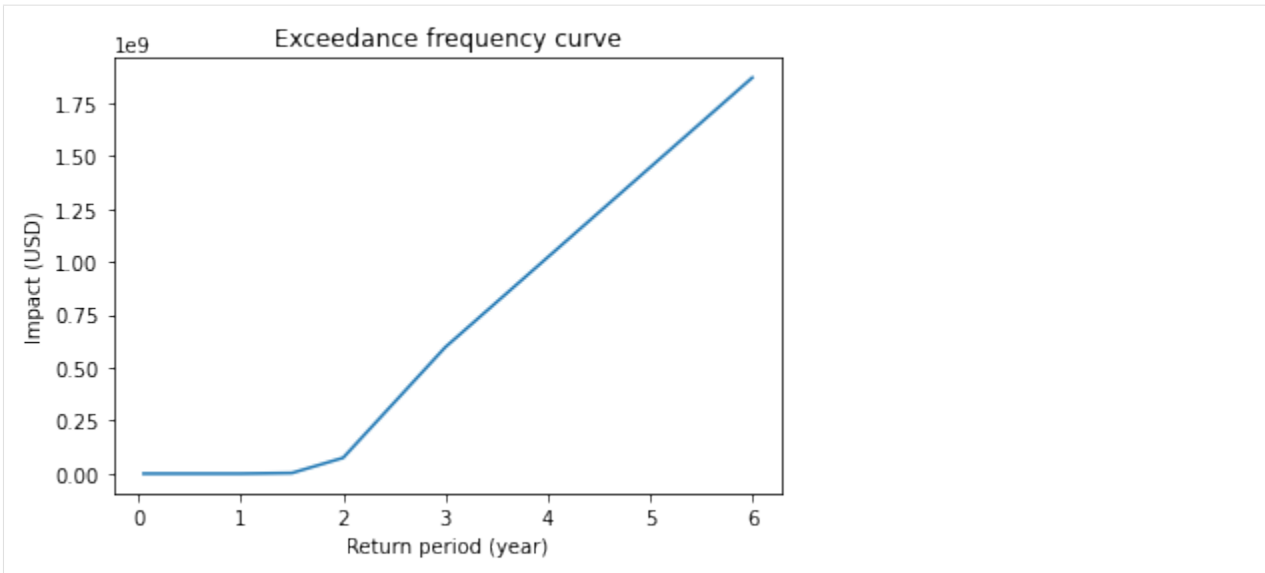
```
[14]: <GeoAxesSubplot:title={'center':'Expected annual impact'}>
```



```
[15]: # Compute exceedance frequency curve
```

```
freq_curve = imp.calc_freq_curve()
freq_curve.plot()
```

```
[15]: <AxesSubplot:title={'center':'Exceedance frequency curve'}, xlabel='Return period (year)
      ↳', ylabel='Impact (USD)'\>
```



Quick examples - points, raster, custom

User defined point exposure and Tropical Cyclone hazard

```
[16]: %matplotlib inline
# EXAMPLE: POINT EXPOSURES WITH POINT HAZARD
import numpy as np
from climada.entity import Exposures, ImpactFuncSet, IFTropCyclone
from climada.hazard import Centroids, TCTracks, TropCyclone
from climada.engine import Impact

# Set Exposures in points
exp_pnt = Exposures(crs='epsg:4326') #set coordinate system
exp_pnt.gdf['latitude'] = np.array([21.899326, 21.960728, 22.220574, 22.298390, 21.
    ↳ 787977, 21.787977, 21.981732])
exp_pnt.gdf['longitude'] = np.array([88.307422, 88.565362, 88.378337, 87.806356, 88.
    ↳ 348835, 88.348835, 89.246521])
exp_pnt.gdf['value'] = np.array([1.0e5, 1.2e5, 1.1e5, 1.1e5, 2.0e5, 2.5e5, 0.5e5])
exp_pnt.check()
exp_pnt.plot_scatter(buffer=0.05)

# Set Hazard in Exposures points
# set centroids from exposures coordinates
centr_pnt = Centroids()
centr_pnt.set_lat_lon(exp_pnt.gdf.latitude.values, exp_pnt.gdf.longitude.values, exp_pnt.
    ↳ crs)
# compute Hazard in that centroids
tr_pnt = TCTracks()
tr_pnt.read_ibtracs_netcdf(storm_id='2007314N10093')
tc_pnt = TropCyclone.from_tracks(tr_pnt, centroids=centr_pnt)
tc_pnt.check()
ax_pnt = tc_pnt.centroids.plot(c=np.array(tc_pnt.intensity[0,:].todense()).squeeze()) #
    ↳ plot intensity per point
```

(continues on next page)

(continued from previous page)

```

ax_pnt.get_figure().colorbar(ax_pnt.collections[0], fraction=0.0175, pad=0.02).set_label(
    ↪ 'Intensity (m/s)') # add colorbar

# Set impact function
impf_pnt = ImpactFuncSet()
impf_tc = IFtropCyclone()
impf_tc.set_emanuel_usa()
impf_pnt.append(impf_tc)
impf_pnt.check()

# Get the hazard type and hazard id
[haz_type] = impf_set.get_hazard_types()
[haz_id] = impf_set.get_ids()[haz_type]
# Exposures: rename column and assign id
exp_lp.gdf.rename(columns={"impf_": "impf_" + haz_type}, inplace=True)
exp_lp.gdf['impf_' + haz_type] = haz_id
exp_lp.gdf.head()

# Compute Impact
imp_pnt = Impact()
imp_pnt.calc(exp_pnt, impf_pnt, tc_pnt)
# nearest neighbor of exposures to centroids gives identity
print('Nearest neighbor hazard.centroids indexes for each exposure:', exp_pnt.gdf.cent_
    ↪ TC.values)
imp_pnt.plot_scatter_eai_exposure(ignore_zero=False, buffer=0.05)

```

```

2021-04-30 13:12:14,922 - climada.entity.exposures.base - INFO - meta set to default_
    ↪ value {}
2021-04-30 13:12:14,924 - climada.entity.exposures.base - INFO - tag set to default_
    ↪ value File:
    Description:
2021-04-30 13:12:14,934 - climada.entity.exposures.base - INFO - ref_year set to default_
    ↪ value 2018
2021-04-30 13:12:14,936 - climada.entity.exposures.base - INFO - value_unit set to_
    ↪ default value USD
2021-04-30 13:12:14,942 - climada.entity.exposures.base - INFO - Setting impf_ to_
    ↪ default impact functions ids 1.
2021-04-30 13:12:14,951 - climada.entity.exposures.base - INFO - category_id not set.
2021-04-30 13:12:14,953 - climada.entity.exposures.base - INFO - cover not set.
2021-04-30 13:12:14,955 - climada.entity.exposures.base - INFO - deductible not set.
2021-04-30 13:12:14,960 - climada.entity.exposures.base - INFO - geometry not set.
2021-04-30 13:12:14,961 - climada.entity.exposures.base - INFO - region_id not set.
2021-04-30 13:12:14,964 - climada.entity.exposures.base - INFO - centr_ not set.
2021-04-30 13:12:17,527 - climada.hazard.tc_tracks - INFO - Progress: 100%
2021-04-30 13:12:17,587 - climada.hazard.centroids.cent_ - INFO - Convert centroids to_
    ↪ GeoSeries of Point shapes.
2021-04-30 13:12:18,545 - climada.util.coordinates - INFO - dist_to_coast: UTM 32645 (1/
    ↪ 1)
2021-04-30 13:12:18,862 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 7_
    ↪ coastal centroids.
2021-04-30 13:12:18,871 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:12:19,305 - 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.

```

(continues on next page)

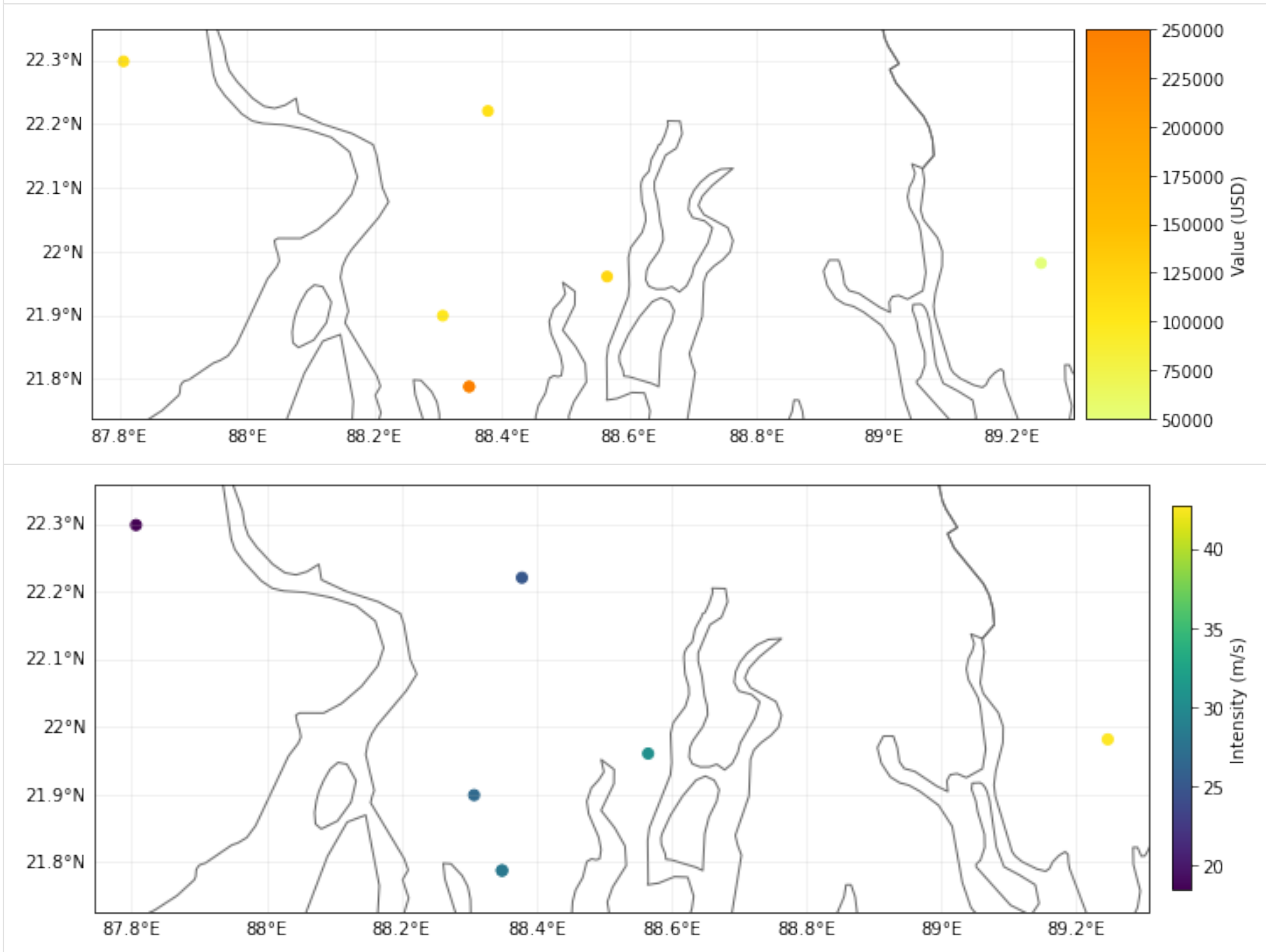
(continued from previous page)

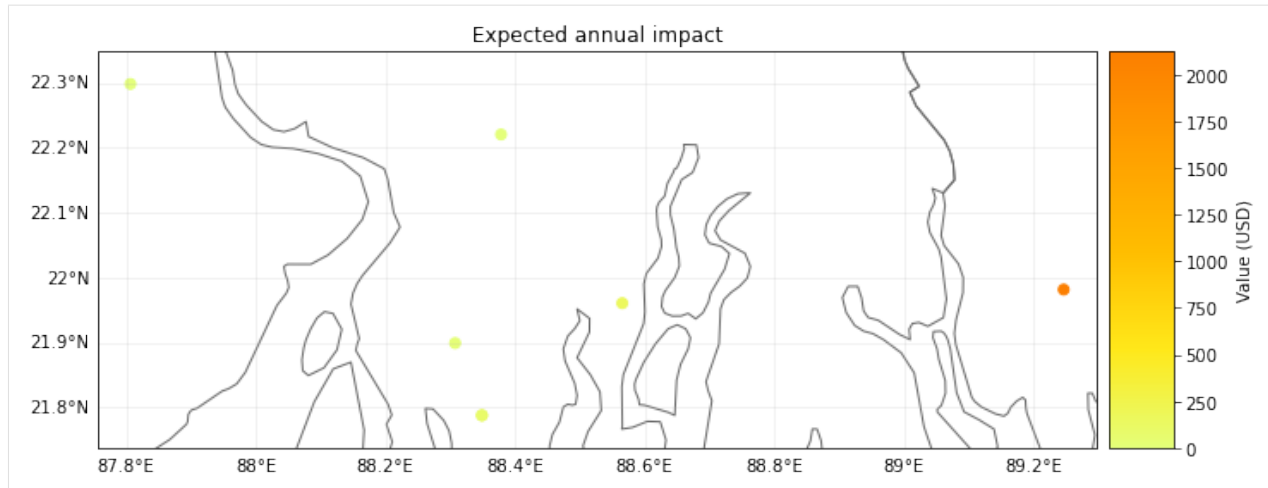
```

2021-04-30 13:12:19,308 - climada.entity.exposures.base - INFO - Matching 7 exposures.
↳with 7 centroids.
2021-04-30 13:12:19,314 - climada.engine.impact - INFO - Calculating damage for 7 assets.
↳(>0) and 1 events.
2021-04-30 13:12:19,315 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.
Nearest neighbor hazard.centroids indexes for each exposure: [0 1 2 3 4 5 6]

```

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





Raster from file

```
[17]: # EXAMPLE: RASTER EXPOSURES WITH RASTER HAZARD
from rasterio.warp import Resampling
from climada.entity import LitPop, ImpactFuncSet, ImpactFunc
from climada.hazard import Hazard
from climada.engine import Impact
from climada.util.constants import HAZ_DEMO_FL

# Exposures belonging to a raster (the raster information is contained in the meta_
↳ attribute)
exp_ras = LitPop()
exp_ras.set_country(countries=['VEN'], res_arcsec=300, fin_mode='income_group')
exp_ras.gdf.reset_index()
exp_ras.check()
exp_ras.plot_raster()
print('\n Raster properties exposures:', exp_ras.meta)

# Initialize hazard object with haz_type = 'FL' (for Flood)
hazard_type='FL'
haz_ras = Hazard(haz_type=hazard_type)
# Load a previously generated (either with CLIMADA or other means) hazard
# from file (HAZ_DEMO_FL) and resample the hazard raster to the exposures' ones
# Hint: check how other resampling methods affect to final impact
haz_ras.set_raster([HAZ_DEMO_FL], dst_crs=exp_ras.meta['crs'], transform=exp_ras.meta[
↳ 'transform'],
                    width=exp_ras.meta['width'], height=exp_ras.meta['height'],
                    resampling=Resampling.nearest)
haz_ras.intensity[haz_ras.intensity==-9999] = 0 # correct no data values
haz_ras.check()
haz_ras.plot_intensity(1)
print('Raster properties centroids:', haz_ras.centroids.meta)

# Set dummy impact function
impf_dum = ImpactFunc()
```

(continues on next page)

(continued from previous page)

```

impf_dum.id = haz_id
impf_dum.name = 'dummy'
impf_dum.intensity_unit = 'm'
impf_dum.haz_type = hazard_type
impf_dum.intensity = np.linspace(0, 10, 100)
impf_dum.mdd = np.linspace(0, 10, 100)
impf_dum.paa = np.ones(impf_dum.intensity.size)
# Add the impact function to the impact function set
impf_ras = ImpactFuncSet()
impf_ras.append(impf_dum)
impf_ras.check()

# Exposures: rename column and assign id
exp_lp.gdf.rename(columns={"impf_": "impf_" + hazard_type}, inplace=True)
exp_lp.gdf['impf_' + haz_type] = haz_id
exp_lp.gdf.head()

# Compute impact
imp_ras = Impact()
imp_ras.calc(exp_ras, impf_ras, haz_ras, save_mat=False)
# nearest neighbor of exposures to centroids is not identity because litpop does not
↳ contain data outside the country polygon
print('\n Nearest neighbor hazard.centroids indexes for each exposure:', exp_ras.gdf.
↳ centr_FL.values)
imp_ras.plot_raster_eai_exposure()

2021-04-30 13:12:24,137 - climada.entity.exposures.base - INFO - meta set to default
↳ value {}
2021-04-30 13:12:24,142 - climada.entity.exposures.base - INFO - tag set to default
↳ value File:
Description:
2021-04-30 13:12:24,145 - climada.entity.exposures.base - INFO - ref_year set to default
↳ value 2018
2021-04-30 13:12:24,146 - climada.entity.exposures.base - INFO - value_unit set to
↳ default value USD
2021-04-30 13:12:24,150 - climada.entity.exposures.base - INFO - crs set to default
↳ value: EPSG:4326
2021-04-30 13:12:25,637 - climada.entity.exposures.litpop - INFO - Generating LitPop
↳ data at a resolution of 300.0 arcsec.
2021-04-30 13:12:29,349 - climada.entity.exposures.gpw_import - INFO - Reference year:
↳ 2016. Using nearest available year for GPW population data: 2015
2021-04-30 13:12:29,350 - climada.entity.exposures.gpw_import - INFO - GPW Version v4.11
2021-04-30 13:12:46,135 - climada.util.finance - INFO - GDP VEN 2014: 4.824e+11.
2021-04-30 13:12:46,205 - climada.util.finance - INFO - Income group VEN 2016: 3.
2021-04-30 13:12:46,756 - climada.entity.exposures.base - INFO - meta set to default
↳ value {}
2021-04-30 13:12:46,757 - climada.entity.exposures.base - INFO - tag set to default
↳ value File:
Description:
2021-04-30 13:12:46,758 - climada.entity.exposures.base - INFO - ref_year set to default
↳ value 2018
2021-04-30 13:12:46,759 - climada.entity.exposures.base - INFO - value_unit set to
↳ default value USD

```

(continues on next page)

(continued from previous page)

```

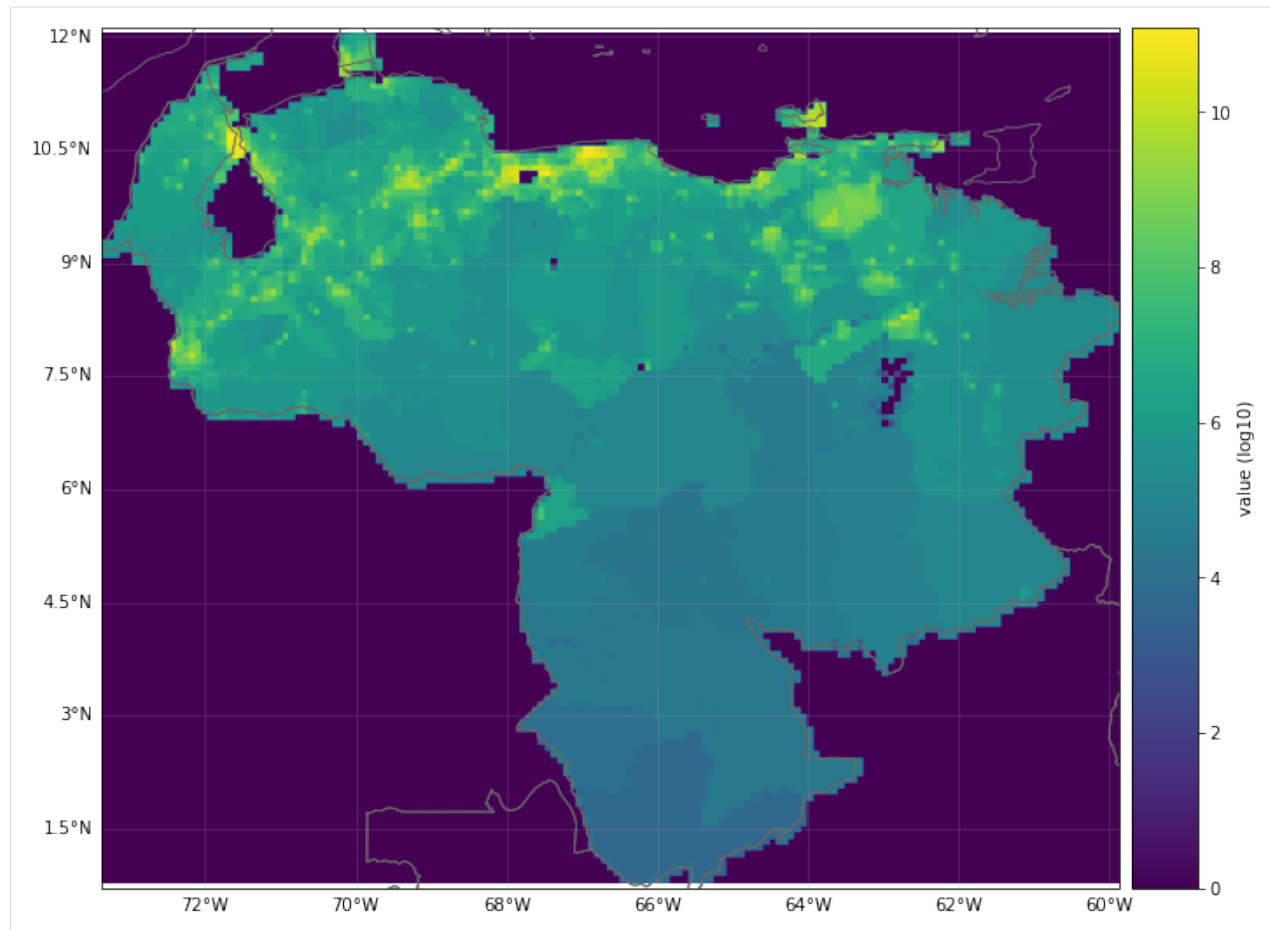
2021-04-30 13:12:46,769 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-04-30 13:12:46,795 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-30 13:12:46,802 - climada.entity.exposures.base - INFO - Hazard type not set in.
↳impf_
2021-04-30 13:12:46,803 - climada.entity.exposures.base - INFO - category_id not set.
2021-04-30 13:12:46,805 - climada.entity.exposures.base - INFO - cover not set.
2021-04-30 13:12:46,808 - climada.entity.exposures.base - INFO - deductible not set.
2021-04-30 13:12:46,809 - climada.entity.exposures.base - INFO - geometry not set.
2021-04-30 13:12:46,810 - climada.entity.exposures.base - INFO - centr_ not set.
2021-04-30 13:12:46,832 - climada.entity.exposures.base - INFO - Hazard type not set in.
↳impf_
2021-04-30 13:12:46,834 - climada.entity.exposures.base - INFO - category_id not set.
2021-04-30 13:12:46,836 - climada.entity.exposures.base - INFO - cover not set.
2021-04-30 13:12:46,837 - climada.entity.exposures.base - INFO - deductible not set.
2021-04-30 13:12:46,839 - climada.entity.exposures.base - INFO - geometry not set.
2021-04-30 13:12:46,840 - climada.entity.exposures.base - INFO - centr_ not set.
2021-04-30 13:12:46,846 - climada.util.coordinates - INFO - Raster from resolution 0.
↳083333333333286 to 0.083333333333286.

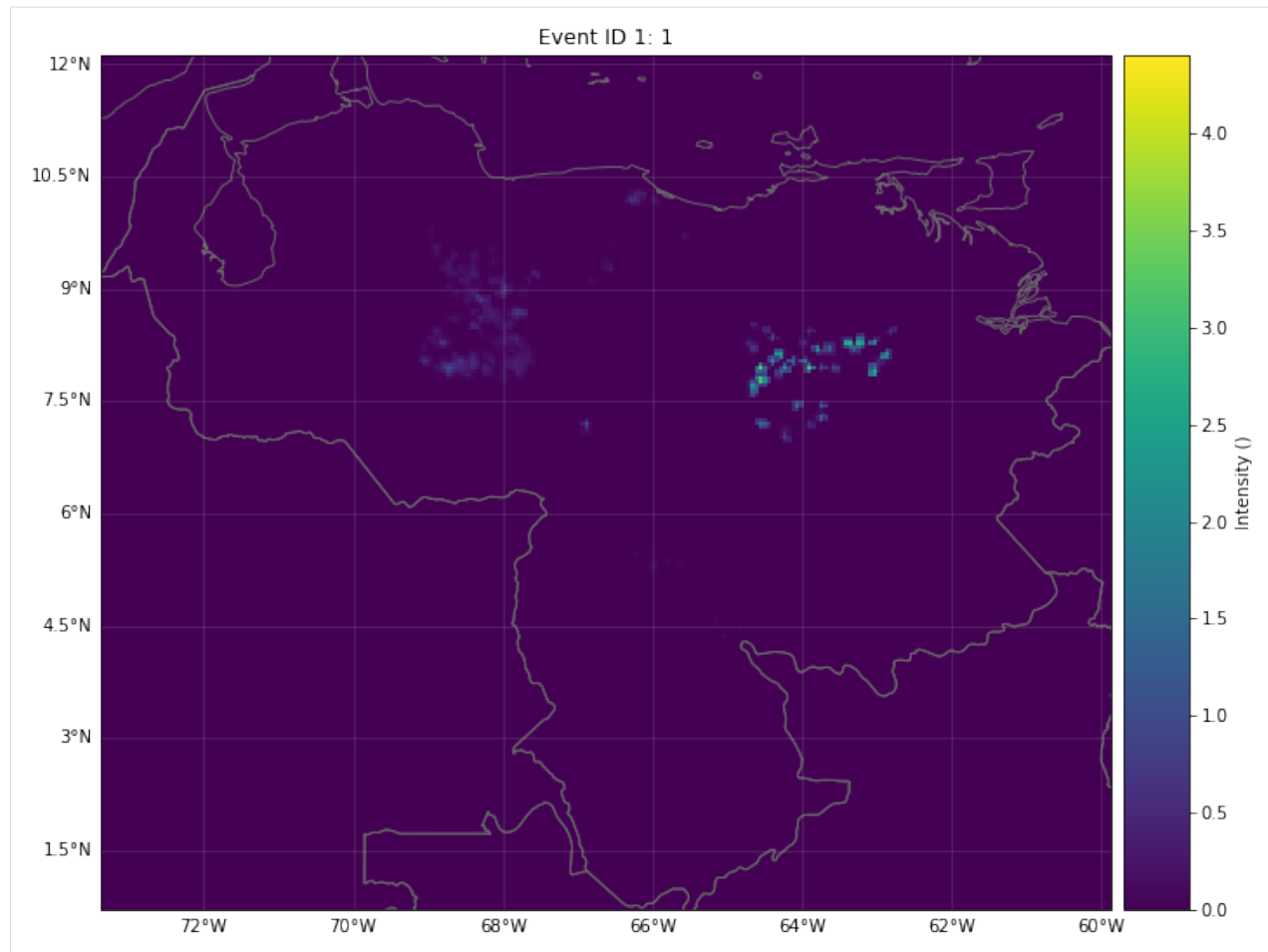
Raster properties exposures: {'width': 163, 'height': 138, 'crs': 'EPSG:4326',
↳'transform': Affine(0.083333333333286, 0.0, -73.41666666666669,
0.0, -0.083333333333286, 12.166666666666664)}
2021-04-30 13:12:53,032 - climada.util.coordinates - INFO - Reading C:\Users\me\climada\
↳demo\data\SC22000_VE_M1.grd.gz
Raster properties centroids: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.
↳701410009187828e+38, 'width': 163, 'height': 138, 'count': 1, 'crs': 'EPSG:4326',
↳'transform': Affine(0.083333333333286, 0.0, -73.41666666666669,
0.0, -0.083333333333286, 12.166666666666664)}
2021-04-30 13:12:56,952 - 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.
2021-04-30 13:12:56,957 - climada.entity.exposures.base - INFO - Matching 10770.
↳exposures with 22494 centroids.
2021-04-30 13:12:56,965 - climada.engine.impact - INFO - Calculating damage for 10717.
↳assets (>0) and 1 events.
2021-04-30 13:12:56,967 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard FL. Using the anonymous 'impf_' column.

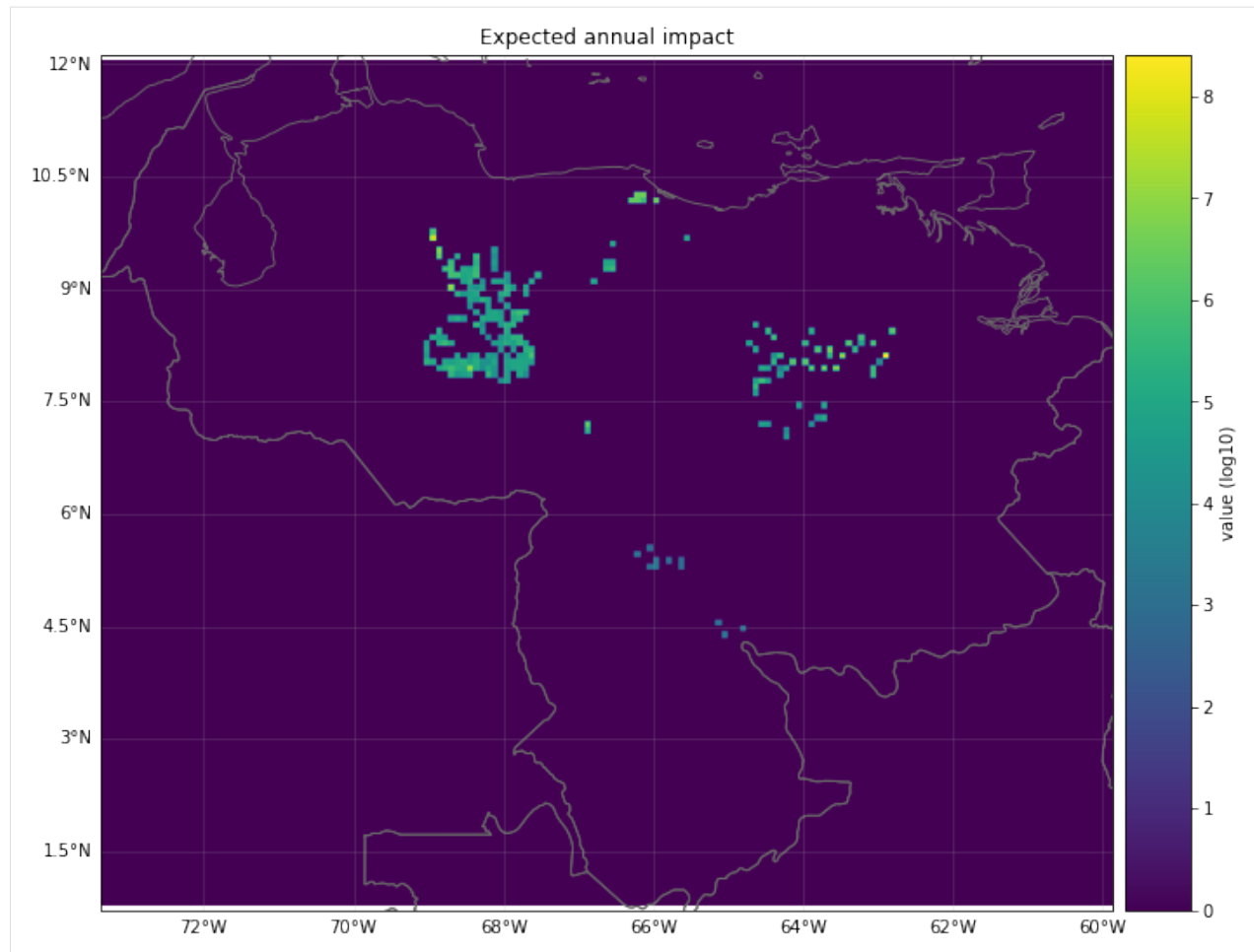
Nearest neighbor hazard.centroids indexes for each exposure: [5705 5543 5706 ... 7659.
↳7822 7660]
2021-04-30 13:12:56,979 - climada.util.coordinates - INFO - Raster from resolution 0.
↳083333333333286 to 0.083333333333286.

```

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







VISUALIZATION

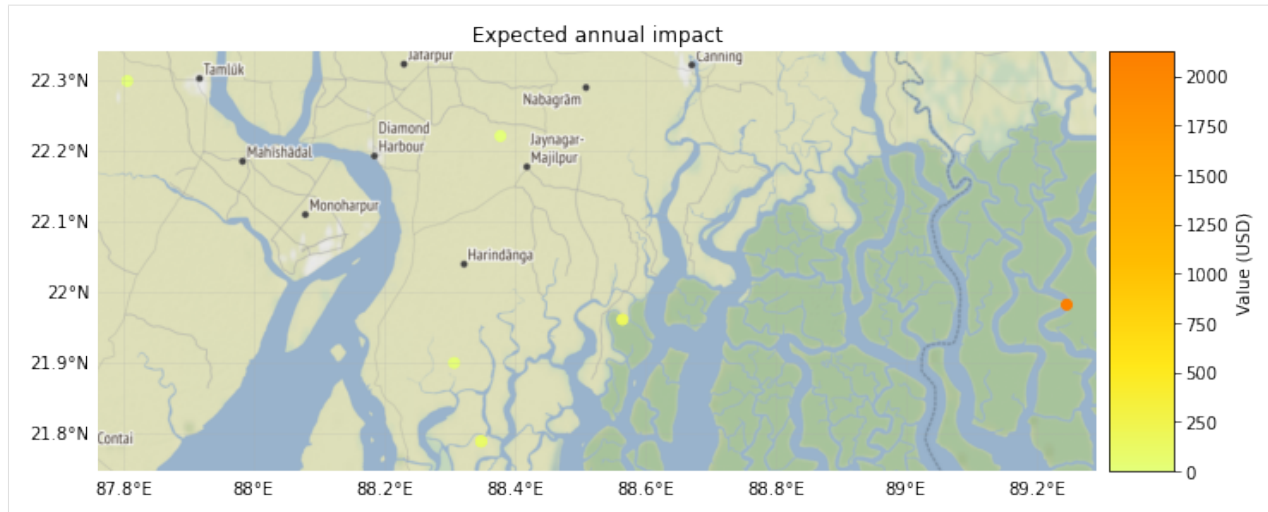
Making plots

The expected annual impact per exposure can be visualized through different methods: `plot_hexbin_eai_exposure()`, `plot_scatter_eai_exposure()`, `plot_raster_eai_exposure()` and `plot_basemap_eai_exposure()` (similarly as with Exposures).

```
[18]: imp_pnt.plot_basemap_eai_exposure(buffer=5000)
```

```
2021-04-30 13:13:06,755 - climada.util.coordinates - INFO - Setting geometry points.
2021-04-30 13:13:06,875 - climada.entity.exposures.base - INFO - Setting latitude and
↳ longitude attributes.
2021-04-30 13:13:08,546 - climada.entity.exposures.base - INFO - Setting latitude and
↳ longitude attributes.
```

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



Making videos

Given a fixed exposure and impact functions, a sequence of hazards can be visualized hitting the exposures.

```
[19]: # exposure
from climada.entity import BlackMarble, add_sea

exp_video = BlackMarble()
exp_video.set_countries(['Cuba'], 2016, res_km=2.5)
exp_video.check()

# impact function
impf_def = IFTropCyclone()
impf_def.set_emanuel_usa()
impfs_video = ImpactFuncSet()
impfs_video.append(impf_def)
impfs_video.check()

# compute sequence of hazards using TropCyclone video_intensity method
exp_sea = add_sea(exp_video, (100, 5))
centr_video = Centroids()
centr_video.set_lat_lon(exp_sea.gdf.latitude.values, exp_sea.gdf.longitude.values)
centr_video.check()

track_name = '2017242N16333'
tr_irma = TCTracks()
tr_irma.read_ibtracs_netcdf(provider='usa', storm_id=track_name) # IRMA 2017

tc_video = TropCyclone()
tc_list, _ = tc_video.video_intensity(track_name, tr_irma, centr_video) # empty file
↳ name to not to write the video

# generate video of impacts
file_name='./results/irma_imp_fl.gif'
imp_video = Impact()
```

(continues on next page)

(continued from previous page)

```

imp_list = imp_video.video_direct_impact(exp_video, impfs_video, tc_list, file_name)

2021-04-30 13:13:09,080 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-30 13:13:09,086 - climada.entity.exposures.base - INFO - tag set to default.
↳value File:
Description:
2021-04-30 13:13:09,088 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-04-30 13:13:09,097 - climada.entity.exposures.base - INFO - value_unit set to
↳default value USD
2021-04-30 13:13:09,100 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-04-30 13:13:10,128 - climada.util.finance - INFO - GDP CUB 2016: 9.137e+10.
2021-04-30 13:13:10,197 - climada.util.finance - INFO - Income group CUB 2016: 3.
2021-04-30 13:13:10,198 - climada.entity.exposures.black_marble - INFO - Nightlights
↳from NASA's earth observatory for year 2016.
2021-04-30 13:13:18,224 - climada.entity.exposures.black_marble - INFO - Processing
↳country Cuba.
2021-04-30 13:13:19,316 - climada.entity.exposures.black_marble - INFO - Generating
↳resolution of approx 2.5 km.
2021-04-30 13:13:19,478 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-30 13:13:19,479 - climada.entity.exposures.base - INFO - tag set to default.
↳value File:
Description:
2021-04-30 13:13:19,480 - climada.entity.exposures.base - INFO - ref_year set to default.
↳value 2018
2021-04-30 13:13:19,481 - climada.entity.exposures.base - INFO - value_unit set to
↳default value USD
2021-04-30 13:13:19,485 - climada.entity.exposures.base - INFO - crs set to default.
↳value: EPSG:4326
2021-04-30 13:13:19,511 - climada.entity.exposures.base - INFO - meta set to default.
↳value {}
2021-04-30 13:13:19,522 - climada.entity.exposures.base - INFO - Hazard type not set in
↳impf_
2021-04-30 13:13:19,525 - climada.entity.exposures.base - INFO - category_id not set.
2021-04-30 13:13:19,528 - climada.entity.exposures.base - INFO - cover not set.
2021-04-30 13:13:19,529 - climada.entity.exposures.base - INFO - deductible not set.
2021-04-30 13:13:19,530 - climada.entity.exposures.base - INFO - geometry not set.
2021-04-30 13:13:19,532 - climada.entity.exposures.base - INFO - centr_ not set.
2021-04-30 13:13:19,534 - 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.
2021-04-30 13:13:19,536 - climada.entity.exposures.base - INFO - Adding sea at 5 km
↳resolution and 100 km distance from coast.
2021-04-30 13:13:20,980 - climada.hazard.tc_tracks - INFO - Progress: 100%
2021-04-30 13:13:21,016 - climada.hazard.centroids.centri - INFO - Convert centroids to
↳GeoSeries of Point shapes.
2021-04-30 13:13:33,062 - climada.util.coordinates - INFO - dist_to_coast: UTM 32616 (1/
↳3)

```

(continues on next page)

(continued from previous page)

```

2021-04-30 13:13:40,527 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617 (2/
↳3)
2021-04-30 13:14:01,057 - climada.util.coordinates - INFO - dist_to_coast: UTM 32618 (3/
↳3)
2021-04-30 13:14:11,473 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 22776_
↳coastal centroids.
2021-04-30 13:14:11,509 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:11,527 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 25701_
↳coastal centroids.
2021-04-30 13:14:11,563 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:11,580 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 29239_
↳coastal centroids.
2021-04-30 13:14:11,633 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:11,648 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 32187_
↳coastal centroids.
2021-04-30 13:14:11,691 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:11,710 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 34921_
↳coastal centroids.
2021-04-30 13:14:11,763 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:11,777 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 37244_
↳coastal centroids.
2021-04-30 13:14:11,831 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:11,845 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 39418_
↳coastal centroids.
2021-04-30 13:14:11,897 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:11,910 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 42155_
↳coastal centroids.
2021-04-30 13:14:11,966 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:11,981 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 43662_
↳coastal centroids.
2021-04-30 13:14:12,054 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,067 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 45523_
↳coastal centroids.
2021-04-30 13:14:12,132 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,151 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 47105_
↳coastal centroids.
2021-04-30 13:14:12,211 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,227 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 48082_
↳coastal centroids.
2021-04-30 13:14:12,291 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,304 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 48019_
↳coastal centroids.
2021-04-30 13:14:12,375 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,389 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 47081_
↳coastal centroids.
2021-04-30 13:14:12,457 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,469 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 45784_
↳coastal centroids.
2021-04-30 13:14:12,534 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,547 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 44307_
↳coastal centroids.
2021-04-30 13:14:12,598 - climada.hazard.trop_cyclone - INFO - Progress: 100%

```

(continues on next page)

(continued from previous page)

```

2021-04-30 13:14:12,613 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 43081_
↳coastal centroids.
2021-04-30 13:14:12,676 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,691 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 42086_
↳coastal centroids.
2021-04-30 13:14:12,751 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,768 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 41313_
↳coastal centroids.
2021-04-30 13:14:12,827 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,844 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 40713_
↳coastal centroids.
2021-04-30 13:14:12,896 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,911 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 40023_
↳coastal centroids.
2021-04-30 13:14:12,964 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2021-04-30 13:14:12,978 - climada.entity.exposures.base - INFO - Matching 21923_
↳exposures with 49817 centroids.
2021-04-30 13:14:13,046 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,047 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,054 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,059 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,060 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,067 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,071 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,073 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,082 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,085 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,089 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,097 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,100 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,101 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,109 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,117 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,123 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,132 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC

```

(continues on next page)

(continued from previous page)

```

2021-04-30 13:14:13,135 - climada.engine.impact - INFO - Calculating damage for 14654.
↳assets (>0) and 1 events.
2021-04-30 13:14:13,139 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,147 - climada.engine.impact - INFO - Exposures matching centroids.
↳found in centr_TC
2021-04-30 13:14:13,151 - climada.engine.impact - INFO - Calculating damage for 14654.
↳assets (>0) and 1 events.
2021-04-30 13:14:13,153 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,163 - climada.engine.impact - INFO - Exposures matching centroids.
↳found in centr_TC
2021-04-30 13:14:13,168 - climada.engine.impact - INFO - Calculating damage for 14654.
↳assets (>0) and 1 events.
2021-04-30 13:14:13,171 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,180 - climada.engine.impact - INFO - Exposures matching centroids.
↳found in centr_TC
2021-04-30 13:14:13,184 - climada.engine.impact - INFO - Calculating damage for 14654.
↳assets (>0) and 1 events.
2021-04-30 13:14:13,187 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,198 - climada.engine.impact - INFO - Exposures matching centroids.
↳found in centr_TC
2021-04-30 13:14:13,202 - climada.engine.impact - INFO - Calculating damage for 14654.
↳assets (>0) and 1 events.
2021-04-30 13:14:13,205 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,214 - climada.engine.impact - INFO - Exposures matching centroids.
↳found in centr_TC
2021-04-30 13:14:13,218 - climada.engine.impact - INFO - Calculating damage for 14654.
↳assets (>0) and 1 events.
2021-04-30 13:14:13,219 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,228 - climada.engine.impact - INFO - Exposures matching centroids.
↳found in centr_TC
2021-04-30 13:14:13,233 - climada.engine.impact - INFO - Calculating damage for 14654.
↳assets (>0) and 1 events.
2021-04-30 13:14:13,234 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,247 - climada.engine.impact - INFO - Exposures matching centroids.
↳found in centr_TC
2021-04-30 13:14:13,251 - climada.engine.impact - INFO - Calculating damage for 14654.
↳assets (>0) and 1 events.
2021-04-30 13:14:13,253 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,262 - climada.engine.impact - INFO - Exposures matching centroids.
↳found in centr_TC
2021-04-30 13:14:13,266 - climada.engine.impact - INFO - Calculating damage for 14654.
↳assets (>0) and 1 events.
2021-04-30 13:14:13,267 - climada.entity.exposures.base - INFO - No specific impact.
↳function column found for hazard TC. Using the anonymous 'impf_' column.

```

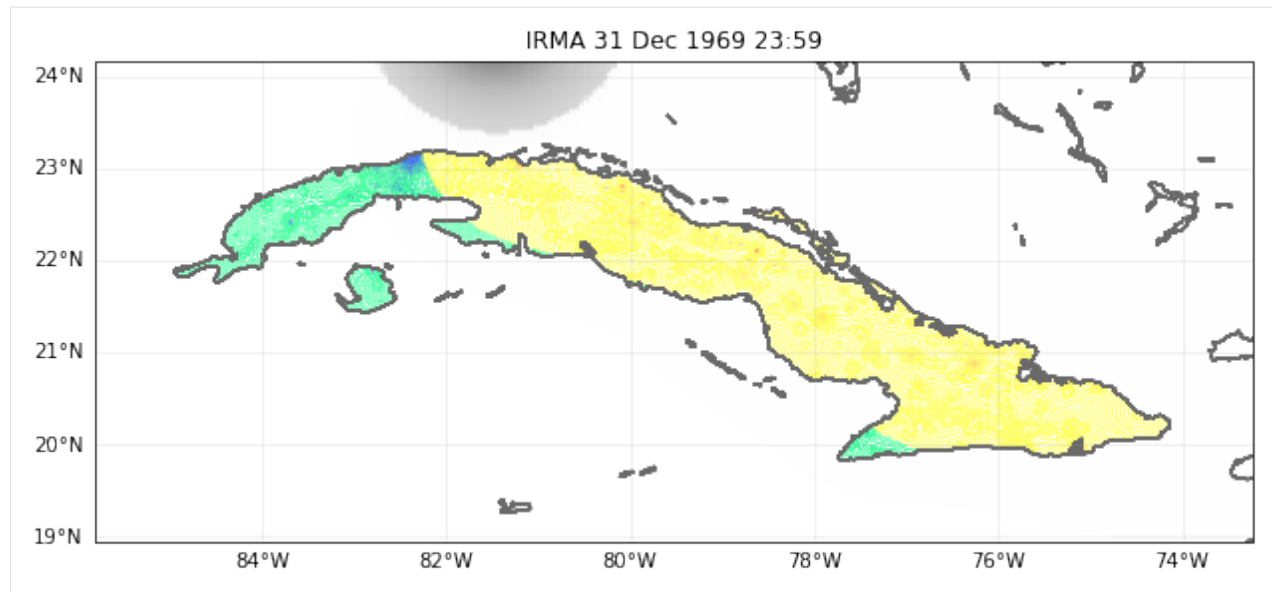
(continues on next page)

(continued from previous page)

```

2021-04-30 13:14:13,276 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,279 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,281 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,288 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,291 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,293 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,300 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,304 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,305 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,314 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,320 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,321 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,328 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,332 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,333 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,342 - climada.engine.impact - INFO - Exposures matching centroids_
↳found in centr_TC
2021-04-30 13:14:13,345 - climada.engine.impact - INFO - Calculating damage for 14654_
↳assets (>0) and 1 events.
2021-04-30 13:14:13,346 - climada.entity.exposures.base - INFO - No specific impact_
↳function column found for hazard TC. Using the anonymous 'impf_' column.
2021-04-30 13:14:13,355 - climada.engine.impact - INFO - Generating video ./results/irma_
↳imp_fl.gif
22it [09:40, 26.39s/it]

```

5.12 Impact Data functionalities

Import data from EM-DAT CSV file and populate Impact()-object with the data.

The core functionality of the module is to read disaster impact data as downloaded from the International Disaster Database EM-DAT (www.emdat.be) and produce a CLIMADA Impact()-instance from it. The purpose is to make impact data easily available for comparison with simulated impact inside CLIMADA, e.g. for calibration purposes.

5.12.1 Data Source

The International Disaster Database EM-DAT www.emdat.be

Download: <https://public.emdat.be/> (register for free and download data to continue)

5.12.2 Most important functions

- `clean_emdat_df`: read CSV from EM-DAT into a DataFrame and clean up.
- `emdat_to_impact`: create Impact-instance populated with impact data from EM-DAT data (CSV).
- `emdat_countries_by_hazard`: get list of countries affected by a certain hazard (disaster (sub-)type) in EM-DAT.
- `emdat_impact_yearlysum`: create DataFrame with impact from EM-DAT summed per country and year.

5.12.3 Demo data

The demo data used here (demo_emdat_impact_data_2020.csv) contains entries for the disaster subtype “Tropical cyclone” from 2000 to 2020.

```
[2]: """Load required packages and set path to CSV-file from EM-DAT"""

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

from climada.util.constants import DEMO_DIR
from climada.engine.impact_data import emdat_countries_by_hazard, \
    emdat_impact_yearlysum, emdat_to_impact, clean_emdat_df

# set path to CSV file downloaded from https://public.emdat.be :
emdat_file_path = DEMO_DIR.joinpath('demo_emdat_impact_data_2020.csv')
```

clean_emdat_df()

read CSV from EM-DAT into a DataFrame and clean up.

Use the parameters countries, hazard, and year_range to filter. These parameters are the same for most functions shown here.

```
[12]: """Create DataFrame df with EM-DAT entries of tropical cyclones in Thailand and Viet Nam,
↳ in the years 2005 and 2006"""

df = clean_emdat_df(emdat_file_path, countries=['THA', 'Viet Nam'], hazard=['TC'], \
    year_range=[2005, 2006])
print(df)
```

	Dis No	Year	Seq	Disaster Group	Disaster Subgroup	Disaster Type	\
0	2005-0540-VNM	2005	540	Natural	Meteorological	Storm	
1	2005-0540-THA	2005	540	Natural	Meteorological	Storm	
2	2005-0536-VNM	2005	536	Natural	Meteorological	Storm	
3	2005-0611-VNM	2005	611	Natural	Meteorological	Storm	
4	2006-0362-VNM	2006	362	Natural	Meteorological	Storm	
5	2006-0648-VNM	2006	648	Natural	Meteorological	Storm	
6	2006-0251-VNM	2006	251	Natural	Meteorological	Storm	
7	2006-0517-VNM	2006	517	Natural	Meteorological	Storm	

	Disaster Subtype	Disaster Subsubtype	Event Name	Entry Criteria	\
0	Tropical cyclone	NaN	Damrey	Kill	
1	Tropical cyclone	NaN	Damrey	Kill	
2	Tropical cyclone	NaN	Vicente	Kill	
3	Tropical cyclone	NaN	Kai Tak (21)	Kill	
4	Tropical cyclone	NaN	Bilis	Kill	
5	Tropical cyclone	NaN	Durian (Reming)	Kill	
6	Tropical cyclone	NaN	Chanchu (Caloy)	Kill	
7	Tropical cyclone	NaN	Xangsane (Milenyo)	Kill	

... End Day Total Deaths No Injured No Affected No Homeless Total Affected \

(continues on next page)

(continued from previous page)

0	...	30.0	75.0	28.0	337632.0	NaN	337660.0
1	...	30.0	10.0	NaN	2000.0	NaN	2000.0
2	...	19.0	8.0	NaN	8500.0	NaN	8500.0
3	...	4.0	20.0	NaN	15000.0	NaN	15000.0
4	...	19.0	17.0	NaN	NaN	2000.0	2000.0
5	...	8.0	95.0	1360.0	975000.0	250000.0	1226360.0
6	...	17.0	204.0	NaN	600000.0	NaN	600000.0
7	...	6.0	71.0	525.0	1368720.0	98680.0	1467925.0

	Reconstruction Costs ('000 US\$)	Insured Damages ('000 US\$)	\
0		NaN	NaN
1		NaN	NaN
2		NaN	NaN
3		NaN	NaN
4		NaN	NaN
5		NaN	NaN
6		NaN	NaN
7		NaN	NaN

	Total Damages ('000 US\$)	CPI
0	219250.0	76.388027
1	20000.0	76.388027
2	20000.0	76.388027
3	11000.0	76.388027
4	NaN	78.852256
5	456000.0	78.852256
6	NaN	78.852256
7	624000.0	78.852256

[8 rows x 43 columns]

emdat_countries_by_hazard()

Pick a hazard and a year range to get a list of countries affected from the EM-DAT data.

```
[2]: """emdat_countries_by_hazard: get lists of countries impacted by tropical cyclones from
↳ 2010 to 2019"""

iso3_codes, country_names = emdat_countries_by_hazard(emdat_file_path, hazard='TC', year_
↳ range=(2010, 2019))

print(country_names)

print(iso3_codes)
```

```
['China', 'Dominican Republic', 'Antigua and Barbuda', 'Fiji', 'Australia', 'Bangladesh',  
→ 'Belize', 'Barbados', 'Cook Islands', 'Canada', 'Bahamas', 'Guatemala', 'Jamaica',  
→ 'Saint Lucia', 'Madagascar', 'Mexico', 'Korea, Democratic People's Republic of', 'El_  
→ Salvador', 'Myanmar', 'French Polynesia', 'Solomon Islands', 'Taiwan, Province of China  
→ ', 'India', 'United States of America', 'Honduras', 'Haiti', 'Pakistan', 'Philippines',  
→ 'Hong Kong', 'Korea, Republic of', 'Nicaragua', 'Oman', 'Japan', 'Puerto Rico',  
→ 'Thailand', 'Martinique', 'Papua New Guinea', 'Tonga', 'Venezuela, Bolivarian Republic_  
→ of', 'Viet Nam', 'Saint Vincent and the Grenadines', 'Vanuatu', 'Dominica', 'Cuba',  
→ 'Comoros', 'Mozambique', 'Malawi', 'Samoa', 'South Africa', 'Sri Lanka', 'Palau',  
→ 'Wallis and Futuna', 'Somalia', 'Seychelles', 'Réunion', 'Kiribati', 'Cabo Verde',  
→ 'Micronesia, Federated States of', 'Panama', 'Costa Rica', 'Yemen', 'Tuvalu',  
→ 'Northern Mariana Islands', 'Colombia', 'Anguilla', 'Djibouti', 'Cambodia', 'Macao',  
→ 'Indonesia', 'Guadeloupe', 'Turks and Caicos Islands', 'Saint Kitts and Nevis', 'Lao_  
→ People's Democratic Republic', 'Mauritius', 'Marshall Islands', 'Portugal', 'Virgin_  
→ Islands, U.S.', 'Zimbabwe', 'Saint Barthélemy', 'Virgin Islands, British', 'Saint_  
→ Martin (French part)', 'Sint Maarten (Dutch part)', 'Tanzania, United Republic of']  
['CHN', 'DOM', 'ATG', 'FJI', 'AUS', 'BGD', 'BLZ', 'BRB', 'COK', 'CAN', 'BHS', 'GTM', 'JAM  
→ ', 'LCA', 'MDG', 'MEX', 'PRK', 'SLV', 'MMR', 'PYF', 'SLB', 'TWN', 'IND', 'USA', 'HND',  
→ 'HTI', 'PAK', 'PHL', 'HKG', 'KOR', 'NIC', 'OMN', 'JPN', 'PRI', 'THA', 'MTQ', 'PNG',  
→ 'TON', 'VEN', 'VNM', 'VCT', 'VUT', 'DMA', 'CUB', 'COM', 'MOZ', 'MWI', 'WSM', 'ZAF',  
→ 'LKA', 'PLW', 'WLF', 'SOM', 'SYC', 'REU', 'KIR', 'CPV', 'FSM', 'PAN', 'CRI', 'YEM',  
→ 'TUV', 'MNP', 'COL', 'AIA', 'DJI', 'KHM', 'MAC', 'IDN', 'GLP', 'TCA', 'KNA', 'LAO',  
→ 'MUS', 'MHL', 'PRT', 'VIR', 'ZWE', 'BLM', 'VGB', 'MAF', 'SXM', 'TZA']
```

emdat_to_impact()

function to load EM-DAT impact data and return impact set with impact per event

Parameters:

- `emdat_file_csv` (str): Full path to EMDAT-file (CSV)
- `hazard_type_climada` (str): Hazard type abbreviation used in CLIMADA, e.g. 'TC'

Optional parameters:

- `hazard_type_emdat` (list or str): List of Disaster (sub-)type according EMDAT terminology or CLIMADA hazard type abbreviations. e.g. ['Wildfire', 'Forest fire'] or ['BF']
- `year_range` (list with 2 integers): start and end year e.g. [1980, 2017]
- `countries` (list of str): country ISO3-codes or names, e.g. ['JAM', 'CUB']. Set to None or ['all'] for all countries
- `reference_year` (int): reference year of exposures for normalization. Impact is scaled proportional to GDP to the value of the reference year. No scaling for `reference_year=0` (default)
- `imp_str` (str): Column name of impact metric in EMDAT CSV, e.g. 'Total Affected'; default = "Total Damages"

Returns:

- `impact_instance` (instance of `climada.engine.Impact`): `Impact()` instance (same format as output from CLIMADA impact computations). Values are scaled with GDP to reference_year if reference_year not equal 0. `impact_instance.eai_exp` holds expected annual impact for each country. `impact_instance.coord_exp` holds rough central coordinates for each country.
- `countries` (list): ISO3-codes of countries in same order as in `impact_instance.eai_exp`

```
[3]: """Global TC damages 2000 to 2009"""
```

```
impact_emdat, countries = emdat_to_impact(emdat_file_path, 'TC', year_range=(2000,2009))

print('Number of TC events in EM-DAT 2000 to 2009 globally: %i' %(impact_emdat.event_id.
↳size))
print('Global annual average monetary damage (AAI) from TCs as reported in EM-DAT 2000_
↳to 2009: USD billion %2.2f' \
      %(impact_emdat.aai_agg/1e9))
```

```
2020-07-10 14:18:25,584 - climada.engine.impact_data - WARNING - ISO3alpha code not_
↳found in iso_country: SPI
SPI
2020-07-10 14:18:26,995 - climada.engine.impact_data - ERROR - Country not found in iso_
↳country: SPI
Number of TC events in EM-DAT 2000 to 2009 globally: 533
Global annual average monetary damage (AAI) from TCs as reported in EM-DAT 2000 to 2009:
↳USD billion 38.07
```

```
[31]: """Total people affected by TCs in the Philippines in 2013:"""
```

```
# People affected
impact_emdat_PHL, countries = emdat_to_impact(emdat_file_path, 'TC', countries='PHL', \
      year_range=(2013,2013), imp_str="Total Affected")

print('Number of TC events in EM-DAT in the Philippines, 2013: %i' \
      %(impact_emdat_PHL.event_id.size))
print('\nPeople affected by TC events in the Philippines in 2013 (per event):')
print(impact_emdat_PHL.at_event)
print('\nPeople affected by TC events in the Philippines in 2013 (total):')
print(int(impact_emdat_PHL.aai_agg))

# Comparison to monetary damages:
impact_emdat_PHL_USD, _ = emdat_to_impact(emdat_file_path, 'TC', countries='PHL', \
      year_range=(2013,2013))

ax = plt.scatter(impact_emdat_PHL_USD.at_event, impact_emdat_PHL.at_event)
plt.title('Typhoon impacts in the Philippines, 2013')
plt.xlabel('Total Damage [USD]')
plt.ylabel('People Affected')
#plt.xscale('log')
#plt.yscale('log')
```

```
Number of TC events in EM-DAT in the Philippines, 2013: 8
```

(continues on next page)

(continued from previous page)

```

People affected by TC events in the Philippines in 2013 (per event):
[7.269600e+04 1.059700e+04 8.717550e+05 2.204430e+05 1.610687e+07
 3.596000e+03 3.957300e+05 2.628840e+05]

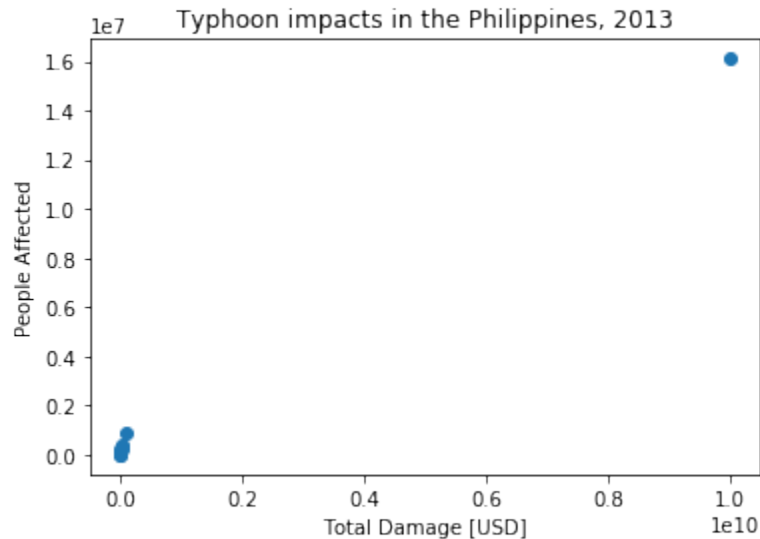
```

```

People affected by TC events in the Philippines in 2013 (total):
17944571

```

```
[31]: Text(0, 0.5, 'People Affected')
```



emdat_impact_yearlysum()

function to load EM-DAT impact data and return DataFrame with impact summed per year and country

Parameters:

- `emdat_file_csv` (str): Full path to EMDAT-file (CSV)

Optional parameters:

- `hazard` (list or str): List of Disaster (sub-)type according EMDAT terminology or CLIMADA hazard type abbreviations. e.g. ['Wildfire', 'Forest fire'] or ['BF']
- `year_range` (list with 2 integers): start and end year e.g. [1980, 2017]
- `countries` (list of str): country ISO3-codes or names, e.g. ['JAM', 'CUB']. Set to None or ['all'] for all countries
- `reference_year` (int): reference year of exposures for normalization. Impact is scaled proportional to GDP to the value of the reference year. No scaling for `reference_year=0` (default)
- `imp_str` (str): Column name of impact metric in EMDAT CSV, e.g. 'Total Affected'; default = "Total Damages"
- `version` (int): given EM-DAT data format version (i.e. year of download), changes naming of columns/variables (default: 2020)

Returns:

- pandas.DataFrame with impact per year and country

```
[5]: """Yearly TC damages in the USA, normalized and current"""

yearly_damage_normalized_to_2019 = emdat_impact_yearlysum(emdat_file_path, countries='USA',
↳ hazard='Tropical cyclone', year_
↳ range=None, \
reference_year=2019)

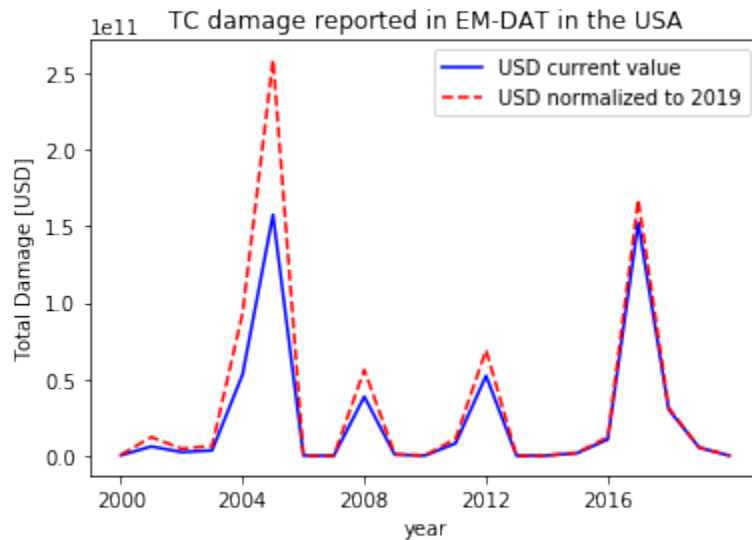
yearly_damage_current = emdat_impact_yearlysum(emdat_file_path, countries=['USA'],
↳ hazard='TC',)

import matplotlib.pyplot as plt

fig, axis = plt.subplots(1, 1)
axis.plot(yearly_damage_current.year, yearly_damage_current.impact, 'b', label='USD_
↳ current value')
axis.plot(yearly_damage_normalized_to_2019.year, yearly_damage_normalized_to_2019.impact_
↳ scaled, \
'r--', label='USD normalized to 2019')
plt.legend()
axis.set_title('TC damage reported in EM-DAT in the USA')
axis.set_xticks([2000, 2004, 2008, 2012, 2016])
axis.set_xlabel('year')
axis.set_ylabel('Total Damage [USD]')

2020-07-10 14:36:28,646 - climada.util.finance - INFO - GDP USA 2019: 2.143e+13.
[2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2014
2015 2016 2017 2018 2019 2020]
2020-07-10 14:36:29,099 - climada.util.finance - INFO - GDP USA 2000: 1.025e+13.
2020-07-10 14:36:29,539 - climada.util.finance - INFO - GDP USA 2001: 1.058e+13.
2020-07-10 14:36:30,302 - climada.util.finance - INFO - GDP USA 2002: 1.094e+13.
2020-07-10 14:36:30,754 - climada.util.finance - INFO - GDP USA 2003: 1.146e+13.
2020-07-10 14:36:31,204 - climada.util.finance - INFO - GDP USA 2004: 1.221e+13.
2020-07-10 14:36:31,648 - climada.util.finance - INFO - GDP USA 2005: 1.304e+13.
2020-07-10 14:36:32,098 - climada.util.finance - INFO - GDP USA 2006: 1.381e+13.
2020-07-10 14:36:32,549 - climada.util.finance - INFO - GDP USA 2007: 1.445e+13.
2020-07-10 14:36:33,037 - climada.util.finance - INFO - GDP USA 2008: 1.471e+13.
2020-07-10 14:36:33,485 - climada.util.finance - INFO - GDP USA 2009: 1.445e+13.
2020-07-10 14:36:33,927 - climada.util.finance - INFO - GDP USA 2010: 1.499e+13.
2020-07-10 14:36:34,785 - climada.util.finance - INFO - GDP USA 2011: 1.554e+13.
2020-07-10 14:36:35,227 - climada.util.finance - INFO - GDP USA 2012: 1.620e+13.
2020-07-10 14:36:35,674 - climada.util.finance - INFO - GDP USA 2014: 1.752e+13.
2020-07-10 14:36:36,125 - climada.util.finance - INFO - GDP USA 2015: 1.822e+13.
2020-07-10 14:36:36,701 - climada.util.finance - INFO - GDP USA 2016: 1.871e+13.
2020-07-10 14:36:37,145 - climada.util.finance - INFO - GDP USA 2017: 1.949e+13.
2020-07-10 14:36:37,591 - climada.util.finance - INFO - GDP USA 2018: 2.058e+13.
2020-07-10 14:36:38,029 - climada.util.finance - INFO - GDP USA 2019: 2.143e+13.
2020-07-10 14:36:38,861 - climada.util.finance - INFO - GDP USA 2019: 2.143e+13.
```

```
[5]: Text(0, 0.5, 'Total Damage [USD]')
```



5.13 Forecast class

This class deals with weather forecasts and uses CLIMADA `Impact.calc()` to forecast impacts of weather events on society. It mainly does one thing: - it contains all plotting and other functionality that are specific for weather forecasts, impact forecasts and warnings

The class is different from the `Impact` class especially because features of the `Impact` class like Exceedence frequency curves, annual average impact etc, do not make sense if the hazard is e.g. a 5 day weather forecast. As the class is relatively new, there might be future changes to the datastructure, the methods, and the parameters used to call the methods.

5.13.1 Example: forecast of building damages due to wind in Switzerland

Before using the forecast class, hazard, exposure and vulnerability need to be created. The hazard looks at the weather forecast from today for an event with two days lead time (meaning the day after tomorrow). `generate_WS_forecast_hazard` is used to download a current weather forecast for wind gust from `opendata.dwd.de`. An `Impact` function for building damages due to storms is created. And with only a few lines of code, a `LitPop` exposure for Switzerland is generated, and the impact is calculated with a default impact function. With a further line of code, the mean damage per grid point for the day after tomorrow is plotted on a map.

```
[1]: from datetime import datetime
from cartopy import crs as ccrs

from climada.util.config import CONFIG
from climada.engine.forecast import Forecast
from climada.hazard.storm_europe import StormEurope, generate_WS_forecast_hazard
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.entity import ImpactFuncSet
from climada.entity import LitPop
```



```

-----
ModuleNotFoundError                                Traceback (most recent call last)
/tmp/ipykernel_3138/2343566291.py in <module>
      2 from cartopy import crs as ccrs
      3
----> 4 from climada.util.config import CONFIG
      5 from climada.engine.forecast import Forecast
      6 from climada.hazard.storm_europe import StormEurope, generate_WS_forecast_hazard

ModuleNotFoundError: No module named 'climada'

```

```

[2]: #generate hazard
hazard, haz_model, run_datetime, event_date = generate_WS_forecast_hazard()
# #generate hazard with with forecasts from past dates (works only if the files have_
↳ already been downloaded)
# hazard, haz_model, run_datetime, event_date = generate_WS_forecast_hazard(
#     run_datetime=datetime(2021,3,7),
#     event_date=datetime(2021,3,11))

```

```

-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_3138/4178640579.py in <module>
      1 #generate hazard
----> 2 hazard, haz_model, run_datetime, event_date = generate_WS_forecast_hazard()
      3 # #generate hazard with with forecasts from past dates (works only if the files_
↳ have already been downloaded)
      4 # hazard, haz_model, run_datetime, event_date = generate_WS_forecast_hazard(
      5 #     run_datetime=datetime(2021,3,7),

NameError: name 'generate_WS_forecast_hazard' is not defined

```

```

[3]: #generate vulnerability
impact_function = ImpfStormEurope()
impact_function.set_welker()
impact_function_set = ImpactFuncSet()
impact_function_set.append(impact_function)

```

```

-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_3138/509005234.py in <module>
      1 #generate vulnerability
----> 2 impact_function = ImpfStormEurope()
      3 impact_function.set_welker()
      4 impact_function_set = ImpactFuncSet()
      5 impact_function_set.append(impact_function)

NameError: name 'ImpfStormEurope' is not defined

```

```

[4]: #generate exposure and save to file
filename_exp = CONFIG.local_data.save_dir.dir() / ('exp_' +
                                                    'litpop' +
                                                    '_' +
                                                    'Switzerland' +

```

(continues on next page)

(continued from previous page)

```

                                '.hdf5')
if filename_exp.exists():
    exposure = LitPop()
    exposure.read_hdf5(filename_exp)
else:
    exposure = LitPop()
    exposure.set_country('Switzerland', reference_year=2020)
    exposure.write_hdf5(filename_exp)

```

```

-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_3138/3799798359.py in <module>
      1 #generate exposure and save to file
----> 2 filename_exp = CONFIG.local_data.save_dir.dir() / ('exp_' +
      3                                           'litpop' +
      4                                           '_' +
      5                                           'Switzerland' +

NameError: name 'CONFIG' is not defined

```

[5]: *#create and calculate Forecast*

```

CH_WS_forecast = Forecast({run_datetime: hazard}, exposure, impact_function_set)
CH_WS_forecast.calc()

```

```

-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_3138/3531404486.py in <module>
      1 #create and calculate Forecast
----> 2 CH_WS_forecast = Forecast({run_datetime: hazard}, exposure, impact_function_set)
      3 CH_WS_forecast.calc()

NameError: name 'Forecast' is not defined

```

[6]: `CH_WS_forecast.plot_imp_map(save_fig=False,close_fig=False,proj=ccrs.epsg(2056))`

```

-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_3138/159247755.py in <module>
----> 1 CH_WS_forecast.plot_imp_map(save_fig=False,close_fig=False,proj=ccrs.epsg(2056))

NameError: name 'CH_WS_forecast' is not defined

```

Here you see a different plot highlighting the spread of the impact forecast calculated from the different ensemble members of the weather forecast.

[7]: `CH_WS_forecast.plot_hist(save_fig=False,close_fig=False)`

```

-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_3138/3197302225.py in <module>
----> 1 CH_WS_forecast.plot_hist(save_fig=False,close_fig=False)

NameError: name 'CH_WS_forecast' is not defined

```

It is possible to color the pixels depending on the probability that a certain threshold of impact is reach at a certain grid point

```
[8]: CH_WS_forecast.plot_exceedence_prob(threshold=5000, save_fig=False, close_fig=False,
    ↪proj=ccrs.epsg(2056))
```

```
-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_3138/431861101.py in <module>
----> 1 CH_WS_forecast.plot_exceedence_prob(threshold=5000, save_fig=False, close_
    ↪fig=False,proj=ccrs.epsg(2056))

NameError: name 'CH_WS_forecast' is not defined
```

It is possible to color the cantons of Switzerland with warning colors, based on aggregated forecasted impacts in their area.

```
[9]: import fiona
    from cartopy.io import shapereader
    from climada.util.config import CONFIG

    #create a file containing the polygons of Swiss cantons using natural earth
    cantons_file = CONFIG.local_data.save_dir.dir() / 'cantons.shp'
    adm1_shape_file = shapereader.natural_earth(resolution='10m',
                                                category='cultural',
                                                name='admin_1_states_provinces')

    if not cantons_file.exists():
        with fiona.open(adm1_shape_file, 'r') as source:
            with fiona.open(
                cantons_file, 'w',
                **source.meta) as sink:

                for f in source:
                    if f['properties']['adm0_a3'] == 'CHE':
                        sink.write(f)

    CH_WS_forecast.plot_warn_map(str(cantons_file),
                                decision_level = 'polygon',
                                thresholds=[1000000,5000000,
                                            10000000,50000000],
                                probability_aggregation='mean',
                                area_aggregation='sum',
                                title="Building damage warning",
                                explain_text="warn level based on aggregated damages",
                                save_fig=False,
                                close_fig=False,
                                proj=ccrs.epsg(2056))
```

```
-----
ModuleNotFoundError                      Traceback (most recent call last)
/tmp/ipykernel_3138/289389475.py in <module>
      1 import fiona
      2 from cartopy.io import shapereader
----> 3 from climada.util.config import CONFIG
      4
```

(continues on next page)

(continued from previous page)

5

`ModuleNotFoundError`: No module named 'climada'

5.13.2 Example 2: forecast of wind warnings in Switzerland

Instead of a fully fledged socio-economic impact of storms, one can also simplify the hazard, exposure, vulnerability model, by looking at a “neutral” exposure (=1 at every gridpoint) and using a step function as impact function to arrive at warn levels. It also shows how the attributes hazard, exposure or vulnerability can be set before calling `calc()`, and are then considered in the forecast instead of the defined defaults.

```
[10]: from pandas import DataFrame
import numpy as np
from climada.entity.exposures import Exposures
from climada.entity.impact_funcs import ImpactFunc, ImpactFuncSet
import climada.util.plot as u_plot

### generate exposure
# find out which hazard coord to consider
CHE_borders = u_plot._get_borders(np.stack([exposure.gdf.latitude.values,
                                             exposure.gdf.longitude.values],
                                             axis=1)
                                )
centroid_selection = np.logical_and(np.logical_and(hazard.centroids.lat >= CHE_
↳borders[2],
                                             hazard.centroids.lat <= CHE_
↳borders[3]),
                                np.logical_and(hazard.centroids.lon >= CHE_
↳borders[0],
                                             hazard.centroids.lon <= CHE_
↳borders[1])
                                )
# Fill DataFrame with values for a "neutral" exposure (value = 1)
exp_df = DataFrame()
exp_df['value'] = np.ones_like(hazard.centroids.lat[centroid_selection]) # provide value
exp_df['latitude'] = hazard.centroids.lat[centroid_selection]
exp_df['longitude'] = hazard.centroids.lon[centroid_selection]
exp_df['impf_WS'] = np.ones_like(hazard.centroids.lat[centroid_selection], int)
# Generate Exposures
exp = Exposures(exp_df)
exp.check()
exp.value_unit = 'warn_level'

### generate impact functions
## impact functions for hazard based warnings
imp_fun_low = ImpactFunc()
imp_fun_low.haz_type = 'WS'
imp_fun_low.id = 1
imp_fun_low.name = 'warn_level_low_elevation'
imp_fun_low.intensity_unit = 'm/s'
```

(continues on next page)

(continued from previous page)

```

imp_fun_low.intensity = np.array([0.0, 19.439,
                                19.44, 24.999,
                                25.0, 30.549,
                                30.55, 38.879,
                                38.88, 100.0])
imp_fun_low.mdd = np.array([1.0, 1.0,
                            2.0, 2.0,
                            3.0, 3.0,
                            4.0, 4.0,
                            5.0, 5.0])
imp_fun_low.paa = np.ones_like(imp_fun_low.mdd)
imp_fun_low.check()
# fill ImpactFuncSet
impf_set = ImpactFuncSet()
impf_set.append(imp_fun_low)

```

```

-----
ModuleNotFoundError                                Traceback (most recent call last)
/tmp/ipykernel_3138/1441365374.py in <module>
      1 from pandas import DataFrame
      2 import numpy as np
----> 3 from climada.entity.exposures import Exposures
      4 from climada.entity.impact_funcs import ImpactFunc, ImpactFuncSet
      5 import climada.util.plot as u_plot

ModuleNotFoundError: No module named 'climada'

```

[11]: *#create and calculate Forecast*

```

warn_forecast = Forecast({run_datetime: hazard}, exp, impf_set)
warn_forecast.calc()

```

```

-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_3138/1666587155.py in <module>
      1 #create and calculate Forecast
----> 2 warn_forecast = Forecast({run_datetime: hazard}, exp, impf_set)
      3 warn_forecast.calc()

NameError: name 'Forecast' is not defined

```

The each grid point now has a warnlevel between 1-5 assigned for each event. Now the cantons can be colored based on a threshold on a grid point level. for each warning level it is assessed if 50% of grid points in the area of a canton has at least a 50% probability of reaching the specified threshold.

```

[12]: warn_forecast.plot_warn_map(cantons_file,
                                thresholds=[2,3,4,5],
                                decision_level = 'exposure_point',
                                probability_aggregation=0.5,
                                area_aggregation=0.5,
                                title="DWD ICON METEOROLOGICAL WARNING",
                                explain_text="warn level based on wind gust thresholds",
                                save_fig=False,
                                close_fig=False,

```

(continues on next page)

(continued from previous page)

```

proj=ccrs.epsg(2056))

-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_3138/3218391721.py in <module>
----> 1 warn_forecast.plot_warn_map(cantons_file,
      2                               thresholds=[2,3,4,5],
      3                               decision_level = 'exposure_point',
      4                               probability_aggregation=0.5,
      5                               area_aggregation=0.5,

NameError: name 'warn_forecast' is not defined

```

5.13.3 Example: Tropical Cylcone

It would be nice to add an example using the tropical cyclone forecasts from the class TCForecast. This has not yet been done.

```
[ ]:
```

```
[ ]:
```

5.14 Calculate probabilistic impact yearset

This module generates a yearly impact `yimp` object which contains probabilistic annual impacts for a specified amount of years (`sampled_years`). The impact values are extracted from a given impact `imp` object that contains impact values per event. The amount of `sampled_years` can be specified as an integer or as a list of years to be sampled for. The amount of events per sampled year (`events_per_year`) are determined with a Poisson distribution centered around `n_events` per year (`lam = sum(event_impacts.frequency)`). Then, the probabilistic events occurring in each sampled year are sampled uniformly from the input `imp` object and summed up per year. Thus, the `yimp` object contains the sum of sampled (event) impacts for each sampled year. In contrast to the expected annual impact (`eai`), an `yimp` object contains an impact for EACH sampled year and this value differs among years. The number of `events_per_year` and the selected `events` are saved in a sampling vector (`sampling_vect`).

The function `impact_yearsets` performs all these computational steps, taking an `imp` and the number of `sampled_years` (`sampled_years`) as input. The output of the function is the `yimp` object and the `sampling_vect`. Moreover, a `sampling_vect` (generated in a previous run) can be provided as optional input and the user can define `lam` and decide whether a correction factor shall be applied (the default is applying the correction factor). Reapplying the same `sampling_vect` does not only allow to reproduce the generated `yimp`, but also for a physically consistent way of sampling impacts caused by different hazards. The correction factor that is applied when the optional input `correction_fac= True` is a scaling of the computed `yimp` that assures that the `eai(yimp) = eai(imp)`.

To make the process more transparent, this tutorial shows the single computations that are performed when generating an `yimp` object for a dummy `event_impacts` object.

```

[5]: import numpy as np

import climada.util.yearsets as yearsets
from climada.engine import Impact

```

(continues on next page)

(continued from previous page)

```

# dummy event_impacts object containing 10 event_impacts with the values 10-110
# and the frequency 0.2 (Return period of 5 years)
imp = Impact()
imp.at_event = np.arange(10,110,10)
imp.frequency = np.array(np.ones(10)*0.2)

# the number of years to sample impacts for (length(yimp.at_event) = sampled_years)
sampled_years = 10

# sample number of events per sampled year
lam = np.sum(imp.frequency)
events_per_year = yearsets.sample_from_poisson(sampled_years, lam)

# generate the sampling vector
sampling_vect = yearsets.sample_events(events_per_year, imp.frequency)

# calculate the impact per year
imp_per_year = yearsets.compute_imp_per_year(imp, sampling_vect)

# calculate the correction factor
correction_factor = yearsets.calculate_correction_fac(imp_per_year, imp)

# compare the resulting yimp with our step-by-step computation without applying the
↳ correction factor:

yimp, sampling_vect = yearsets.impact_yearset(imp, sampling_vect=sampling_vect,
                                              correction_fac = False)

print('The yimp.at_event values equal our step-by-step computed imp_per_year:')
print('yimp.at_event = ', yimp.at_event)
print('imp_per_year = ', imp_per_year)

# and here the same comparison with applying the correction factor (default settings):
yimp, sampling_vect = yearsets.impact_yearset(imp, sampling_vect=sampling_vect)

print('The same can be shown for the case of applying the correction factor.')
print('The yimp.at_event values equal our step-by-step computed imp_per year:')
print('yimp.at_event = ', yimp.at_event)
print('imp_per_year = ', imp_per_year/correction_factor)

```

```

2021-06-02 16:21:06,423 - climada.util.yearsets - INFO - The correction factor amounts
↳ to -14.72868217054264
The yimp.at_event values equal our step-by-step computed imp_per_year:
yimp.at_event = [ 60.  70.  90. 260.   0. 270. 140.   0. 210. 190.]
imp_per_year = [ 60.  70.  90. 260.   0. 270. 140.   0. 210. 190.]
2021-06-02 16:21:06,431 - climada.util.yearsets - INFO - The correction factor amounts
↳ to -14.72868217054264

```

(continues on next page)

(continued from previous page)

The same can be shown for the case of applying the correction factor. The `yimp.at_event` values equal our step-by-step computed `imp_per_year`:

```
yimp.at_event = [ 70.36363636 82.09090909 105.54545455 304.90909091 0.
                 316.63636364 164.18181818 0.                246.27272727 222.81818182]
imp_per_year = [ 70.36363636 82.09090909 105.54545455 304.90909091 0.
                 316.63636364 164.18181818 0.                246.27272727 222.81818182]
```

[]:

DEVELOPER GUIDE

6.1 Development and Git and CLIMADA

Chris Fairless

Table of Contents

1 development and Git and CLIMADA

1.1 Introduction

1.2 Git and GitHub

1.3 Gitflow

1.4 Installing CLIMADA for development

1.5 Features and branches

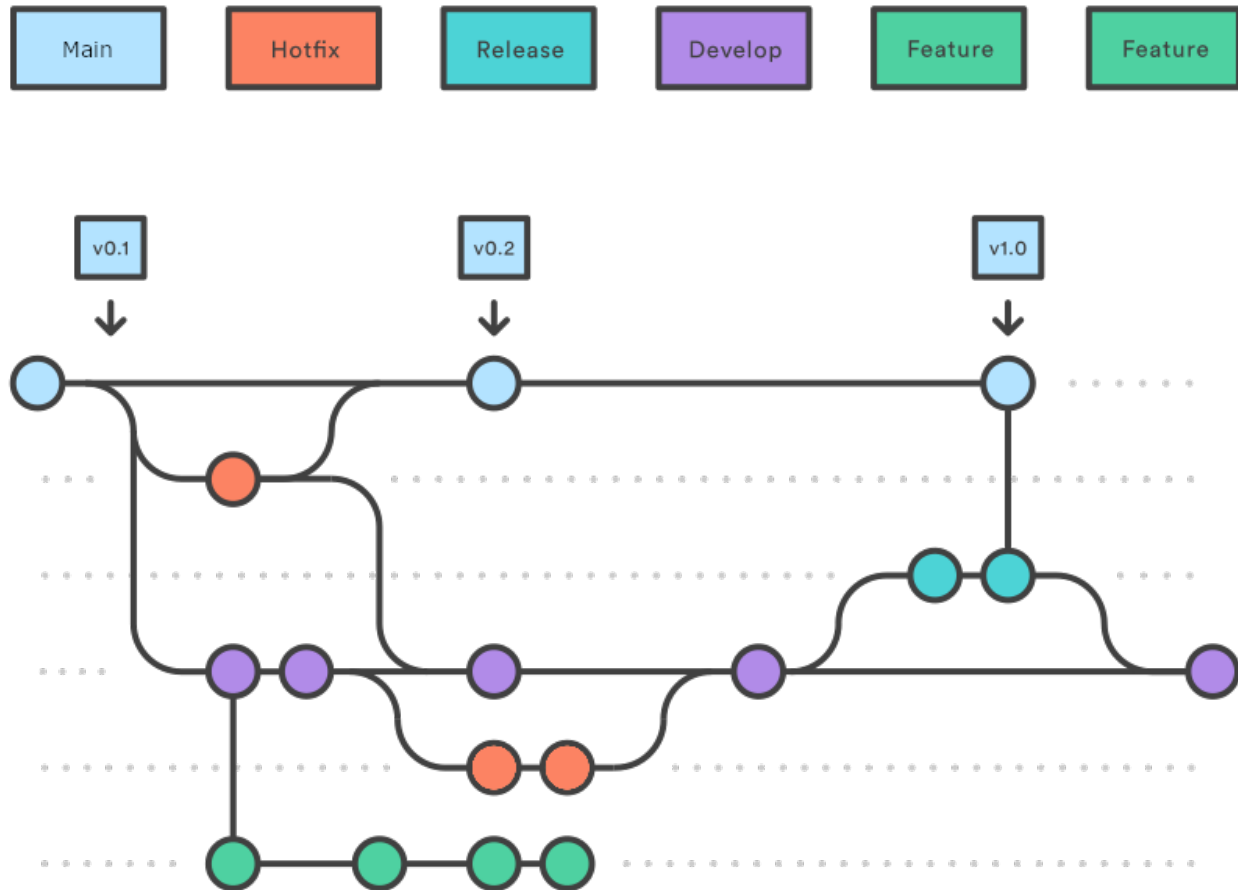
1.6 Pull Requests

1.7 General tips and tricks

Introduction

6.1.1 Git and GitHub

- Git's not that scary
 - 95% of your work on Git will be done with the same handful of commands
 - (the other 5% will always be done with careful Googling)
 - Almost everything in Git can be undone by design (but use `rebase`, `--force` and `--hard` with care!)
 - Your favourite IDE (Spyder, PyCharm, ...) will have a GUI for working with Git, or you can download a standalone one.
- The [Git Book](#) is a great introduction to how Git works and to using it on the command line.
- Consider using a GUI program such as “git desktop” or “Gitkraken” to have a visual git interface, in particular at the beginning. Your python IDE is also likely to have a visual git interface.
- Feel free to ask for help



What I assume you know

I'm assuming you're all familiar with the basics of Git.

- What (and why) is version control
- How to clone a repository
- How to make a commit and push it to GitHub
- What a branch is, and how to make one
- How to merge two branches
- The basics of the GitHub website

If you're not feeling great about this, I recommend - sending me a message so we can arrange an introduction with CLIMADA - exploring the [Git Book](#)

Terms we'll be using today

These are terms I'll be using a lot today, so let's make sure we know them

- local versus remote
 - Our **remote** repository is hosted on GitHub. This is the central location where all updates to CLIMADA that we want to share end up. If you're updating CLIMADA for the community, your code will end up here too.
 - Your **local** repository is the copy you have on the machine you're working on, and where you do your work.
 - Git calls the (first, default) remote the **origin**
 - (It's possible to set more than one remote repository, e.g. you might set one up on a network-restricted computing cluster)
- push, pull and pull request
 - You **push** your work when you send it from your local machine to the remote repository
 - You **pull** from the remote repository to update the code on your local machine
 - A **pull request** is a standardised review process on GitHub. Usually it ends with one branch merging into another
- Conflict resolution
 - Sometimes two people have made changes to the same bit of code. Usually this comes up when you're trying to merge branches. The changes have to be manually compared and the code edited to make sure the 'correct' version of the code is kept.

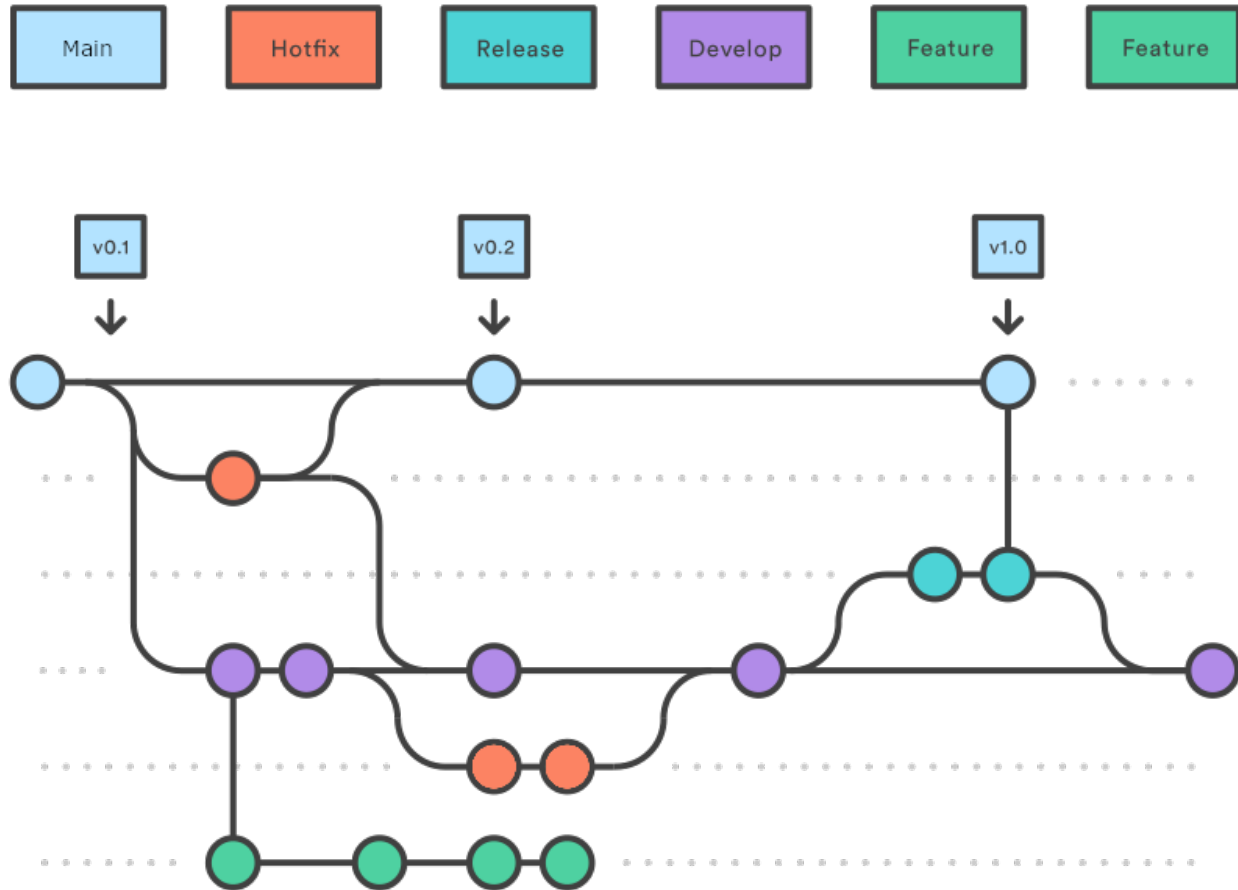
Gitflow

Gitflow is a particular way of using git to organise projects that have - multiple developers - working on different features - with a release cycle

It means that - there's always a stable version of the code available to the public - the chances of two developers' code conflicting are reduced - the process of adding and reviewing features and fixes is more standardised for everyone

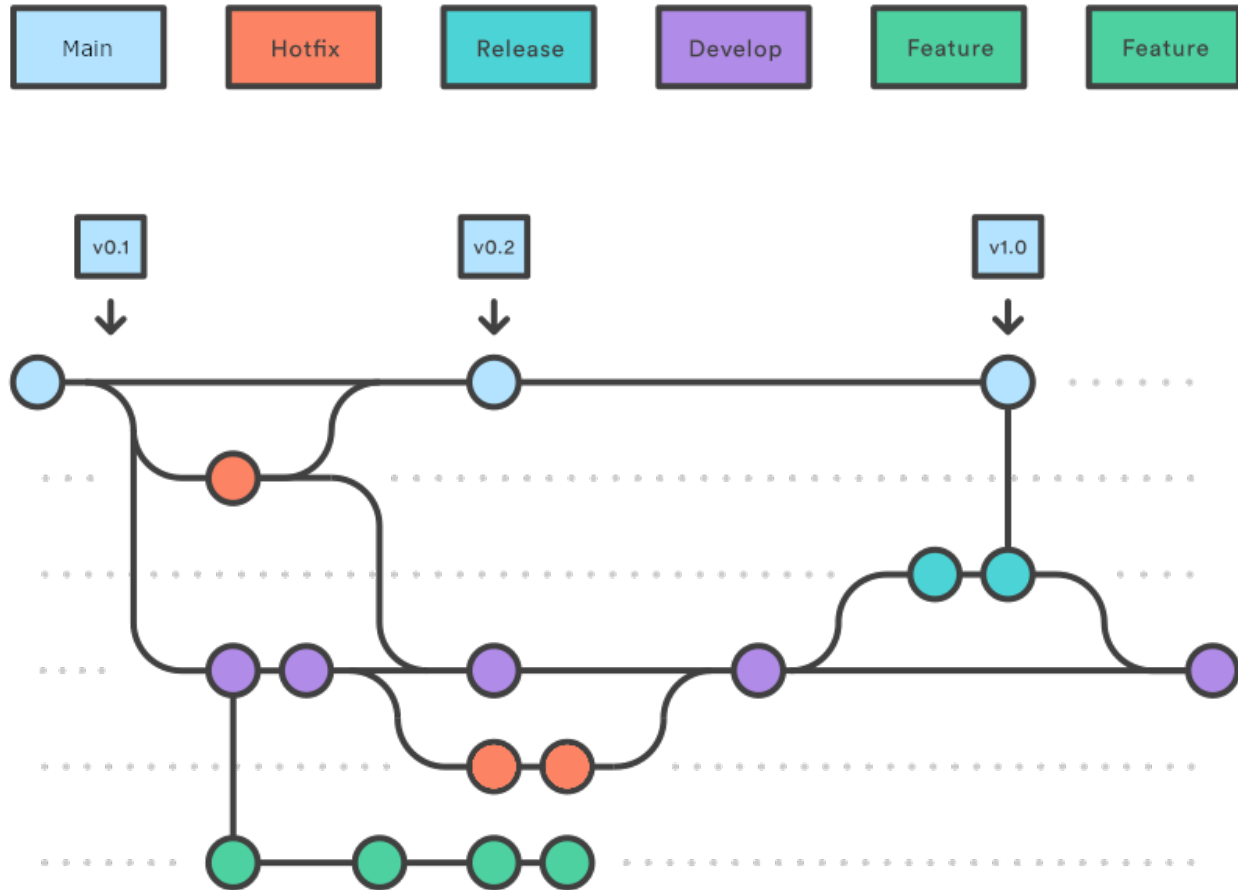
Gitflow is a *convention*, so you don't need any additional software. - ... but if you want you can get some: a popular extension to the git command line tool allows you to issue more intuitive commands for a Gitflow workflow. - Mac/Linux users can install git-flow from their package manager, and it's included with Git for Windows

Gitflow works on the `develop` branch instead of `main`



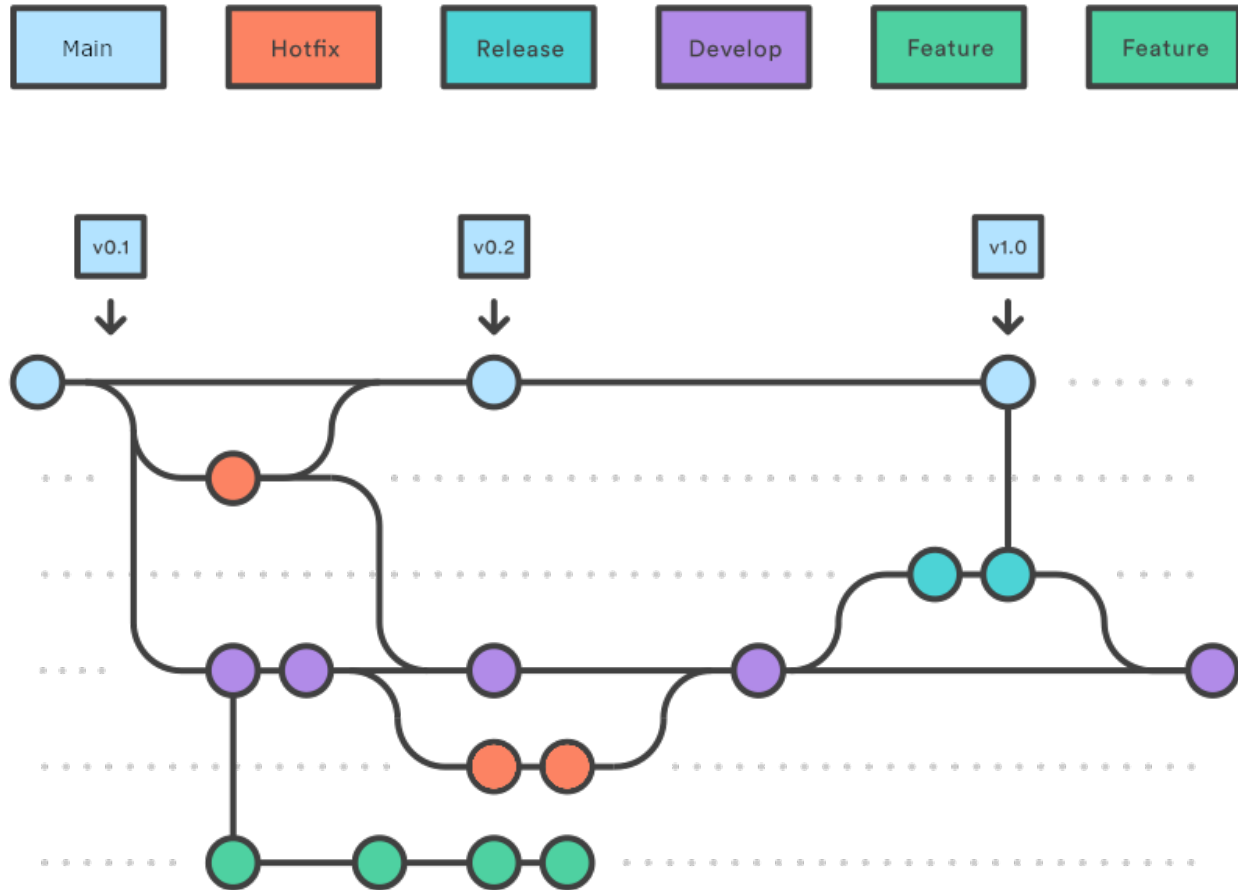
- The critical difference between Gitflow and 'standard' git is that almost all of your work takes place on the `develop` branch, instead of the `main` (formerly `master`) branch.
- The `main` branch is reserved for planned, stable product releases, and it's what the general public download when they install CLIMADA. The developers almost never interact with it.

Gitflow is a feature-based workflow



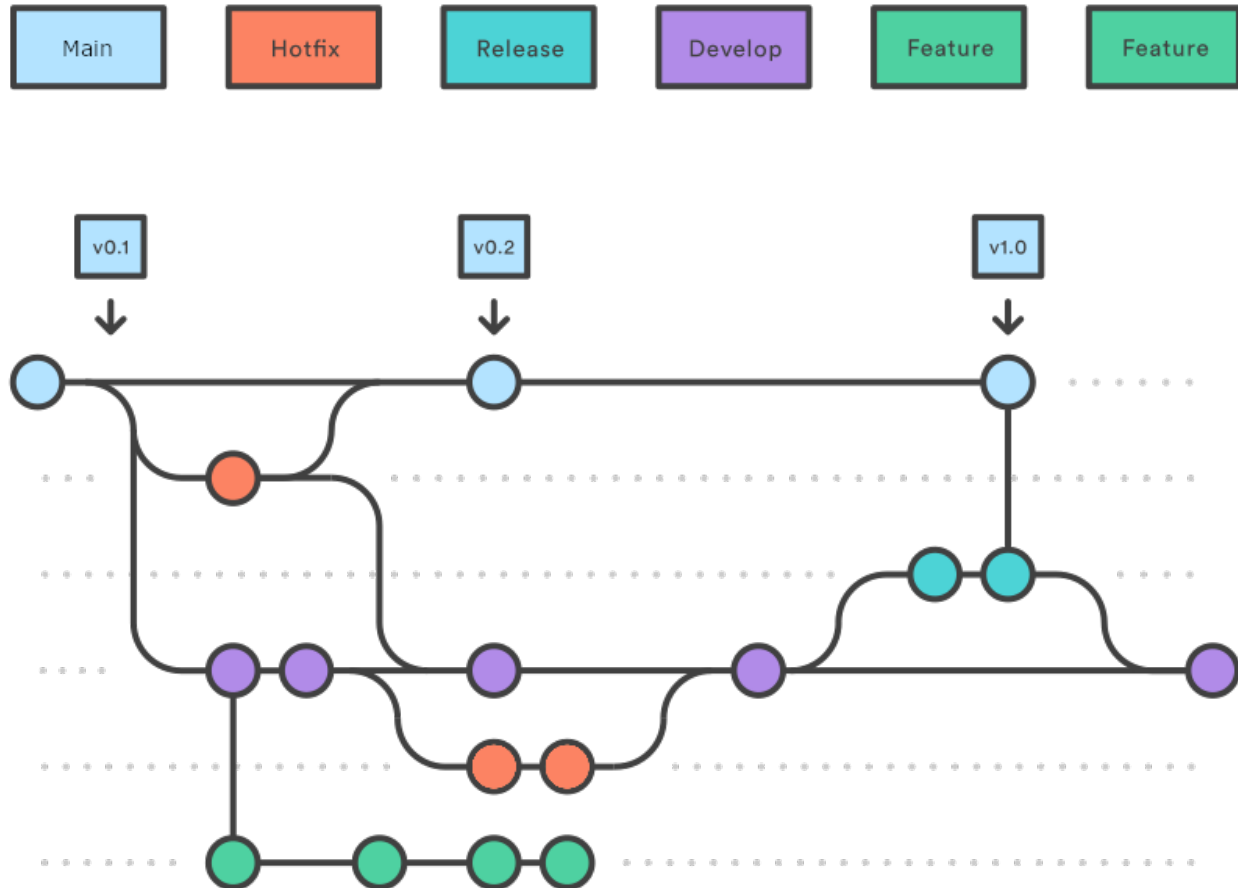
- This is common to many workflows: when you want to add something new to the model you start a new branch, work on it locally, and then merge it back into **develop with a pull request** (which we'll cover later).
- By convention we name all CLIMADA feature branches `feature/*` (e.g. `feature/meteorite`).
- Features can be anything, from entire hazard modules to a smarter way to do one line of a calculation. Most of the work you'll do on CLIMADA will be a features of one size or another.
- We'll talk more about developing CLIMADA features later!

Gitflow enables a regular release cycle



- A release is usually more complex than merging develop into main.
- So for this a `release-*` branch is created from develop. We'll all be notified repeatedly when the deadline is to submit (and then to review) pull requests so that you can be included in a release.
- The core developer team (mostly Emanuel) will then make sure tests, bugfixes, documentation and compatibility requirements are met, merging any fixes back into develop.
- On release day, the release branch is merged into main, the commit is tagged as a release and the release notes are published on the GitHub at https://github.com/CLIMADA-project/clinada_python/releases

Everything else is hotfixes



- The other type of branch you'll create is a hotfix.
- Hotfixes are generally small changes to code that do one thing, fixing typos, small bugs, or updating docstrings. They're done in much the same way as features, and are usually merged with a pull request.
- The difference between features and hotfixes is fuzzy and you don't need to worry about getting it right.
- Hotfixes will occasionally be used to fix bugs on the main branch, in which case they will merge into both main and develop.
- Some hotfixes are so simple - e.g. fixing a typo or a docstring - that they don't need a pull request. Use your judgement, but as a rule, if you change what the code does, or how, you should be merging with a pull request.

6.1.2 Installing CLIMADA for development

0. **Install** [Git](#) and [Anaconda](#) (or [Miniconda](#)).

Also consider installing Git flow. This is included with [Git for Windows](#) and has different implementations e.g. [here](#) for Windows and Mac.

1. **Clone (or fork)** the project on GitHub

From the location where you want to create the project folder, run in your terminal:

```
::
```

git clone https://github.com/CLIMADA-project/limada_python.git

2. **Install the packages** in `limada_python/requirements/env_limada.yml` and `limada_python/requirements/env_developer.yml` (see [install](#)). You might need to install additional environments contained in `limada_python/requirements` when using specific functionalities.

6.1.3 Features and branches

Planning a new feature

Here we're talking about large features such as new modules, new data sources, or big methodological changes. Any extension to CLIMADA that might affect other developers' work, modify the CLIMADA core, or need a big code review.

Smaller feature branches don't need such formalities. Use your judgment, and if in doubt, let people know.

Talk to the group

- Before starting coding a module, do not forget to coordinate with one of the repo admins (Emanuel, Chahan or David)
- This is the chance to work out the Big Picture stuff that is better when it's planned with the group - possible intersections with other projects, possible conflicts, changes to the CLIMADA core, additional dependencies (see Chahan's presentation later)
- Also talk with others from the core development team (see [the GitHub wiki](#)).
- Bring it to a developers meeting - people may be able to help/advice and are always interested in hearing about new projects. You can also find reviewers!
- Also, keep talking! Your plans *will* change :)

Planning the work

- Does the project go in its own repository and import CLIMADA, or does it extend the main CLIMADA repository?
 - The way this is done is slowly changing, so definitely discuss it with the group.
 - Chahan will discuss this later!
- Find a few people who will help to review your code.
 - Ask in a developers' meeting, on Slack (for WCR developers) or message people on the development team (see [the GitHub wiki](#)).
 - Let them know roughly how much code will be in the reviews, and when you'll be creating pull requests.
- How can the work split into manageable chunks?
 - A series of smaller pull requests is far more manageable than one big one (and takes off some of the pre-release pressure)
 - Reviewing and spotting issues/improvements/generalisations early is always a good thing.
 - It encourages modularisation of the code: smaller self-contained updates, with documentation and tests.
- Will there be any changes to the CLIMADA core?
 - These should be planned carefully

- Will you need any new dependencies? Are you sure?
 - Chahan will discuss this later!

Working on feature branches

When developing a big new feature, consider creating a feature branch and merging smaller branches into that feature branch with pull requests, keeping the whole process separate from `develop` until it's completed. This makes step-by-step code review nice and easy, and makes the final merge more easily tracked in the history.

e.g. developing the big `feature/meteorite` module you might write `feature/meteorite-hazard` and merge it in, then `feature/meteorite-impact`, then `feature/meteorite-stochastic-events` etc... before finally merging `feature/meteorite` into `develop`. Each of these could be a reviewable pull request.

Make a new branch

For new features in Git flow:

```
git flow feature start feature_name
```

Which is equivalent to (in vanilla git):

```
git checkout -b feature/feature_name
```

Or work on an existing branch:

```
git checkout -b branch_name
```

Follow the python do's and don't and performance guides. Write small readable methods, classes and functions.

get the latest data from the remote repository and update your branch

```
git pull
```

see your locally modified files

```
git status
```

add changes you want to include in the commit

```
git add climada/modified_file.py climada/test/test_modified_file.py
```

commit the changes

```
git commit -m "new functionality of .. implemented"
```

Make unit and integration tests on your code, preferably during development

see [Guide on unit and integration tests](#)

6.1.4 Pull requests

We want every line of code that goes into the CLIMADA repository to be reviewed!

Code review: - catches bugs (there are *always* bugs) - lets you draw on the experience of the rest of the team - makes sure that more than one person knows how your code works - helps to unify and standardise CLIMADA's code, so new users find it easier to read and navigate - creates an archived description and discussion of the changes you've made

When to make a pull request

- When you've finished writing a big new class or method (and its tests)
- When you've fixed a bug or made an improvement you want to merge
- When you want to merge a change of code into `develop` or `main`
- When you want to *discuss* a bit of code you've been working on - pull requests aren't only for merging branches

Not all pull requests have to be into `develop` - you can make a pull request into any active branch that suits you.

Pull requests need to be made latest two weeks before a release, see [releases](#).

Step by step pull request!

Let's suppose you've developed a cool new module on the `feature/meteorite` branch and you're ready to merge it into `develop`.

Checklist before you start

- Documentation
- Tests
- Tutorial (if a complete new feature)
- Updated dependencies (if need be)
- Added your name to the AUTHORS file
- (Advanced, optional) interactively rebase/squash recent commits that *aren't yet on GitHub*.

Step by step pull request!

- 1) Make sure the `develop` branch is up to date on your own machine

```
git checkout develop
git pull
```

- 2) Merge `develop` into your feature branch and resolve any conflicts

```
git checkout feature/meteorite
git merge develop
```

In the case of more complex conflicts, you may want to speak with others who worked on the same code. Your IDE should have a tool for conflict resolution.

- 3) Check all the tests pass locally

```
make unit_test
make integ_test
```

- 4) Perform a static code analysis using pylint with CLIMADA's configuration `.pylintrc` (in the climada root directory). Jenkins executes it after every push. To do it locally, your IDE probably provides a tool, or you can run `make lint` and see the output in `pylint.log`.

- 5) Push to GitHub. If you're pushing this branch for the first time, use

```
git push -u origin feature/meteorite
```

and if you're updating a branch that's already on GitHub:

```
git push
```

- 6) Check all the tests pass on the WCR Jenkins server (<https://ied-wcr-jenkins.ethz.ch>). See Emanuel's presentation for how to do this! You should regularly be pushing your code and checking this!

- 7) Create the pull request!

- On the CLIMADA GitHub page, navigate to your feature branch (there's a drop-down menu above the file structure, pointing by default to `main`).
- Above the file structure is a branch summary and an icon to the right labelled "Pull request".
- Choose which branch you want to merge with. This will usually be `develop`, but may be another feature branch for more complex feature development.
- Give your pull request an informative title (like a commit message).
- Write a description of the pull request. This can usually be adapted from your branch's commit messages (you wrote informative commit messages, didn't you?), and should give a high-level summary of the changes, specific points you want the reviewers' input on, and explanations for decisions you've made. The code documentation (and any references) should cover the more detailed stuff.
- Assign reviewers in the page's right hand sidebar. Tag anyone who might be interested in reading the code. You should already have found one or two people who are happy to read the whole request and sign it off (they could also be added to 'Assignees').
- Create the pull request.
- Contact the reviewers to let them know the request is live. GitHub's settings mean that they may not be alerted automatically. Maybe also let people know on the WCR Slack!

- 8) Talk with your reviewers

- Use the comment/chat functionality within GitHub's pull requests - it's useful to have an archive of discussions and the decisions made.
- Take comments and suggestions on board, but you don't need to agree with everything and you don't need to implement everything.
- If you feel someone is asking for too many changes, prioritise, especially if you don't have time for complex rewrites.
- If the suggested changes and or features don't block functionality and you don't have time to fix them, they can be moved to Issues.

- Chase people up if they're slow. People are slow.
- 9) Once you implement the requested changes, respond to the comments with the corresponding commit implementing each requested change.
- 10) If the review takes a while, remember to merge `develop` back into the feature branch every now and again (and check the tests are still passing on Jenkins). Anything pushed to the branch is added to the pull request.
- 11) Once everyone reviewing has said they're satisfied with the code you can merge the pull request using the GitHub interface. Delete the branch once it's merged, there's no reason to keep it. (Also try not to re-use that branch name later.)
- 12) Update the `develop` branch on your local machine.

How to review a pull request

- Be friendly
- Decide how much time you can spare and the detail you can work in. Tell the author!
- Use the comment/chat functionality within GitHub's pull requests - it's useful to have an archive of discussions and the decisions made.
- Fix the big things first! If there are more important issues, not every style guide has to be stuck to, not every slight increase in speed needs to be pointed out, and test coverage doesn't have to be 100%.
- Make it clear when a change is optional, or is a matter of opinion

At a minimum - Make sure unit and integration tests are passing on Jenkins - (For complete modules) Run the tutorial on your local machine and check it does what it says it does - Check everything is fully documented

At least one reviewer needs to - Review all the changes in the pull request. Read what it's supposed to do, check it does that, and make sure the logic is sound. - Check that the code follows the CLIMADA style guidelines [#TODO: link](#) - If the code is implementing an algorithm it should be referenced in the documentation. Check it's implemented correctly. - Try to think of edge cases and ways the code could break. See if there's appropriate error handling in cases where the function might behave unexpectedly. - (Optional) suggest easy ways to speed up the code, and more elegant ways to achieve the same goal.

There are a few ways to suggest changes - As questions and comments on the pull request page - As code suggestions (max a few lines) in the code review tools on GitHub. The author can then approve and commit the changes from GitHub pull request page. This is great for typos and little stylistic changes. - If you decide to help the author with changes, you can either push them to the same branch, or create a new branch and make a pull request with the changes back into the branch you're reviewing. This lets the author review it and merge.

6.1.5 General tips and tricks

6.1.6 Ask for help with Git

- Git isn't intuitive, and rewinding or resetting is always work. If you're not certain what you're doing, or if you think you've messed up, send someone a message.

Don't push or commit to develop or main

- Almost all new additions to CLIMADA should be merged into the `develop` branch with a pull request.
- You won't merge into the `main` branch, except for emergency hotfixes (which should be communicated to the team).
- You won't merge into the `develop` branch without a pull request, except for small documentation updates and typos.
- The above points mean you should never need to push the `main` or `develop` branches.

So if you find yourself on the `main` or `develop` branches typing `git merge ...` or `git push` stop and think again - you should probably be making a pull request.

This can be difficult to undo, so contact someone on the team if you're unsure!

Commit more often than you think, and use informative commit messages

- Committing often makes mistakes less scary to undo

```
git reset --hard HEAD
```

- Detailed commit messages make writing pull requests really easy
- Yes it's boring, but *trust me*, everyone (usually your future self) will love you when they're rooting through the git history to try and understand why something was changed

Commit message syntax guidelines

Basic syntax guidelines taken from here <https://chris.beams.io/posts/git-commit/> (on 17.06.2020)

- Limit the subject line to 50 characters
- Capitalize the subject line
- Do not end the subject line with a period
- Use the imperative mood in the subject line (e.g. "Add new tests")
- Wrap the body at 72 characters (most editors will do this automatically)
- Use the body to explain what and why vs. how
- Separate the subject from body with a blank line (This is best done with a GUI. With the command line you have to use text editor, you cannot do it directly with the git command)
- Put the name of the function/class/module/file that was edited
- When fixing an issue, add the reference `gh-ISSUENUMBER` to the commit message e.g. "fixes gh-40." or "Closes gh-40." For more infos see here <https://docs.github.com/en/enterprise/2.16/user/github/managing-your-work-on-github/closing-issues-using-keywords#about-issue-references>.

What not to commit

There are a lot of things that don't belong in the Git repository: - Don't commit data, except for config files and very small files for tests. - Don't commit anything containing passwords or authentication credentials or tokens. (These are annoying to remove from the Git history.) Contact the team if you need to manage authorisations within the code. - Don't commit anything that can be created by the CLIMADA code itself

If files like this are going to be present for other users as well, add them to the repository's `.gitignore`.

Log ideas and bugs as GitHub Issues

If there's a change you might want to see in the code - something that generalises, something that's not quite right, or a cool new feature - it can be set up as a GitHub Issue. Issues are pages for conversations about changes to the codebase and for logging bugs, and act as a 'backlog' for the CLIMADA project.

For a bug, or a question about functionality, make a minimal working example, state which version of CLIMADA you are using, and post it with the Issue.

How not to mess up the timeline

Git builds the repository through incremental edits. This means it's great at keeping track of its history. But there are a few commands that *edit* this history, and if histories get out of sync on different copies of the repository you're going to have a bad time.

- Don't rebase any commits that already exist remotely!
- Don't `--force` anything that exists remotely unless you know what you're doing!
- Otherwise, you're unlikely to do anything irreversible
- You can do what you like with commits that only exist on your machine.

That said, doing an interactive rebase to tidy up your commit history *before* you push it to GitHub is a nice friendly gesture :)

Don't fast forward merges

(This shouldn't be relevant - all your merges into `develop` should be through pull requests, which doesn't fast forward. But:)

Don't fast forward your merges unless your branch is a single commit. Use `git merge --no-ff ...`

The exceptions is when you're merging `develop` into your feature branch.

Merge the remote `develop` branch into your feature branch every now and again

- This way you'll find conflicts early

```
git checkout develop
git pull
git checkout feature/myfeature
git merge develop
```

Create frequent pull requests

I said this already: - It structures your workflow - It's easier for reviewers - If you're going to break something for other people you all know sooner - It saves work for the rest of the team right before a release

Whenever you do something with CLIMADA, make a new local branch

You never know when a quick experiment will become something you want to save for later.

But don't do everything in the CLIMADA repository

- If you're running CLIMADA rather than developing it, create a new folder, initialise a new repository with `git init` and store your scripts and data there
- If you're writing an extension to CLIMADA that doesn't change the model core, create a new folder, initialise a new repository with `git init` and import CLIMADA. You can always add it to the model later if you need to.

Questions



<https://xkcd.com/1597/>

6.2 CLIMADA Tutorial Template

6.2.1 Content

1. *Why tutorials*
2. *Basic structure*
3. *Good examples*

1. Why tutorials

Main goal: The main goal of the tutorials is it to give a complete overview on: * essential CLIMADA components * introduce newly developed modules and features

More specifically, tutorials should introduce CLIMADA users to the core functionalities and modules and guide users in their application. Hence, each new module created needs to be accompanied with a tutorial. The following sections give an overview of the basic structure desired for CLIMADA tutorials.

Important: A tutorial needs to be included with the final pull request for every new feature.

2. Basic structure

Every tutorial should cover the following main points. Additional features characteristic to the modules presented can and should be added as see fit.

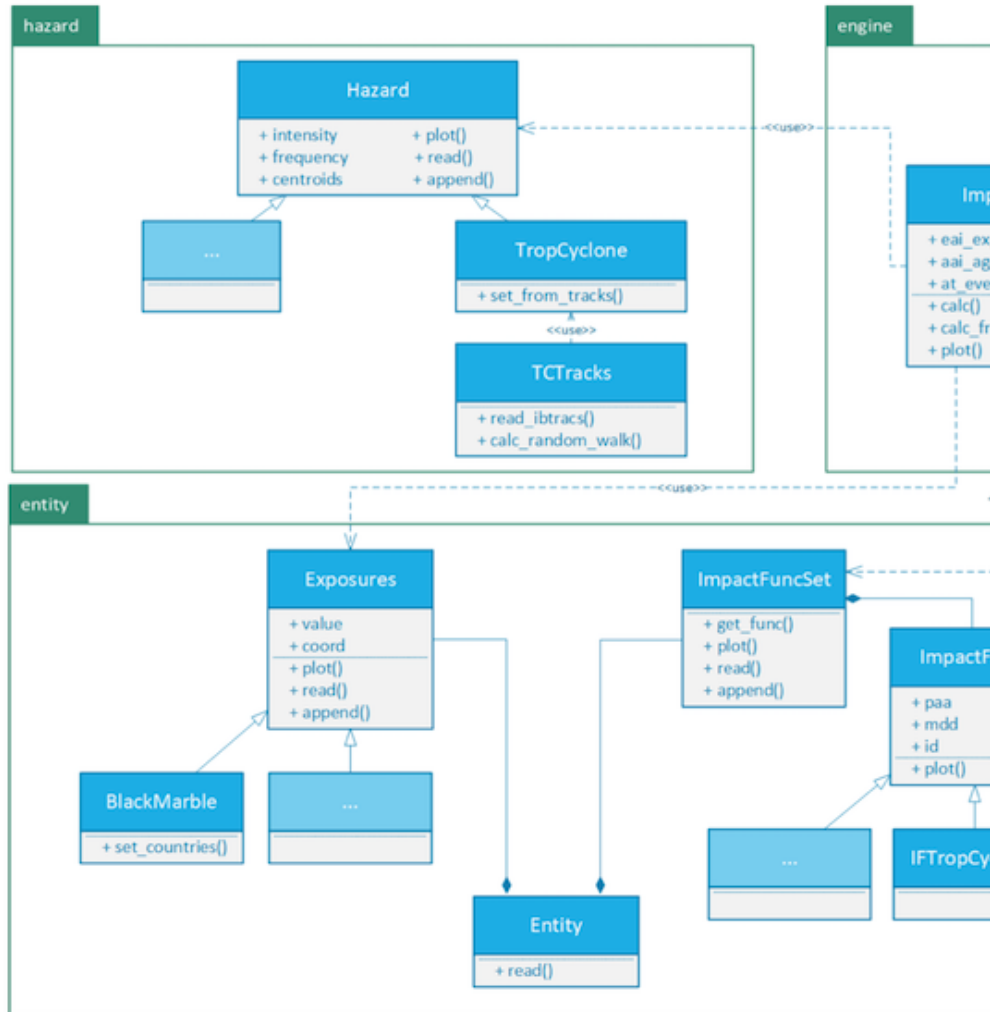
6.2.2 Introduction

- What is the feature presented? Briefly describe the feature and introduce how it's presented in the CLIMADA framework.
- What is its data structure? Present and overview (in the form of a table for example) of where the feature is built into CLIMADA. What class does it belong to, what are the variables of the feature, what is their data structure.
- Table of content: How is this tutorial structured?

6.2.3 Illustration of feature functionality and application

Walk users through the core functions of the module and illustrate how the feature can be used. This obviously is dependent on the feature itself. A few core points should be considered when creating the tutorial: * **SIZE MATTERS!** * each notebook as a total should not exceed the critical (yet vague) size of “a couple MB” * keep the size of data you use as examples in the tutorial in mind * we aim for computational efficiency * a lean, well-organized, concise notebook is more informative than a long, messy all-encompassing one.

- follow the general CLIMADA naming convention for the notebook. For example: “cli-



mada_hazard_TropCyclone.ipynb”

3. Good examples

The following examples can be used as templates and inspiration for your tutorial: * https://github.com/CLIMADA-project/climada_python/blob/tutorial_update/doc/tutorial/climada_entity_Exposures.ipynb * https://github.com/CLIMADA-project/climada_python/blob/tutorial_update/doc/tutorial/climada_hazard_Hazard.ipynb

6.3 Constants and Configuration

6.3.1 Content

1. *Constants*
 1. *Hard Coded*
 2. *Configurable*
 3. *Where to put constants?*
2. *Configuration*

1. *Config files*
2. *Accessing configuration values*
3. *Default Configuration*
4. *Test Configuration*

1. Constants

Constants are values that, once initialized, are never changed during the runtime of a program. In Python constants are assigned to variables with capital letters by convention, and vice versa, variables with capital letters are supposed to be constants.

In principle there are about four ways to define a constant's value: - *hard coding*: the value is defined in the python code directly - *argument*: the value is taken from an execution argument - *context*: the value is derived from the environmental context of the execution, e.g., the current working directory or the date-time of execution start. - *configuration*: read from a file or database

In CLIMADA, we only use *hard coding* and *configuration* to assign values to constants.

6.3.2 1.A. Hard Coded

Hard coding constants is the preferred way to deal with strings that are used to identify objects or files.

```
[22]: # suboptimal
my_dict = {'x': 4}
if my_dict['x'] > 3:
    msg = 'well, arh, ...'
msg
```

```
[22]: 'well, arh, ...'
```

```
[21]: # good
X = 'x'
my_dict = {X: 4}
if my_dict[X] > 3:
    msg = 'yeah!'
msg
```

```
[21]: 'yeah!'
```

```
[28]: # possibly overdoing it
X = 'x'
Y = "this doesn't mean that every string must be a constant"
my_dict = {X: 4}
if my_dict[X] > 3:
    msg = Y
msg
```

```
[28]: "this doesn't mean that every string must be a constant"
```

```
[26]: import pandas as pd
X = 'x'
df = pd.DataFrame({'x':[1,2,3], 'y':[4,5,6]})
```

(continues on next page)

(continued from previous page)

```

try:
    df.X
except:
    from sys import stderr; stderr.write("this does not work\n")
df[X] # this does work but it's less pretty
df.x

```

this does not work

```

[26]: 0    1
      1    2
      2    3
      Name: x, dtype: int64

```

6.3.3 1.B. Configurable

When it comes to absolute pathes, it is urgently suggested to not use hard coded constant values, for the obvious reasons. But also relative pathes can cause problems. In particular, they may point to a location where the user has not sufficient access permissions. In order to avoid these problems, *all* pathes constants in CLIMADA are supposed to be defined through configuration.

→ pathes must be configurable

The same applies to urls to external resources, databases or websites. Since they may change at any time, there addresses are supposed to be defined through configuration. Like this it will be possible to access them without the need of tampering with the source code or waiting for a new release.

→ urls must be configurable

Another category of constants that should go into the configuration file are system specifications, such as number of CPU's available for CLIMADA or memory settings.

→ OS settings must be configurable

6.3.4 1.C. Where to put constants?

As a general rule, constants are defined in the module where they intrinsically belong to. If they belong equally to different modules though or they are meant to be used globally, there is the module `climada.util.constants` which is compiling constants CLIMADA wide.

2. Configuration

6.3.5 2.A. Configuration files

The proper place to define constants that a user may want (or need) to change without changing the CLIMADA installation are the configuration files.

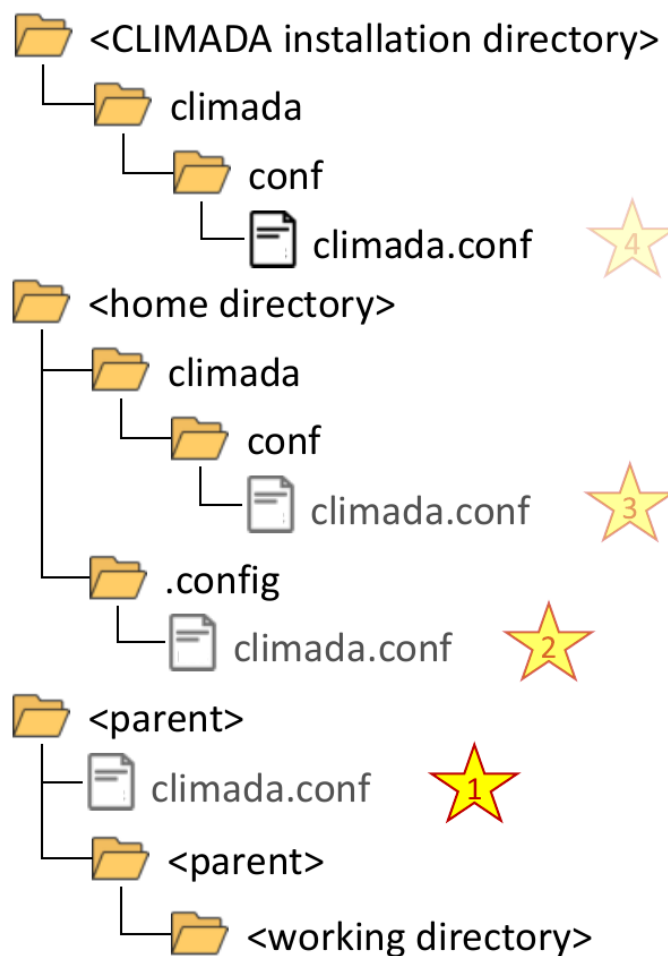
These are files in *json* format with the name `climada.conf`. There is a default config file that comes with the installation of CLIMADA. But it's possible to have several of them. In this case they are complementing one another.

CLIMADA looks for configuration files upon `import climada`. There are four locations to look for configuration files: - `climada/conf`, the installation directory - `~/climada/conf`, the user's default climada directory - `~/ .config`, the user's configuration directory, - `.`, the current working directory

At each location, the path is followed upwards until a file called `climada.conf` is found or the root of the path is reached. Hence, if e.g., `~/climada/climada.conf` is missing but `~/climada.conf` is present, the latter would be read.

When two config files are defining the same value, the priorities are:

```
[././climada.conf > ~/.config/climada.conf > ~/climada/conf/climada.conf >
installation_dir/climada/conf/climada.conf
```



A configuration file is a JSON file, with the additional restriction, that all keys must be strings without a `'` (dot) character .

For configuration values that belong to a particular module it is suggested to reflect the code repositories file structure in the json object. For example if a configuration for `my_config_value` that belongs to the module `climada.util.dates_times` is wanted, it would be defined as

```
{
  "util": {
    "dates_times": {
      "my_config_value": 42
    }
  }
}
```

Configuration string values can be referenced from other configuration values. E.g.

```
{
  "a": "x",
  "b": "{a}y"
}
```

In this example “b” is eventually resolved to “xy”.

6.3.6 2.B. Accessing configuration values

Configuration values can be accessed through the (constant) `CONFIG` from the `climada` module:

```
[1]: from climada import CONFIG
```

```
[3]: CONFIG.hazard
```

```
[3]: {drought: {resources: {spei_file_url: http://digital.csic.es/bitstream/10261/153475/8}},
→ landslide: {resources: {opensearch: https://pmmpublisher.pps.eosdis.nasa.gov/
→ opensearch, climatology_monthly: https://svs.gsfc.nasa.gov/vis/a0000000/a004600/a004631/
→ frames/9600x5400_16x9_30p/MonthlyClimatology/[01-12]_ClimatologyMonthly_032818_
→ 9600x5400.tif}}, local_data: .}, relative_croproyiel: {local_data: ~/climada/data/ISIMIP_
→ crop}}, trop_cyclone: {random_seed: 54}}
```

The configuration itself and its attributes have the data type `climada.util.config.Config`

```
[12]: CONFIG.__class__, CONFIG.hazard.trop_cyclone.random_seed.__class__
```

```
[12]: (climada.util.config.Config, climada.util.config.Config)
```

The actual configuration values can be accessed as basic types (float, int, str), provided the definition is according to the respective data type:

```
[4]: CONFIG.hazard.trop_cyclone.random_seed.int()
```

```
[4]: 54
```

```
[3]: try:
    CONFIG.hazard.trop_cyclone.random_seed.str()
except Exception as e:
    from sys import stderr; stderr.write(f"cannot convert random_seed to str: {e}\n")
cannot convert random_seed to str: <class 'int'>, not str
```

However, configuration string values can be converted to `pathlib.Path` objects if they are pointing to a directory.

```
[19]: CONFIG.hazard.relative_cropyield.local_data.dir()
```

```
[19]: WindowsPath('C:/Users/me/clinada/data/ISIMIP_crop')
```

Note that converting a configuration string to a Path object like this will create the specified directory on the fly, unless `dir` is called with the parameter `create=False`.

6.3.7 2.C. Default Configuration

The configuration file `clinada/conf/clinada.conf` contains the default configuration.

On the top level it has the following attributes - **local_data**: definition of main pathes for accessing and storing

CLIMADA related data - **system**: top directory, where (persistent) clinada data is stored

default: `~/clinada/data` - **demo**: top directory for data that is downloaded or created in the CLIMADA tutorials

default: `~/clinada/demo/data` - **save_dir**: directory where transient (non-persistent) data is stored

default: `./results` - **log_level**: minimum log level showed by logging, one of DEBUG, INFO, WARNING, ERROR or CRITICAL.

default: INFO - **max_matrix_size**: maximum matrix size that can be used, can be decreased in order to avoid memory issues

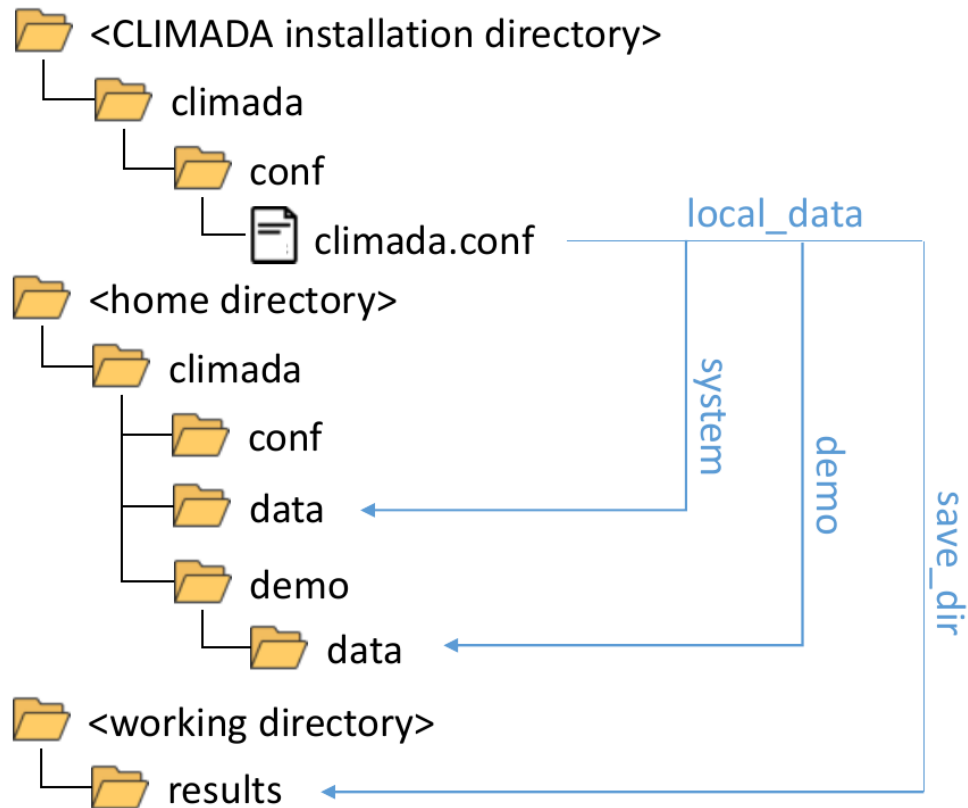
default: 1000000000 (1e8) - **exposures**: exposures modules specific configuration - **hazard**: hazard modules specific configuration

```
[5]: CONFIG.__dict__.keys()
```

```
[5]: dict_keys(['_root', '_comment', 'local_data', 'exposures', 'hazard', 'log_level', 'max_
↪matrix_size'])
```

When `import clinada` is executed in a python script or shell, data files from the installation directory are copied to the location specified in the current configuration.

This happens only when clinada is used for the first time with the current configuration. Subsequent execution will only check for presence of files and won't overwrite existing files.



Thus, the home directory will automatically be populated with a climada directory and several files from the repository when climada is used.

To prevent this and keep the home directory clean, create a config file `~/.config/climada.conf` with customized values for `local_data.system` and `local_data.demo`.

As an example, a file with the following content would suppress creation of directories and copying of files during execution of CLIMADA code:

```
{
  "local_data": {
    "system": "/path/to/installation-dir/climada/data/system",
    "demo": "/path/to/installation-dir/climada/data/demo",
  },
}
```

6.3.8 2.D. Test Configuration

The configuration values for unit and integration tests are not part of the default configuration (2.C), since they are irrelevant for the regular CLIMADA user and only aimed for developers.

The default test configuration is defined in the `climada.conf` file of the installation directory. This file contains paths to files that are read during tests. If they are part of the GitHub repository, their path i.g. starts with the `climada` folder within the installation directory:

```
{
  "_comment": "this is a climada configuration file meant to supersede the default_
↪configuration in climada/conf during test",
  "test_directory": "./climada",
  "test_data": "{test_directory}/test/data",
  "disc_rates": {
    "test_data": "{test_directory}/entity/disc_rates/test/data"
  },
  ...
}
```

Obviously, the default `test_directory` is given as the relative path to `./climada`. This is fine if (but only if) unit or integration tests are started from the installation directory, which is the case in the automated tests on the CI server. Developers who intend to start a test from another working directory may have to edit this file and replace the relative path with the absolute path to the installation directory:

```
{
  "_comment": "this is a climada configuration file meant to supersede the default_
↪configuration in climada/conf during test",
  "test_directory": "/path/to/installation-dir/climada",
  "test_data": "{test_directory}/test/data",
  "disc_rates": {
    "test_data": "{test_directory}/entity/disc_rates/test/data"
  },
  ...
}
```

6.4 Testing and Continuous Integration

6.4.1 Content

1. *Testing CLIMADA*
2. *Notes on Testing*
 1. *Basic Test Procedure*
 2. *Testing Types*
 3. *Unit Tests*
 4. *Integration Tests*

5. *System Tests*
6. *Error Messages*
7. *Dealing with External Resources*
8. *Test Configuration*
3. *Continuous Integration*
 1. *Automated Tests*
 2. *Test Coverage*
 3. *Static Code Analysis*
 4. *Jenkins Projects Overview*

1. Testing CLIMADA

- **Installation Test**

From the installation directory run `make install_test` It lasts about 45 seconds. If it succeeds, CLIMADA is properly installed and ready to use.

- **Unit Tests**

From the installation directory run `make unit_test` It lasts about 5 minutes and runs unit tests for all modules.

- **Integration Tests**

From the installation directory run `make integ_test` It lasts about 45 minutes and runs extensive integration tests, during which also data from external resources is read. An open internet connection is required for a successful test run.

2. Notes on Testing

Any programming code that is meant to be used more than once should have a test, i.e., an additional piece of programming code that is able to check whether the original code is doing what it's supposed to do.

Writing tests is work. As a matter of facts, it can be a *lot* of work, depending on the program often more than writing the original code.

Luckily, it essentially follows always the same basic procedure and there are a lot of tools and frameworks available to facilitate this work.

In CLIMADA we use the Python in-built *test runner* `unittest` for execution of the tests and the `Jenkins` framework for *continuous integration*, i.e., automated test execution and code analysis.

Why do we write test?

- The code is most certainly **buggy** if it's not properly tested.
- Software without tests is **worthless**. It won't be trusted and therefore it won't be used.

When do we write test?

- **Before implementation.** A very good idea. It is called *Test Driven Development*.
- **During implementation.** Test routines can be used to run code even while it's not fully implemented. This is better than running it interactively, because the full context is set up by the test. *By command line:* `python -m unittest climada.x.test_y.TestY.test_z` *Interactively:* `climada.x.test_y.TestY().test_z()`
- **Right after implementation.** In case the coverage analysis shows that there are missing tests, see *Test Coverage*.
- **Later, when a bug was encountered.** Whenever a bug gets fixed, also the tests need to be adapted or amended.

6.4.2 2.A. Basic Test Procedure

- **Test data setup** Creating suitable test data is crucial, but not always trivial. It should be extensive enough to cover all functional requirements and yet as small as possible in order to save resources, both in space and time.
- **Code execution** The main goal of a test is to find bugs *before* the user encounters them. Ultimately every single line of the program should be subject to test. In order to achieve this, it is necessary to run the code with respect to the whole parameter space. In practice that means that even a simple method may require a lot of test code. (Bear this in mind when designing methods or functions: the number of required tests increases dramatically with the number of function parameters!)
- **Result validation** After the code was executed the *actual* result is compared to the *expected* result. The expected result depends on test data, state and parametrization. Therefore result validation can be very extensive. In most cases it won't be practical nor required to validate every single byte. Nevertheless attention should be paid to validate a range of results that is wide enough to discover as many thinkable discrepancies as possible.

6.4.3 2.B. Testing types

Despite the common basic procedure there are many different kinds of tests distinguished. (See *Wikipedia:Software testing*). Very commonly a distinction is made based on levels: - **Unit Test**: tests only a small part of the code, a single function or method, essentially without interaction between modules - **Integration Test**: tests whether different methods and modules work well with each other - **System Test**: tests the whole software at once, using the exposed interface to execute a program

6.4.4 2.C. Unit Tests

Unit tests are meant to check the correctness of program units, i.e., single methods or functions, they are supposed to be fast, simple and easy to write.

For each module in CL

- **Each module in CLIMADA has a counter part containing unit tests.**
Naming suggestion: `climada.x.y` → `climada.x.test.test_y`
- **Write a test class for each class of the module, plus a test class for the module itself in case it contains (module) functions.**

Naming suggestion: `class X` → `class TestX(unittest.TestCase)`, module `climada.x.y` → `class TestY(unittest.TestCase)`

- **Ideally, each method or function should have at least one test method.**

Naming suggestion: `def xy()` → `def test_xy()`, `def test_xy_suffix1()`, `def test_xy_suffix2()`

Functions that are created for the sole purpose of structuring the code do not necessarily have their own unit test.

- **Aim at having *very* fast unit tests!**

*There will be hundreds of unit tests and in general they are called *in corpore* and expected to finish after a reasonable amount of time. Less than 10 milisecond is good, 2 seconds is the maximum acceptable duration*.*

- **A unit test shouldn't call more than one climada method or function.**

The motivation to combine more than one method in a test is usually creation of test data. Try to provide test data by other means. Define them on the spot (within the code of the test module) or create a file in a test data directory that can be read during the test. If this is too tedious, at least move the data acquisition part to the constructor of the test class.

- **Do not use external resources in unit tests.**

Methods depending on external resources can be skipped from unit tests. See 'Dealing with External Resources' <#TestExtern> `__`.

6.4.5 2.D. Integration Tests

Integration tests are meant to check the correctness of interaction between units of a module or a package.

As a general rule, more work is required to write integration tests than to write unit tests and they have longer runtime.

- **Write integration tests for all intended use cases.**
- **Do not expect external resources to be immutable.** If calling on external resources is part of the workflow to be tested, take into account that they may change over time.
If the according API has means to indicate the precise version of the requested data, make use of it, otherwise, adapt your expectations and leave room for future changes.
Example given: your function is ultimately relying on the *current* GDP retrieved from an online data provider, and you test it for Switzerland where it's in about 700 Bio CHF at the moment. Leave room for future development, try to be on a reasonably save side, tolerate a range between 70 Bio CHF and 7000 Bio CHF.
- **Test location:** Integration are written in modules `climada.test.test_xy` or in `climada.x.test.test_y`, like the unit tests.
For the latter it is required that they do not use external resources and that the tests do not have a runtime longer than 2 seconds.

6.4.6 2.E. System Tests

Integration tests are meant to check whether the whole software package is working correctly.

In CLIMADA, the system test that checks the core functionality of the package is executed by calling `make install_test` from the installation directory.

6.4.7 2.F. Error Messages

When a test fails, make sure the raised exception contains all information that might be helpful to identify the exact problem.

If the error message is ever going to be read by someone else than you while still developing the test, you best assume it will be someone who is completely naive about CLIMADA.

Writing extensive failure messages will eventually save more time than it takes to write them.

Putting the failure information into logs is neither required nor sufficient: the automated tests are built around error messages, not logs.

Anything written to `stdout` by a test method is useful mainly for the developer of the test.

6.4.8 2.G. Dealing with External Resources

Methods depending on external resources (calls a url or database) are ideally atomic and doing nothing else than providing data. If this is the case they can be skipped in unit tests on safe grounds - provided they are tested at some point in higher level tests.

In CLIMADA there are the utility functions `climada.util.files_handler.download_file` and `climada.util.files_handler.download_ftp`, which are assigned to exactly this task for the case of external data being available as files.

Any other method that is calling such a data providing method can be made compliant to unit test rules by having an option to replace them by another method. Like this one can write a dummy method in the test module that provides data, e.g., from a file or hard coded, which be given as the optional argument.

```
[7]: import climada
def x(download_file=climada.util.files_handler.download_file):
    filepath = download_file('http://real_data.ch')
    return Path(filepath).stat().st_size

import unittest
class TestX(unittest.TestCase):
    def download_file_dummy(url):
        return "phony_data.ch"

    def test_x(self):
        self.assertEqual(44, x(download_file=self.download_file_dummy))
```

- When introducing a new external resource, add a test method in `test_data_api.py`.

6.4.9 2.H. Test Configuration

Use the configuration file `climada.config` in the installation directory to define file pathes and external resources used during tests (see the *Constants and Configuration Guide*).

3. Continuous Integration

The CLIMADA Jenkins server used for continuous integration is at (<https://ied-wcr-jenkins.ethz.ch>).

6.4.10 3.A. Automated Tests

On Jenkins tests are executed and analyzed automatically, in an unbiased environment. The results are stored and can be compared with previous test runs.

Jenkins has a GUI for monitoring individual tests, full test runs and test result trends.

Developers are requested to watch it. At first when they push commits to the code repository, but also later on, when other changes in data or sources may make it necessary to review and refactor code that once passed all tests. #####

Developer guidelines: - All tests must pass before submitting a pull request. - Integration tests don't run on feature branches in Jenkins, therefore developers are requested to run them locally. - After a pull request was accepted and the changes are merged to the develop branch, integration tests may still fail there and have to be addressed.

6.4.11 3.B. Test Coverage

Jenkins also has an interface for exploring code coverage analysis result.

This shows which part of the code has never been run in any test, by module, by function/method and even by single line of code.

Ultimately every single line of code should be tested.

- Make sure the coverage of novel code is at 100% before submitting a pull request.

Be aware that the having a code coverage alone does not grant that all required tests have been written!

The following artificial exmple would have a 100% coverage and still obviously misses a test for `y(False)`

```
[27]: def x(b:bool):
      if b:
          print('been here')
          return 4
      else:
          print('been there')
          return 0

      def y(b:bool):
          print('been everywhere')
          return 1/x(b)

      import unittest
```

(continues on next page)

(continued from previous page)

```

class TestXY(unittest.TestCase):
    def test_x(self):
        self.assertEqual(x(True), 4)
        self.assertEqual(x(False), 0)

    def test_y(self):
        self.assertEqual(y(True), 0.25)

unittest.TextTestRunner().run(unittest.TestLoader().loadTestsFromTestCase(TestXY));
..
been here
been there
been everywhere
been here

-----
Ran 2 tests in 0.003s

OK

```

6.4.12 3.C. Static Code Analysis

At last Jenkins provides an elaborate GUI for pylint findings which is especially useful when working in feature branches.

Observe it!

- *High Priority Warnings* are as severe as test failures and must be addressed at once.
- Do not introduce new *Medium Priority Warnings*.
- Try to avoid introducing *Low Priority Warnings*, in any case their total number should not increase.

6.4.13 3.D. Jenkins Projects Overview

- **climada_install_env**

Branch: **develop** Runs every day at 1:30AM CET

- creates conda environment from scratch
- runs core functionality system test (make install_test)

- **climada_ci_night**

Branch: **develop** Runs when `climada_install_env` has finished successfully

- runs all test modules
- runs static code analysis

- **climada_branches**

Branch: **any** Runs when a commit is pushed to the repository

- runs all test modules *outside of* `climada.test`
- runs static code analysis

- **climada_data_api**

Branch: **develop** Runs every day at 0:20AM CET

- tests availability of external data APIs

- **climada_data_api**

Branch: **develop** No automated running

- tests executability of CLIMADA tutorial notebooks.

6.5 Reviewer Checklist

- The code must be readable without extra effort from your part. The code should be easily readable (for infos e.g. [here](#))
- Include references to the used algorithms in the docstring
- If the algorithm is new, please include a description in the docstring, or be sure to include a reference as soon as you publish the work
- Variable names should be chosen to be clear. Avoid `item`, `element`, `var`, `list`, `data` etc... A good variable name makes it immediately clear what it contains.
- Avoid as much as possible hard-coded indices for list (no `x = l[0]`, `y = l[1]`). Rather, use tuple unpacking (see [here](#)). Note that tuple unpacking can also be used to update variables. For example, the Fibonacci sequence next number pair can be written as `n1, n2 = n2, n1+n2`.
- Do not use `mutable` (lists, dictionaries, ...) as default values for functions and methods. Do not write:

```
def function(default=[]):
```

but use

```
def function(default=None):
    if default is None: default=[]
```

- Use pythonic loops, [list comprehensions](#)

- Make sure the unit tests are testing all the lines of the code. Do not only check for working cases, but also the most common wrong use cases.
- Check the docstrings (Do they follow the [Numpydoc conventions](#), is everything clearly explained, are the default values given and is it clear why they are set to this value)
- Keep the code simple. Avoid using complex Python functionalities whose use is opaque to non-expert developers unless necessary. For example, the `@staticmethod` decorator should only be used if really necessary. Another example, for counting the dictionary `colors = ['red', 'green', 'red', 'blue', 'green', 'red']`, version: `d = {} for color in colors: d[color] = d.get(color, 0) + 1` is perfectly fine, no need to complicate it to a maybe more pythonic version

```
d = collections.defaultdict(int)
for color in colors:
    d[color] += 1
```

- Did the code writer perform a static code analysis? Does the code respect Pep8 (see also the [pylint config file](#))?
- Did the code writer perform a profiling and checked that there are no obviously inefficient (computation time-wise and memory-wise) parts in the code?

6.6 Coding in Python: Dos and Don'ts

6.6.1 Content

0. To Code or Not to Code?

1. *Clean Code* 1.1 *PEP 8 Quickie: Code Layout* 1.2 *PEP 8 Quickie: Basic Naming Conventions* 1.3 *PEP 8 Quickie: Programming Recommendations* 1.4 *Static Code Analysis and PyLint* 1.5 *A few more best practices* 1.6 *Pythonic Code*
2. *Commenting & Documenting* 2.1 *What is what* 2.2 *Numpy-style docstrings*
3. *Exception Handling and Logging* 3.1 *Exception Handling* 3.2 *Logging*
4. *Importing*
5. *How to structure a method or function*
6. *Debugging*

0. To Code or Not to Code?

Before you start implementing functions which then go into the climada code base, you have to ask yourself a few questions:

Has something similar already been implemented? This is far from trivial to answer! First, search for functions in the same module where you'd be implementing the new piece of code. Then, search in the `util` folders, there's a lot of functions in some of the scripts! You could also search the index (a list of all functions and global constants) in the [climada documentation](#) for key-words that may be indicative of the functionality you're looking for.

Don't expect this process to be fast!

Even if you want to implement *just* a small helper function, which might take 10mins to write, it may take you 30mins to check the existing code base! That's part of the game! Even if you found something, most likely, it's not the *exact* same thing which you had in mind. Then, ask yourself how you can re-use what's there, or whether you can easily add

another option to the existing method to also fit your case, and only if it's nearly impossible or highly unreadable to do so, write your own implementation.

Can my code serve others? You probably have a very specific problem in mind. Yet, think about other use-cases, where people may have a similar problem, and try to either directly account for those, or at least make it easy to configure to other cases. Providing keyword options and hard-coding as few things as possible is usually a good thing. For example, if you want to write a daily aggregation function for some time-series, consider that other people might find it useful to have a general function that can also aggregate by week, month or year.

Can I get started? Before you finally start coding, be sure about placing them in a sensible location. Functions in non-util modules are actually specific for that module (e.g. a file-reader function is probably not river-flood specific, so put it into the `util` section, not the `RiverFlood` module, even if that's what you're currently working on)! If unsure, talk with other people about where your code should go.

If you're implementing more than just a function or two, or even an entirely new module, the planning process should be talked over with someone doing climada-administration.

1. Clean Code

A few basic principles:

- Follow the [PEP 8](#) Style Guide. It contains, among others, recommendations on:
 - code layout
 - basic naming conventions
 - programming recommendations
 - commenting (in detail described in Chapter 4)
 - varia
- Perform a static code analysis - or: PyLint is your friend
- Follow the best practices of *Correctness - Tightness - Readability*
- Adhere to principles of pythonic coding (idiomatic coding, the “python way”)

The Zen of Python

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do  
it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than *right* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!
```

6.6.2 1.1 PEP 8 Quickie: Code Layout

- *Indentation*: 4 spaces per level. For continuation lines, decide between vertical alignment & hanging indentation as shown here:
- *Line limit*: maximum of 79 characters (docstrings & comments 72).
- *Blank lines*:
 - **Two**: Surround top-level function and class definitions;
 - **One**: Surround Method definitions inside a class
 - **Several**: may be used (sparingly) to separate groups of related functions
 - **None**: Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations).
- *Whitespaces*:
 - **None** immediately inside parentheses, brackets or braces; after trailing commas; for keyword assignments in functions.
 - **Do** for assignments (`i = i + 1`), around comparisons (`>=`, `==`, etc.), around booleans (`and`, `or`, `not`)
- There's more in the PEP 8 guide!

6.6.3 1.2 PEP 8 Quickie: Basic Naming Conventions

A short typology: b (single lowercase letter); B (single uppercase letter); lowercase; lower_case_with_underscores; UPPER_CASE; UPPER_CASE_WITH_UNDERSCORES; CapitalizedWords (or CapWords, or CamelCase); mixed-Case; Capitalized_Words_With_Underscores (ugly!)

A few basic rules: - packages and modules: short, all-lowercase names. Underscores can be used in the module name if it improves readability. E.g. `numpy`, `climada` - classes: use the CapWords convention. E.g. `RiverFlood` - functions, methods and variables: lowercase, with words separated by underscores as necessary to improve readability. E.g. `set_raster()`, `dst_meta` - function- and method arguments: Always use `self` for the first argument to instance methods, `cls` for the first argument to class methods. - constants: all capital letters with underscores, e.g. `DEF_VAR_EXCEL`

Use of underscores - `_single_leading_underscore`: weak “internal use” indicator. E.g. `from M import *` does not import objects whose names start with an underscore. A side-note to this: Always decide whether a class’s methods and instance variables (collectively: “attributes”) should be public or non-public. If in doubt, choose non-public; it’s easier to make it public later than to make a public attribute non-public. Public attributes are those that you expect unrelated clients of your class to use, with your commitment to avoid backwards incompatible changes. Non-public attributes are those that are not intended to be used by third parties; you make no guarantees that non-public attributes won’t change or even be removed. Public attributes should have no leading underscores. - `_single_trailing_underscore_`: used by convention to avoid conflicts with Python keyword, e.g. `tkinter`. `Toplevel(master, class_='ClassName')` - `__double_leading_and_trailing_underscore__`: “magic” objects or attributes that live in user-controlled namespaces. E.g. `__init__`, `__import__` or `__file__`. Never invent such names; only use them as documented.

There are many more naming conventions, some a bit messy. Have a look at the PEP8 style guide for more cases.

6.6.4 1.3 PEP 8 Quickie: Programming Recommendations

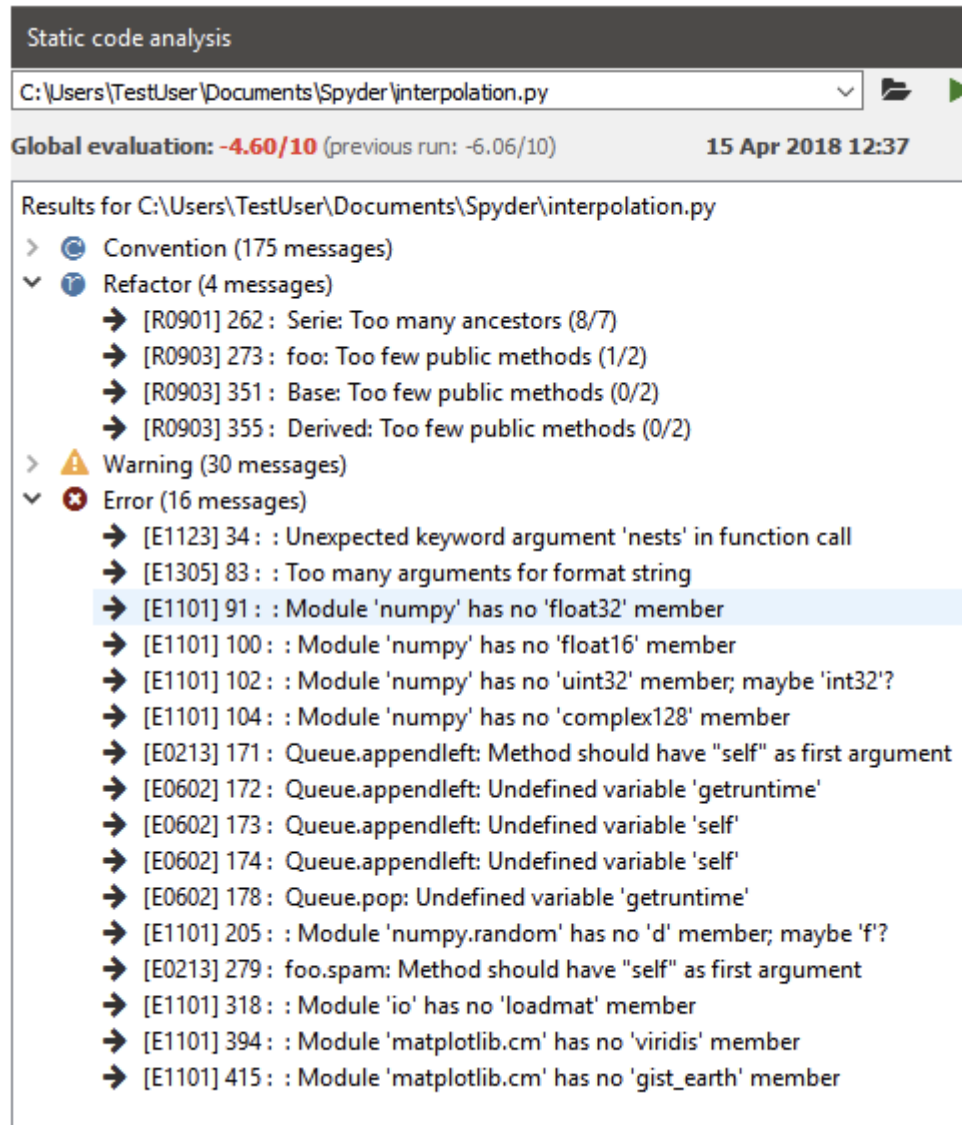
- comparisons to singletons like `None` should always be done with `is` or `is not`, never the equality operators.
- Use `is not` operator rather than `not ... is`.
- Be consistent in return statements. Either all return statements in a function should return an expression, or none of them should. Any return statements where no value is returned should explicitly state this as `return None`.
- Object type comparisons should always use `isinstance()` instead of comparing types directly:
- Remember: sequences (strings, lists, tuples) are false if empty; this can be used:
- Don’t compare boolean values to `True` or `False` using `==`:
- Use `“.startswith()` and `“.endswith()` instead of string slicing to check for prefixes or suffixes.
- Context managers exist and can be useful (mainly for opening and closing files

6.6.5 1.4 Static Code Analysis and PyLint

Static code analysis detects style issues, bad practices, potential bugs, and other quality problems in your code, all without having to actually execute it. In Spyder, this is powered by the best in class Pylint back-end, which can intelligently detect an enormous and customizable range of problem signatures. It follows the style recommended by PEP 8 and also includes the following features: Checking the length of each line, checking that variable names are well-formed according to the project’s coding standard, checking that declared interfaces are truly implemented.

A detailed instruction can be found [here](#).

In brief: In the editor, select the Code Analysis pane (if not visible, go to View -> Panes -> Code Analysis) and the file you want to be analysed; hit the Analyse button.



The output will look somewhat similar to that:

There are 4 categories in the analysis output: * *convention*, * *refactor*, * *warning*, * *error* * a global score regarding code quality.

All messages have a line reference and a short description on the issue. Errors *must* be fixed, as this is a no-go for actually executing the script. Warnings and refactoring messages should be taken seriously; so should be the convention messages, even though some of the naming conventions etc. may not fit the project style. This is configurable. In general, there should be no errors and warnings left, and the overall code quality should be in the “green” range (somewhere above 5 or so).

There are [advanced options](#) to configure the type of warnings and other settings in pylint.

6.6.6 1.5 A few more best practices

Correctness

Methods and functions must return correct and verifiable results, not only under the best circumstances but in any possible context. I.e. ideally there should be unit tests exploring the full space of parameters, configuration and data states. This is often clearly a non-achievable goal, but still - we aim at it.

Tightness

- Avoid code redundancy.
- Make the program efficient, use profiling tools for detection of bottlenecks.
- Try to minimize memory consumption.
- Don't introduce new dependencies (library imports) when the desired functionality is already covered by existing dependencies.
- Stick to already supported file types.

Readability

- Write complete Python Docstrings.
- Use meaningful method and parameter names, and always annotate the data types of parameters and return values.
- No context-dependent return types! Also: Avoid `None` as return type, rather raise an `Exception` instead.
- Be generous with defining `Exception` classes.
- Comment! Comments are welcome to be redundant. And whenever there is a particular reason for the way something is done, comment on it! See below for more detail.
- For functions which implement mathematical/scientific concepts, add the actual mathematical formula as comment or to the Docstrings. This will help maintain a high level of scientific accuracy. E.g. How is the random walk tracks computed for tropical cyclones?

6.6.7 1.6 Pythonic Code

In Python, there are certain structures that are specific to the language, or at least the syntax of how to use them. This is usually referred to as “pythonic” code.

There is an extensive overview on crucial “pythonic” structures and methods in the [Python 101 library](#).

A few important examples are:

- iterables such as dictionaries, tuples, lists
- iterators and generators (a very useful construct when it comes to code performance, as the implementation of generators avoids reading into memory huge iterables at once, and allows to read them lazily on-the-go; see [this blog post](#) for more details)
- f-strings (“formatted string literals,” have an `f` at the beginning and curly braces containing expressions that will

```
>>> name = "Eric"
>>> age = 74
>>> f"Hello, {name}. You are {age}."
'Hello, Eric. You are 74.'
```

be replaced with their values:

- decorators (a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure) something like
- type checking (Python is a dynamically typed language; also: cf. “Duck typing”. Yet, as a best practice, variables should not change type once assigned)
- Do not use mutable default arguments in your functions (e.g. lists). For example, if you define a function as such:

```
def function(x, list=[]):  
    default_list.append(x)
```

Your list will be mutated for future calls of the functions too. The correct implementation would be the following:
`def func(x, list=None): list = [] if list is None`

- lambda functions (little, anonymous functions, sth like `high_ord_func(2, lambda x: x * x)`)
- list comprehensions (a short and possibly elegant syntax to create a new list in one line, sth like `newlist = [x for x in range(10) if x < 5]` returns `[0, 1, 2, 3, 4]`)

It is recommended to look up the above concepts in case not familiar with them.

2. Commenting & Documenting

6.6.8 2.1 What is what

Comments are for developers. They describe parts of the code where necessary to facilitate the understanding of programmers. They are marked by putting a `#` in front of every comment line (for multi-liners, wrapping them inside triple double quotes `"""` is basically possible, but discouraged to not mess up with docstrings). A *documentation string* (*docstring*) is a string that describes a module, function, class, or method definition. The docstring is a special attribute of the object (`object.__doc__`) and, for consistency, is surrounded by triple double quotes (`"""`). This is also where elaboration of the scientific foundation (explanation of used formulae, etc.) should be documented.

A few general rules:

- Have a look at this blog-post on [commenting basics](#)
- Comments should be D.R.Y (“Don’t Repeat Yourself.”)
- Obvious naming conventions can avoid unnecessary comments (cf. `families_by_city[city]` vs. `my_dict[p]`)
- comments should rarely be longer than the code they support
- All public methods need a doc-string. See below for details on the convention used within the climada project.
- Non-public methods that are not immediately obvious to the reader should at least have a short comment after

```
def complicated_function(s):  
    # This function does something complicated
```

the def line:

6.6.9 2.2 Numpy-style docstrings

Full reference can be found [here](#). The standards are such that they use re-structured text (reST) syntax and are rendered using Sphinx.

There are several sections in a docstring, with headings underlined by hyphens (---). The sections of a function's docstring are:

1. *Short summary*: A one-line summary that does not use variable names or the function name

```
def add(a, b):
    """The sum of two numbers.

    """
```

2. *Deprecation warning* (use if applicable): to warn users that the object is deprecated, including version the object that was deprecated, and when it will be removed, reason for deprecation, new recommended way of obtaining the same functionality. Use the deprecated Sphinx directive:

```
.. deprecated:: 1.6.0
    `ndobj_old` will be removed in NumPy 2.0.0, it is replaced by
    `ndobj_new` because the latter works also with array subclasses.
```

3. *Extended Summary*: A few sentences giving an extended description to clarify functionality, not to discuss implementation detail or background theory (see Notes section below!)

```
Parameters
-----
x : type
    Description of parameter `x`.
```

4. *Parameters*: Description of the function arguments, keywords and their respective types. Enclose variables in single backticks in the description. The colon must be preceded by a space, or omitted if the type is absent. For the parameter types, be as precise as possible. If it is not necessary to specify a keyword argument, use `optional` after the type specification: e.g. `x: int, optional`. Default values of optional parameters can also be detailed in the description. (e.g. ... description of parameter ... (default is -1))
5. *Returns*: Explanation of the returned values and their types. Similar to the Parameters section, except the name of each return value is optional, type isn't. If both the name and type are specified, the Returns section takes the same form as the Parameters section.

```
Returns
-----
err_code : int
    Non-zero value indicates error code, or zero on success.
err_msg : str or None
    Human readable error message, or None on success.
```

There is a range of other sections that can be included, if sensible and applicable, such as `Yield` (for generator functions only), `Raises` (which errors get raised and under what conditions), `See also` (refer to related code), `Notes` (additional information about the code, possibly including a discussion of the algorithm; may include mathematical equations, written in LaTeX format), `References`, `Examples` (to illustrate usage).

3. Exception Handling and Logging

Exception handling and logging are two important components of programming, in particular for debugging purposes. Detailed technical guides are available online (e.g., [Loggin](#), [Error and Exceptions](#)). Here we only repeat a few key points and list a few guidelines for CLIMADA.

6.6.10 3.1 Exception handling

1. Catch specific exceptions if possible, i.e, if not needed do not catch all exceptions.
2. Do not catch exception if you do not handle them.
3. Make a clear explanatory message when you raise an error (similarly to when you use the logger to inform the user). Think of future users and how it helps them understanding the error and debugging their code.
4. Catch an exception when it arises.
5. When you can an exception and raise an error, it is in often (but not always) a good habit to not throw away the first caught exception as it may contain useful information for debugging. (use `raise Error from`)

```
[1]: #Bad (1)
x = 1
try:
    l = len(events)
    if l < 1:
        print("l is too short")
except:
    pass
```

```
[ ]: #Still bad (2)
try:
    l = len(events)
    if l < 1:
        print("l is too short")
except TypeError:
    pass
```

```
[ ]: #Better, but still insufficient (3)
try:
    l = len(events)
    if l < 1:
        raise ValueError("To compute an impact there must be at least one event.")
except TypeError:
    raise TypeError("The provided variable events is not a list")
```

```
[ ]: #Even better (4)
try:
    l = len(events)
except TypeError:
    raise TypeError("The provided variable events is not a list")
if l < 1:
    raise ValueError("To compute an impact there must be at least one event.")
```



```
[ ]: #Even better (5)
try:
    l = len(events)
except TypeError as tper:
    raise TypeError("The provided variable events is not a list") from tper
if l < 1:
    raise ValueError("To compute an impact there must be at least one event.")
```

Why do we bother to handle exceptions?

- The most essential benefit is to inform the user of the error, while still allowing the program to proceed.

6.6.11 3.2 Logging

- In CLIMADA, you cannot use printing. Any output must go into the `LOGGER`.
- For any logging messages, always think about the audience. What would a user or developer need for information? This also implies to carefully think about the correct `LOGGER` level. For instance, some information is for debugging, then use the debug level. In this case, make sure that the message actually helps the debugging process! Some message might just to inform the user about certain default parameters, then use the inform level. See below for more details about logger levels.
- Do not overuse the `LOGGER`. Think about which level of logging. Logging errors must be useful for debugging.

You can set the level of the `LOGGER` using `climada.util.config.LOGGER.setLevel(logging.XXX)`. This way you can for instance ‘turn-off’ info messages when you are making an application. For example, setting the logger to the “ERROR” level, use:

```
[ ]: import logging
from climada.util.config import LOGGER
LOGGER.setLevel(logging.ERROR)
```

What levels to use in CLIMADA?

- Debug: what you would print while developing/debugging
- Info: information for example in the check instance
- Warning: whenever CLIMADA fills in values, makes an extrapolation, computes something that might potentially lead to unwanted results (e.g., the 250year damages extrapolated from data over 20 years)

No known use case:

- Error: instead, raise an Error and add the message (`raise ValueError("Error message")`)
- Critical: ...

“Logging is a means of tracking events that happen when some software runs.”

When to use logging

“Logging provides a set of convenience functions for simple logging usage. These are `debug()`, `info()`, `warning()`, `error()` and `critical()`. To determine when to use logging, see the table below, which states, for each of a set of common tasks, the best tool to use for it.”

Task you want to perform	The best tool for the task
Display console output for ordinary usage of a command line script or program	<code>print()</code>
Report events that occur during normal operation of a program (e.g. for status monitoring or fault investigation)	<code>logging.info()</code> (or <code>logging.debug()</code> for very detailed output for diagnostic purposes)
Issue a warning regarding a particular runtime event	<code>warnings.warn()</code> in library code if the issue is avoidable and the client application should be modified to eliminate the warning <code>logging.warning()</code> if there is nothing the client application can do about the situation, but the event should still be noted
Report an error regarding a particular runtime event	Raise an exception
Report suppression of an error without raising an exception (e.g. error handler in a long-running server process)	<code>logging.error()</code> , <code>logging.exception()</code> or <code>logging.critical()</code> as appropriate for the specific error and application domain

Logger level

“The logging functions are named after the level or severity of the events they are used to track. The standard levels and their applicability are described below (in increasing order of severity):”

Level	When it's used
DEBUG	Detailed information, typically of interest only when diagnosing problems.
INFO	Confirmation that things are working as expected.
WARNING	An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
ERROR	Due to a more serious problem, the software has not been able to perform some function.
CRITICAL	A serious error, indicating that the program itself may be unable to continue running.

4. Importing

General remarks

- Imports should be grouped in the following order:
 - Standard library imports (such as `re`, `math`, `datetime`, cf. [here](#))
 - Related third party imports (such as `numpy`)
 - Local application/library specific imports (such as `climada.hazard.base`)
- You should put a blank line between each group of imports.

- Don't introduce new dependencies (library imports) when the desired functionality is already covered by existing dependencies.

Avoid circular importing!! Circular imports are a form of circular dependencies that are created with the import statement in Python; e.g. module A loads a method in module B, which in turn requires loading module A. This can generate problems such as tight coupling between modules, reduced code reusability, more difficult maintainance. Circular dependencies can be the source of potential failures, such as infinite recursions, memory leaks, and cascade effects. Generally, they can be resolved with better code design. Have a look [here](#) for tips to identify and resolve such imports.

Varia * there are absolute imports (uses the full path starting from the project's root folder) and relative imports (uses the path starting from the current module to the desired module; usually in the for `from .<module/package> import X`; dots `.` indicate how many directories upwards to traverse. A single dot corresponds to the current directory; two dots indicate one folder up; etc.) * generally try to avoid star imports (e.g. `from packagename import *`)

Importing utility functions

When importing CLIMADA utility functions (from `climada.util`), the convention is to import the function as “u_name_of_function”, e.g.:

```
from climada.util import coordinates as u_coord
u_coord.make_map()
```

5. How to structure a method or function

To clarify ahead: The questions of [how to structure an entire module](#), or even “just” a class, are not treated here. For this, please get in contact with the [repository admins](#) to help you go devise a plan.

The following few principles should be adhered to when designing a function or method (which is simply the term for a function inside a class):

- have a look at this [blog-post](#) summarizing a few important points to define your function (key-words *abstraction*, *reusability*, *modularity*)
- separate algorithmic computations and data curation
- adhere to a maximum method length (rule of thumb: if it doesn't fit your screen, it's probably an indicator that you should refactor into sub-functions)
- divide functions into single purpose pieces (one function, one goal)

6. Debugging

When writing code, you will encounter bugs and hence go through (more or less painful) debugging. Depending on the IDE you use, there are different debugging tools that will make your life much easier. They offer functionalities such as stopping the execution of the function just before the bug occurs (via breakpoints), allowing to explore the state of defined variables at this moment of time.

For spyder specifically, have a look at the instructions on [how to use ipdb](#)

6.7 Python performance tips and best practice for CLIMADA developers

This guide covers the following recommendations:

Use profiling tools to find and assess performance bottlenecks. **Replace for-loops** by built-in functions and efficient external implementations. **Consider algorithmic performance**, not only implementation performance. **Get familiar with NumPy**: vectorized functions, slicing, masks and broadcasting. **Miscellaneous**: sparse arrays, Numba, parallelization, huge files (xarray), memory. **Don't over-optimize** at the expense of readability and usability.

Table of Contents

- 1 Profiling
- 2 General considerations
 - 2.1 for-loops
 - 2.2 Converting data structures
 - 2.3 Always consider several implementations
 - 2.4 Efficient algorithms
 - 2.5 Memory usage
- 3 NumPy-related tips and best practice
 - 3.1 Vectorized functions
 - 3.2 Broadcasting
 - 3.3 A note on in-place operations
- 4 Miscellaneous
 - 4.1 Sparse matrices
 - 4.2 Fast for-loops using Numba
 - 4.3 Parallelizing tasks
 - 4.4 Read NetCDF datasets with xarray
- 5 Take-home messages

6.7.1 1 Profiling

Python comes with powerful packages for the **performance assessment** of your code. Within IPython and notebooks, there are several magic commands for this task:

- `%time`: Time the execution of a single statement
- `%timeit`: Time repeated execution of a single statement for more accuracy
- `%%timeit` Does the same as `%timeit` for a whole cell
- `%prun`: Run code with the profiler
- `%lprun`: Run code with the line-by-line profiler
- `%memit`: Measure the memory use of a single statement
- `%mprun`: Run code with the line-by-line memory profiler

More information on profiling in the [Python Data Science Handbook](#).

Also useful: unofficial Jupyter extension [Execute Time](#).

While it's easy to assess how fast or slow parts of your code are, including finding the bottlenecks, **generating an improved version of it is much harder**. This guide is about **simple best practices** that everyone should know who works with Python, especially when models are performance-critical.

In the following, we will **focus on arithmetic operations** because they play an important role in CLIMADA. Operations on non-numeric objects like strings, graphs, databases, file or network IO might be just as relevant inside and outside of the CLIMADA context. Some of the tips presented here do also apply to other contexts, but **it's always worth looking for context-specific performance guides**.

6.7.2 2 General considerations

This section will be concerned with:

for-loops and built-ins **external implementations** and converting data structures **algorithmic efficiency** **memory usage**

As this section's toy example, let's assume we want to sum up all the numbers in a list:

```
[ ]: list_of_numbers = list(range(10000))
```

2.1 for-loops

A developer with a background in C++ would probably loop over the entries of the list:

```
[2]: %%timeit
result = 0
for i in list_of_numbers:
    result += i

332 µs ± 65.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

The built-in function `sum` is much faster:

```
[3]: %timeit sum(list_of_numbers)

54.9 µs ± 5.63 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

The timing improves by a factor of 5-6 and this is not a coincidence: **for-loops generally tend to get prohibitively expensive** when the number of iterations increases.

When you have a for-loop with many iterations in your code, check for built-in functions or efficient external implementations of your programming task.

A special case worth noting are append operations on lists which can often be replaced by more efficient *list comprehensions*.

2.2 Converting data structures

When you find an external library that solves your task efficiently, always consider that it might be necessary to convert your data structure which takes time.

For arithmetic operations, NumPy is a great library, but if your data comes as a Python list, NumPy will spend quite some time converting it to a NumPy array:

```
[4]: import numpy as np
      %timeit np.sum(list_of_numbers)

572 µs ± 80 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

This operation is even slower than the for-loop!

However, if you can somehow obtain your data in the form of **NumPy arrays from the start**, or if you perform many operations that might compensate for the conversion time, the gain in performance can be considerable:

```
[5]: # do the conversion outside of the `%timeit`
      ndarray_of_numbers = np.array(list_of_numbers)
      %timeit np.sum(ndarray_of_numbers)

10.6 µs ± 1.56 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Indeed, this is 5-6 times faster than the built-in sum and 20-30 times faster than the for-loop.

2.3 Always consider several implementations

Even for such a basic task as summing, there exist several implementations whose performance can vary more than you might expect:

```
[6]: %timeit ndarray_of_numbers.sum()
      %timeit np.einsum("i->", ndarray_of_numbers)

9.07 µs ± 1.39 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
5.55 µs ± 383 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

This is up to 50 times faster than the for-loop. More information about the `einsum` function will be given in the NumPy section of this guide.

2.4 Efficient algorithms

Consider algorithmic performance, not only implementation performance.

All of the examples above do exactly the same thing, algorithmically. However, often the largest performance improvements can be obtained from **algorithmical changes**. This is the case when your model or your data contain symmetries or more complex structure that allows you to skip or boil down arithmetic operations.

In our example, we are summing the numbers from 1 to 10,000 and it's a well known mathematical theorem that this can be done using only two multiplications and an increment:

```
[7]: n = max(list_of_numbers)
      %timeit 0.5 * n * (n + 1)

83.1 ns ± 2.5 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

Not surprisingly, This is almost 100 times faster than even the fastest implementation of the 10,000 summing operations listed above.

You don't need a degree in maths to find algorithmical improvements. Other algorithmical improvements that are often easy to detect are: * **Filter your data set as much as possible** to perform operations only on those entries that are really relevant. ***Example:*** When computing a physical hazard (e.g. extreme wind) with CLIMADA, restrict to Centroids on land unless you know that some of your exposure is off shore. * Make sure to **detect inconsistent or trivial input parameters early on**, before starting any operations. ***Example:*** If your code does some complicated stuff and applies a user-provided normalization factor at the very end, make sure to check that the factor is not 0 before you start applying those complicated operations.

In general: Before starting to code, take pen and paper and write down what you want to do from an algorithmic perspective.

2.5 Memory usage

Be careful with deep copies of large data sets and only load portions of large files into memory as needed.

Write your code in such a way that you **handle large amounts of data chunk by chunk** so that Python does not need to load everything into memory before performing any operations. When you do, Python's [generators](#) might help you with the implementation.

Allocating unnecessary amounts of memory might slow down your code substantially due to swapping.

6.7.3 3 NumPy-related tips and best practice

As mentioned above, arithmetic operations in Python can profit a lot from NumPy's capabilities. In this section, we collect some tips how to make use of NumPy's capabilities when performance is an issue.

3.1 Vectorized functions

We mentioned above that Python's **for-loops are really slow**. This is even more important when looping over the entries in a NumPy array. Fortunately, NumPy's masks, slicing notation and vectorization capabilities help to avoid for-loops in almost every possible situation:

```
[8]: # TASK: compute the column-sum of a 2-dimensional array
input_arr = np.random.rand(100, 3)
```

```
[9]: %%timeit
# SLOW: summing over columns using loops
output = np.zeros(100)
for row_i in range(input_arr.shape[0]):
    for col_i in range(input_arr.shape[1]):
        output[row_i] += input_arr[row_i, col_i]

145 µs ± 5.47 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

```
[10]: # FASTER: using NumPy's vectorized `sum` function with `axis` attribute
%%timeit output = input_arr.sum(axis=1)

4.23 µs ± 216 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In the special case of multiplications and sums (linear operations) over the axes of two multi-dimensional arrays, NumPy's ``**einsum**`` is even faster:

```
[11]: %timeit output = np.einsum("ij->i", input_arr)
2.38 µs ± 214 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Another einsum example: **Euclidean norms**

```
[12]: many_vectors = np.random.rand(1000, 3)
%timeit np.sqrt((many_vectors**2).sum(axis=1))
%timeit np.linalg.norm(many_vectors, axis=1)
%timeit np.sqrt(np.einsum("...j,...j->...", many_vectors, many_vectors))

24.4 µs ± 2.18 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
26.5 µs ± 2.44 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
9.5 µs ± 91.1 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

For more information about the capabilities of NumPy's einsum function, refer to [the official NumPy documentation](#). However, note that future releases of NumPy will eventually improve the performance of core functions, so that einsum will become an example of over-optimization (see above) at some point. Whenever you use einsum, consider adding a comment that explains what it does for users that are not familiar with einsum's syntax.

Not only sum, but many NumPy functions come with similar vectorization capabilities. You can take minima, maxima, means or standard deviations along selected axes. But did you know that the same is true for the diff and argmin functions?

```
[13]: arr = np.random.randint(low=0, high=10, size=(4, 3))
arr
```

```
[13]: array([[4, 2, 6],
           [2, 3, 4],
           [3, 3, 3],
           [3, 2, 4]])
```

```
[14]: arr.argmin(axis=1)
```

```
[14]: array([1, 0, 0, 1])
```

3.2 Broadcasting

When operations are performed on several arrays, possibly of differing shapes, be sure to use NumPy's **broadcasting** capabilities. This will save you a lot of memory and time when performing arithmetic operations.

Example: We want to multiply the columns of a two-dimensional array by values stored in a one-dimensional array. There are two naive approaches to this:

```
[15]: input_arr = np.random.rand(100, 3)
col_factors = np.random.rand(3)
```

```
[16]: # SLOW: stack/tile the one-dimensional array to be two-dimensional
%timeit output = np.tile(col_factors, (input_arr.shape[0], 1)) * input_arr

5.67 µs ± 718 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
[17]: %%timeit
# SLOW: loop over columns and factors
```

(continues on next page)

(continued from previous page)

```
output = input_arr.copy()
for i, factor in enumerate(col_factors):
    output[:, i] *= factor
```

9.63 μs \pm 95.2 ns per loop (mean \pm std. dev. of 7 runs, 1000000 loops each)

The idea of *broadcasting* is that NumPy **automatically matches axes from right to left and implicitly repeats data along missing axes** if necessary:

```
[18]: %timeit output = col_factors * input_arr
```

1.41 μs \pm 51.7 ns per loop (mean \pm std. dev. of 7 runs, 10000000 loops each)

For automatic broadcasting, the *trailing* dimensions of two arrays have to match. NumPy is matching the shapes of the arrays *from right to left*. If you happen to have arrays where other dimensions match, **you have to tell NumPy which dimensions to add by adding an axis of length 1 for each missing dimension**:

```
[19]: input_arr = np.random.rand(3, 100)
row_factors = np.random.rand(3)
output = row_factors.reshape(3, 1) * input_arr
```

Because this concept is so important, there is a short-hand notation for adding an axis of length 1. In the slicing notation, **add ``None`` in those positions where broadcasting should take place**.

```
[20]: input_arr = np.random.rand(3, 100)
row_factors = np.random.rand(3)
output = row_factors[:, None] * input_arr
```

```
[21]: input_arr = np.random.rand(7, 3, 5, 4, 6)
factors = np.random.rand(7, 3, 4)
output = factors[:, :, None, :, None] * input_arr
```

3.3 A note on in-place operations

While **in-place operations** are generally faster than long and explicit expressions, they shouldn't be over-estimated when looking for performance bottlenecks. Often, the loss in code readability is not justified because NumPy's memory management is really fast.

Don't over-optimize!

```
[22]: shape = (1200, 1700)
arr_a = np.random.rand(*shape)
arr_b = np.random.rand(*shape)
arr_c = np.random.rand(*shape)
```

```
[23]: # long expression in one line
%timeit arr_d = arr_c * (arr_a + arr_b) - arr_a + arr_c
```

17.3 ms \pm 820 μs per loop (mean \pm std. dev. of 7 runs, 10 loops each)

```
[24]: %%timeit
# almost same performance: in-place operations
arr_d = arr_a + arr_b
```

(continues on next page)

(continued from previous page)

```
arr_d *= arr_c
arr_d -= arr_a
arr_d += arr_c
```

17.4 ms ± 618 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

```
[25]: %load_ext memory_profiler
```

```
[26]: # long expression in one line
```

```
%memit arr_d = arr_c * (arr_a + arr_b) - arr_a + arr_c
```

peak memory: 156.68 MiB, increment: 31.20 MiB

```
[27]: %%memit
```

```
# almost same memory usage: in-place operations
```

```
arr_d = arr_a + arr_b
```

```
arr_d *= arr_c
```

```
arr_d -= arr_a
```

```
arr_d += arr_c
```

peak memory: 157.27 MiB, increment: 0.00 MiB

6.7.4 4 Miscellaneous

4.1 Sparse matrices

In many contexts, we deal with sparse matrices or sparse data structures, i.e. two-dimensional arrays where most of the entries are 0. In CLIMADA, this is especially the case for the intensity attributes of Hazard objects. This kind of data is usually handled using SciPy's submodule `scipy.sparse`.

When dealing with sparse matrices make sure that you always understand exactly which of your variables are sparse and which are dense and only switch from sparse to dense when absolutely necessary.

Multiplications (``multiply``) and matrix multiplications (``dot``) are often faster than operations that involve masks or indexing.

As an example for the last rule, consider the problem of multiplying certain rows of a sparse array by a scalar:

```
[28]: import scipy.sparse as sparse
```

```
array = np.tile(np.array([0, 0, 0, 2, 0, 0, 0, 1, 0], dtype=np.float64), (100, 80))
```

```
row_mask = np.tile(np.array([False, False, True, False, True], dtype=bool), (20,))
```

In the following cells, note that the code in the first line after the `%%timeit` statement is not timed, it's the setup line.

```
[29]: %%timeit sparse_array = sparse.csr_matrix(array)
```

```
sparse_array[row_mask, :] *= 5
```

```
/home/tovogt/.local/share/miniconda3/envs/tc/lib/python3.7/site-packages/scipy/sparse/
↳data.py:55: RuntimeWarning: overflow encountered in multiply
    self.data *= other
```

1.52 ms ± 155 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

```
[30]: %%timeit sparse_array = sparse.csr_matrix(array)
sparse_array.multiply(np.where(row_mask, 5, 1)[: , None]).tocsr()

340 µs ± 7.32 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
[31]: %%timeit sparse_array = sparse.csr_matrix(array)
sparse.diags(np.where(row_mask, 5, 1)).dot(sparse_array)

400 µs ± 6.43 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

4.2 Fast for-loops using Numba

As a last resort, if there's no way to avoid a for-loop even with NumPy's vectorization capabilities, you can use the `@njit` decorator provided by the Numba package:

```
[32]: from numba import njit

@njit
def sum_array(arr):
    result = 0.0
    for i in range(arr.shape[0]):
        result += arr[i]
    return result
```

In fact, the Numba function is more than 100 times faster than without the decorator:

```
[33]: input_arr = np.float64(np.random.randint(low=0, high=10, size=(10000,)))
```

```
[34]: %timeit sum_array(input_arr)

10.9 µs ± 444 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
[35]: # Call the function without the @njit
%timeit sum_array.py_func(input_arr)

1.84 ms ± 65.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

However, whenever available, NumPy's own vectorized functions will usually be faster than Numba.

```
[36]: %timeit np.sum(input_arr)
%timeit input_arr.sum()
%timeit np.einsum("i->", input_arr)

7.6 µs ± 687 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
5.27 µs ± 411 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
7.89 µs ± 499 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Make sure you understand the basic idea behind Numba before using it, read the ``Numba docs <https://numba.readthedocs.io/en/stable/user/5minguide.html>``.

Don't use ``@jit``, but use ``@njit`` which is an alias for ``@jit(nopython=True)``.

When you know what you are doing, the `fastmath` and `parallel` options can boost performance even further: read more about this in the [Numba docs](#).

4.3 Parallelizing tasks

Depending on your hardware setup, parallelizing tasks using `pathos` and Numba's automatic parallelization feature can improve the performance of your implementation.

Expensive hardware is no excuse for inefficient code.

Many tasks in CLIMADA could profit from GPU implementations. However, currently there are **no plans to include GPU support in CLIMADA** because of the considerable development and maintenance workload that would come with it. If you want to change this, contact the core team of developers, open an issue or mention it in the bi-weekly meetings.

4.4 Read NetCDF datasets with `xarray`

When dealing with NetCDF datasets, memory is often an issue, because even if the file is only a few megabytes in size, the uncompressed raw arrays contained within can be several gigabytes large (especially when data is sparse or similarly structured). One way of dealing with this situation is to open the dataset with `xarray`.

```xarray``` allows to read the shape and type of variables contained in the dataset without loading any of the actual data into memory.

Furthermore, when loading slices and arithmetically aggregating variables, memory is allocated not more than necessary, but values are obtained on-the-fly from the file.

## 6.7.5 5 Take-home messages

We conclude by repeating the gist of this guide:

**Use profiling tools** to find and assess performance bottlenecks. **Replace for-loops** by built-in functions and efficient external implementations. **Consider algorithmic performance**, not only implementation performance. **Get familiar with NumPy**: vectorized functions, slicing, masks and broadcasting. **Miscellaneous**: sparse arrays, Numba, parallelization, huge files (`xarray`), memory. **Don't over-optimize** at the expense of readability and usability.

## 6.8 Miscellaneous CLIMADA conventions

Table of Contents

```
<ul class="toc-item">
 <a href="#Miscellaneous-CLIMADA-conventions" data-toc-modified-id=
 ↪ "Miscellaneous-CLIMADA-conventions-1">
 1
 Miscellaneous CLIMADA conventions
 <ul class="toc-item">
 <a href="#Dependencies-(python-packages)" data-toc-modified-id=
 ↪ "Dependencies-(python-packages)-1.1">
 1.1
 Dependencies (python packages)
 <a href="#Class-inheritance" data-toc-modified-id="Class-
 ↪ inheritance-1.2">
 1.2
 Class inheritance
 <a href="#Does-it-belong-into-CLIMADA?" data-toc-modified-id="Does-
 ↪ it-belong-into-CLIMADA?-1.3">
```

(continues on next page)

(continued from previous page)

```

 1.3
 Does it belong into CLIMADA?
 <a href="#Paper-repository" data-toc-modified-id="Paper-repository-
↪1.4">
 1.4
 Paper repository
 <a href="#Utility-function" data-toc-modified-id="Utility-function-
↪1.5">
 1.5
 Utility function
 <a href="#Impact-function-renaming---if-to-impf" data-toc-modified-
↪id="Impact-function-renaming---if-to-impf-1.6">
 1.6
 Impact function renaming - if to impf
 <a href="#Data-dependencies" data-toc-modified-id="Data-
↪dependencies-1.7">
 1.7
 Data dependencies
 <a href="#Side-Note-on-Parameters" data-toc-modified-id="Side-Note-
↪on-Parameters-1.8">
 1.8
 Side Note on Parameters


```

### 6.8.1 Dependencies (python packages)

Python is extremely powerful thanks to the large amount of available libraries, packages and modules. However, maintaining a code with a large number of such packages creates dependencies which is very care intensive. Indeed, each package developer can and does update and develop continuously. This means that certain code can become obsolete over time, stop working altogether, or become incompatible with other packages. Hence, it is crucial to keep the philosophie:

*As many packages as needed, as few as possible.*

Thus, when you are coding, follow these priorities:

1. Python standard library
2. Funktions and methods already implemented in CLIMADA (do NOT introduce circulary imports though)
3. Packages already included in CLIMADA
4. Before adding a new dependency:
  - Contact a [repository admin](#) to get permission
  - Open an [issue](#)

Hence, first try to solve your problem with the standard library and function/methods already implemented in CLIMADA (see in particular the *util functions*) then use the packages included in CLIMADA, and if this is not enough, propose the addition of a new package. Do not hesitate to propose new packages if this is needed for your work!

## 6.8.2 Class inheritance

In Python, a [class can inherit from other classes](#), which is a very useful mechanism in certain circumstance. However, it is wise to think about inheritance before implementing it. Very important, is that CLIMADA classes do not inherit from external library classes. For example, `Exposure` directly inherited from `Geopandas`. This caused problems in CLIMADA when the package `Geopandas` was updated.

**CLIMADA classes shall NOT inherit classes from external modules**

## 6.8.3 Does it belong into CLIMADA?

When developing for CLIMADA, it is important to distinguish between core content and particular applications. Core content is meant to be included into the `climada_python` repository and will be subject to a code review. Any new addition should first be discussed with one of the [repository admins](#). The purpose of this discussion is to see

- How does the planned module fit into CLIMADA?
- What is an optimal architecture for the new module?
- What parts might already exist in other parts of the code?

Applications made with CLIMADA, such as an [ECA study](#) can be stored in the [paper repository](#) once they have been published. For other types of work, consider making a separate repository that imports CLIMADA as an external package.

## 6.8.4 Paper repository

Applications made with CLIMADA which are published in the form of a paper or a report are very much encouraged to be submitted to the [climada/paper](#) repository. You can either:

- Prepare a well-commented jupyter notebook with the code necessary to reproduce your results and upload it to the [climada/paper](#) repository. Note however that the repository cannot be used for storing data files.
- Upload the code necessary to reproduce your results to a separate repository of your own. Then, add a link to your repository and to your publication to the readme file on the [climada/paper](#) repository.

### Notes about DOI

Some journals requires you to provide a DOI to the code and data used for your publication. In this case, we encourage to create a separate repository for your code and create a DOI using [Zenodo](#) or any specific service from your institution (e.g. [ETH Zürich](#)).

The CLIMADA releases are also identified with a DOI.

## 6.8.5 Utility function

In CLIMADA, there is a set of utility functions defined in `climada.util`. A few examples are:

- convert large monetary numbers into thousands, millions or billions together with the correct unit name
- compute distances
- load hdf5 files
- convert iso country numbers between formats
- ...

Whenever you develop a module or make a code review, be attentive to see whether a given functionality has already been implemented as a utility function. In addition, think carefully whether a given function/method does belong in its module or is actually independent of any particular module and should be defined as a utility function.

It is very important to not reinvent the wheel and to avoid unnecessary redundancies in the code. This makes maintenance and debugging very tedious.

## 6.8.6 Impact function renaming - `if` to `impf`

In the original CLIMADA code, the impact function is often referred to as `if` or `if_`. This is easy to confuse with the conditional operator *if*. Hence, in future a transition from

`if` ———> `impf`

will be performed. Once the change is active, known developers will be notified and this message updated.

## 6.8.7 Data dependencies

### 6.8.8 Web APIs

CLIMADA relies on open data available through web APIs such as those of the World Bank, Natural Earth, NASA and NOAA. You might execute the test `climada_python-x.y.z/test_data_api.py` to check that all the APIs used are active. If any is out of service (temporarily or permanently), the test will indicate which one.

### 6.8.9 Manual download

As indicated in the software and tutorials, other data might need to be downloaded manually by the user. The following table shows these last data sources, their version used, its current availability and where they are used within CLIMADA:

Availability	Name	Version	Link	CLIMADA class	CLIMADA version	CLIMADA tutorial reference
OK	Fire Information for Resource Management System		<a href="#">FIRMS</a>	BushFire	>V1.2.5	cli-mada_hazard_BushFire.ipynb
OK	Gridded Population of the World (GPW)	v4.11	<a href="#">GPW4.11</a>	LitPop	> v1.2.3	cli-mada_entity_LitPop.ipynb
FAILED	Gridded Population of the World (GPW)	v4.10	<a href="#">GPW1.10</a>	LitPop	>= v1.2.0	cli-mada_entity_LitPop.ipynb

### 6.8.10 Side Note on Parameters

**Don't use `*args` and `**kwargs` parameters without a very good reason.**

There *are* valid use cases for [this kind of parameter notation](#).

In particular `*args` comes in handy when there is an unknown number of equal typed arguments to be passed. E.g., the `pathlib.Path` constructor.

But if the parameters are expected to be structured in any way, it is just a bad idea.

```
[4]: def f(x, y, z):
 return x + y + z

 # bad in most cases
 def g(*args, **kwargs):
 x = args[0]
 y = kwargs['y']
 s = f(*args, **kwargs)
 print(x, y, s)

 g(1,y=2,z=3)

 1 2 6
```

```
[]: # usually just fine
 def g(x, y, z):
 s = f(x, y, z)
 print(x, y, s)

 g(1,y=2,z=3)
```

### Decrease the number of parameters.

Though CLIMADA's pylint configuration .pylintrc allows 7 arguments for any method or function before it complains, it is advisable to aim for less. It is quite likely that a function with so many parameters has an inherent design flaw. There are very well designed command line tools with innumerable optional arguments, e.g., rsync - but these are command line tools. There are also methods like pandas.DataFrame.plot() with countless optional arguments and it makes perfectly sense.

But within the climada package it probably doesn't. divide et impera!

Whenever a method has more than 5 parameters, it is more than likely that it can be refactored pretty easily into two or more methods with less parameters and less complexity:

```
[2]: def f(a, b, c, d, e, f, g, h):
 print(f'f does many things with a lot of arguments: {a, b, c, d, e, f, g, h}')
 return sum([a, b, c, d, e, f, g, h])

 f(1, 2, 3, 4, 5, 6, 7, 8)

 f does many things with a lot of arguments: (1, 2, 3, 4, 5, 6, 7, 8)

[2]: 36
```

```
[3]: def f1(a, b, c, d):
 print(f'f1 does less things with fewer arguments: {a, b, c, d}')
 return sum([a, b, c, d])

 def f2(e, f, g, h):
 print(f'f2 dito: {e, f, g, h}')
 return sum([e, f, g, h])
```

(continues on next page)



(continued from previous page)

```
def f3(x, y):
 print(f'f3 dito, but on a higher level: {x, y}')
 return sum([x, y])
```

```
f3(f1(1, 2, 3, 4), f2(5, 6, 7, 8))
```

```
f1 does less things with fewer arguments: (1, 2, 3, 4)
```

```
f2 dito: (5, 6, 7, 8)
```

```
f3 dito, but on a higher level: (10, 26)
```

```
[3]: 36
```

This of course pleads the case on a strictly formal level. No real complexities have been reduced during the making of this example.

Nevertheless there is the benefit of reduced test case requirements. And in real life real complexity *will* be reduced.



## SOFTWARE DOCUMENTATION

Documents functions, classes and methods:

### 7.1 Software documentation per package

#### 7.1.1 climada.engine package

**climada.engine.uncertainty\_quantification package**

**climada.engine.uncertainty\_quantification.calc\_base module**

**class** climada.engine.uncertainty\_quantification.calc\_base.Calc

Bases: object

Base class for uncertainty quantification

Contains the generic sampling and sensitivity methods. For computing the uncertainty distribution for specific CLIMADA outputs see the subclass CalcImpact and CalcCostBenefit.

**\_\_init\_\_()**

Empty constructor to be overwritten by subclasses

**check\_distr()**

Log warning if input parameters repeated among input variables

**Returns**

**Return type** True.

**property input\_vars**

Uncertainty variables

**Returns** All uncertainty variables associated with the calculation

**Return type** tuple(uncvar)

**property distr\_dict**

Dictionary of the input variable distribution

Probability density distribution of all the parameters of all the uncertainty variables listed in self.InputVars

**Returns** **distr\_dict** – Dictionary of all probability density distributions.

**Return type** dict( sp.stats objects )

**est\_comp\_time(n\_samples, time\_one\_run, pool=None)**

Estimate the computation time

**Parameters**

- **n\_samples** (*int/float*) – The total number of samples
- **time\_one\_run** (*int/float*) – Estimated computation time for one parameter set in seconds
- **pool** (*pathos.pool, optional*) – pool that would be used for parallel computation. The default is None.

**Returns**

**Return type** Estimated computation time in secs.

**make\_sample**(*N, sampling\_method='saltelli', sampling\_kwargs=None*)

Make samples of the input variables

For all input parameters, sample from their respective distributions using the chosen `sampling_method` from SALib. <https://salib.readthedocs.io/en/latest/api.html>

This sets the attributes `unc_output.samples_df`, `unc_output.sampling_method`, `unc_output.sampling_kwargs`.

**Parameters**

- **N** (*int*) – Number of samples as used in the sampling method from SALib
- **sampling\_method** (*str, optional*) – The sampling method as defined in SALib. Possible choices: 'saltelli', 'fast\_sampler', 'latin', 'morris', 'dgs', 'ff' <https://salib.readthedocs.io/en/latest/api.html> The default is 'saltelli'.
- **sampling\_kwargs** (*kwargs, optional*) – Optional keyword arguments passed on to the SALib `sampling_method`. The default is None.

**Returns** `unc_output` – Uncertainty data object with the samples

**Return type** `climada.engine.uncertainty.unc_output.UncOutput()`

**See also:**

**SALib.sample** sampling methods from SALib `SALib.sample`

**https** [//salib.readthedocs.io/en/latest/api.html](https://salib.readthedocs.io/en/latest/api.html)

**sensitivity**(*unc\_output, sensitivity\_method='sobol', sensitivity\_kwargs=None*)

Compute the sensitivity indices using SALib.

Prior to doing the sensitivity analysis, one must compute the uncertainty (distribution) of the output values (with `self.uncertainty()`) for all the samples (rows of `self.samples_df`).

According to Wikipedia, sensitivity analysis is “the study of how the uncertainty in the output of a mathematical model or system (numerical or otherwise) can be apportioned to different sources of uncertainty in its inputs.” The sensitivity of each input is often represented by a numeric value, called the sensitivity index. Sensitivity indices come in several forms.

This sets the attributes: `sens_output.sensitivity_method` `sens_output.sensitivity_kwargs` `sens_output.xxx_sens_df` for each metric `unc_output.xxx_unc_df`

**Parameters**

- **unc\_output** (*climada.engine.uncertainty.unc\_output.UncOutput()*) – Uncertainty data object in which to store the sensitivity indices
- **sensitivity\_method** (*str*) – sensitivity analysis method from SALib.analyse Possible choices:

‘fast’, ‘rbd\_fact’, ‘morris’, ‘sobol’, ‘delta’, ‘ff’

The default is ‘sobol’. Note that in Salib, sampling methods and sensitivity analysis methods should be used in specific pairs. <https://salib.readthedocs.io/en/latest/api.html>

- **sensitivity\_kwargs** (*dict()*, *optional*) – Keyword arguments of the chosen SALib analyse method. The default is to use SALib’s default arguments.

**Returns** **sens\_output** – Uncertainty data object with all the sensitivity indices, and all the uncertainty data copied over from **unc\_output**.

**Return type** `climada.engine.uncertainty.unc_output.UncOutput()`

### **climada.engine.uncertainty\_quantification.calc\_cost\_benefit module**

```
class climada.engine.uncertainty_quantification.calc_cost_benefit.CalcCostBenefit(haz_input_var,
ent_input_var,
haz_fut_input_var=None,
ent_fut_input_var=None)
```

Bases: `climada.engine.uncertainty_quantification.calc_base.Calc`

Cost Benefit uncertainty analysis class

This is the base class to perform uncertainty analysis on the outputs of `climada.engine.costbenefit.CostBenefit()`.

#### **metric\_names**

Names of the cost benefit output metris (‘tot\_climate\_risk’, ‘benefit’, ‘cost\_ben\_ratio’, ‘imp\_meas\_present’, ‘imp\_meas\_future’)

**Type** `tuple(str)`

#### **value\_unit**

Unit of the exposures value

**Type** `str`

#### **input\_var\_names**

Names of the required uncertainty variables (‘haz\_input\_var’, ‘ent\_input\_var’, ‘haz\_fut\_input\_var’, ‘ent\_fut\_input\_var’)

**Type** `tuple(str)`

#### **haz\_input\_var**

Present Hazard uncertainty variable

**Type** `climada.engine.uncertainty.input_var.InputVar`

#### **ent\_input\_var**

Present Entity uncertainty variable

**Type** `climada.engine.uncertainty.input_var.InputVar`

#### **haz\_unc\_fut\_Var**

Future Hazard uncertainty variable

**Type** `climada.engine.uncertainty.input_var.InputVar`

#### **ent\_fut\_input\_var**

Future Entity uncertainty variable

**Type** climada.engine.uncertainty.input\_var.InputVar

**\_\_init\_\_**(*haz\_input\_var, ent\_input\_var, haz\_fut\_input\_var=None, ent\_fut\_input\_var=None*)  
Initialize UncCalcCostBenefit

Sets the uncertainty input variables, the cost benefit metric\_names, and the units.

#### Parameters

- **haz\_input\_var** (*climada.engine.uncertainty.input\_var.InputVar*) – or climada.hazard.Hazard Hazard uncertainty variable or Hazard for the present Hazard in climada.engine.CostBenefit.calc
- **ent\_input\_var** (*climada.engine.uncertainty.input\_var.InputVar*) – or climada.entity.Entity Entity uncertainty variable or Entity for the present Entity in climada.engine.CostBenefit.calc
- **haz\_fut\_input\_var** (*climada.engine.uncertainty.input\_var.InputVar*) – or climada.hazard.Hazard, optional Hazard uncertainty variable or Hazard for the future Hazard The Default is None.
- **ent\_fut\_input\_var** (*climada.engine.uncertainty.input\_var.InputVar*) – or climada.entity.Entity, optional Entity uncertainty variable or Entity for the future Entity in climada.engine.CostBenefit.calc

**uncertainty**(*unc\_data, pool=None, \*\*cost\_benefit\_kwargs*)

Computes the cost benefit for each sample in unc\_output.sample\_df.

By default, imp\_meas\_present, imp\_meas\_future, tot\_climate\_risk, benefit, cost\_ben\_ratio are computed.

This sets the attributes: unc\_output.imp\_meas\_present\_unc\_df, unc\_output.imp\_meas\_future\_unc\_df, unc\_output.tot\_climate\_risk\_unc\_df, unc\_output.benefit\_unc\_df, unc\_output.cost\_ben\_ratio\_unc\_df, unc\_output.unit, unc\_output.cost\_benefit\_kwargs

#### Parameters

- **unc\_data** (*climada.engine.uncertainty.unc\_output.UncOutput*) – Uncertainty data object with the input parameters samples
- **pool** (*pathos.pools.ProcessPool, optional*) – Pool of CPUs for parralel computations. Default is None. The default is None.
- **cost\_benefit\_kwargs** (*keyword arguments*) – Keyword arguments passed on to climada.engine.CostBenefit.calc()

**Returns** **unc\_output** – Uncertainty data object in with the cost benefit outputs for each sample and all the sample data copied over from unc\_sample.

**Return type** climada.engine.uncertainty.unc\_output.UncCostBenefitOutput

**Raises** **ValueError**: – If no sampling parameters defined, the uncertainty distribution cannot be computed.

**See also:**

[\*climada.engine.cost\\_benefit\*](#) Compute risk and adptation option cost benefits.

**climada.engine.uncertainty\_quantification.calc\_impact module**

**class** climada.engine.uncertainty\_quantification.calc\_impact.**CalcImpact**(*exp\_input\_var*,  
*impf\_input\_var*,  
*haz\_input\_var*)

Bases: *climada.engine.uncertainty\_quantification.calc\_base.Calc*

Impact uncertainty calculation class.

This is the class to perform uncertainty analysis on the outputs of a climada.engine.impact.Impact() object.

**rp**

List of the chosen return periods.

**Type** list(int)

**calc\_eai\_exp**

Compute eai\_exp or not

**Type** bool

**calc\_at\_event**

Compute eai\_exp or not

**Type** bool

**metric\_names**

Names of the impact output metris ('aai\_agg', 'freq\_curve', 'at\_event', 'eai\_exp', 'tot\_value')

**Type** tuple(str)

**value\_unit**

Unit of the exposures value

**Type** str

**input\_var\_names**

Names of the required uncertainty input variables ('exp\_input\_var', 'impf\_input\_var', 'haz\_input\_var')

**Type** tuple(str)

**exp\_input\_var**

Exposure uncertainty variable

**Type** climada.engine.uncertainty.input\_var.InputVar

**impf\_input\_var**

Impact function set uncertainty variable

**Type** climada.engine.uncertainty.input\_var.InputVar

**haz\_input\_var**

Hazard uncertainty variable

**Type** climada.engine.uncertainty.input\_var.InputVar

**\_\_init\_\_**(*exp\_input\_var*, *impf\_input\_var*, *haz\_input\_var*)

Initialize UncCalcImpact

Sets the uncertainty input variables, the impact metric\_names, and the units.

**Parameters**

- **exp\_input\_var** (*climada.engine.uncertainty.input\_var.InputVar* or *climada.entity.Exposure*) – Exposure uncertainty variable or Exposure

- **impf\_input\_var** (*climada.engine.uncertainty.input\_var.InputVar* or *climada.entity.ImpactFuncSet*) – Impact function set uncertainty variable or Impact function set
- **haz\_input\_var** (*climada.engine.uncertainty.input\_var.InputVar* or *climada.hazard.Hazard*) – Hazard uncertainty variable or Hazard

**uncertainty**(*unc\_sample*, *rp=None*, *calc\_eai\_exp=False*, *calc\_at\_event=False*, *pool=None*)

Computes the impact for each sample in *unc\_data.sample\_df*.

By default, the aggregated average annual impact (*impact.aai\_agg*) and the exceeds impact at return periods *rp* (*impact.calc\_freq\_curve(self.rp).impact*) is computed. Optionally, *eai\_exp* and *at\_event* is computed (this may require a larger amount of memory if the number of samples and/or the number of centroids and/or exposures points is large).

This sets the attributes *self.rp*, *self.calc\_eai\_exp*, *self.calc\_at\_event*, *self.metrics*.

This sets the attributes: *unc\_output.aai\_agg\_unc\_df*, *unc\_output.freq\_curve\_unc\_df*, *unc\_output.eai\_exp\_unc\_df*, *unc\_output.at\_event\_unc\_df*, *unc\_output.tot\_value\_unc\_df*, *unc\_output.unit*

#### Parameters

- **unc\_sample** (*climada.engine.uncertainty.unc\_output.UncOutput*) – Uncertainty data object with the input parameters samples
- **rp** (*list(int)*, *optional*) – Return periods in years to be computed. The default is [5, 10, 20, 50, 100, 250].
- **calc\_eai\_exp** (*boolean*, *optional*) – Toggle computation of the impact at each centroid location. The default is False.
- **calc\_at\_event** (*boolean*, *optional*) – Toggle computation of the impact for each event. The default is False.
- **pool** (*pathos.pools.ProcessPool*, *optional*) – Pool of CPUs for parallel computations. The default is None.

**Returns** **unc\_output** – Uncertainty data object with the impact outputs for each sample and all the sample data copied over from *unc\_sample*.

**Return type** *climada.engine.uncertainty.unc\_output.UncImpactOutput*

**Raises** **ValueError**: – If no sampling parameters defined, the distribution cannot be computed.

**See also:**

*climada.engine.impact* Compute risk.

### climada.engine.uncertainty\_quantification.input\_var module

**class** *climada.engine.uncertainty\_quantification.input\_var.InputVar*(*func*, *distr\_dict*)

Bases: object

Input variable for the uncertainty analysis

An uncertainty input variable requires a single or multi-parameter function. The parameters must follow a given distribution. The uncertainty input variables are the input parameters of the model.

#### **distr\_dict**

Distribution of the uncertainty parameters. Keys are uncertainty parameters names and Values are probability density distribution from the *scipy.stats* package <https://docs.scipy.org/doc/scipy/reference/stats.html>



**Type** dict

### labels

Names of the uncertainty parameters (keys of distr\_dict)

**Type** list

### func

User defined python function with the uncertainty parameters as keyword arguments and which returns a climada object.

**Type** function

## Notes

A few default Variables are defined for Hazards, Exposures, Impact Functions, Measures and Entities.

## Examples

**Categorical variable function: LitPop exposures with m,n exponents in [0,5]** import scipy as sp def litpop\_cat(m, n):

```
exp = Litpop() exp.set_country('CHE', exponent=[m, n]) return exp
```

```
distr_dict = { 'm': sp.stats.randint(low=0, high=5), 'n': sp.stats.randint(low=0, high=5) }
```

```
iv_cat = InputVar(func=litpop_cat, distr_dict=distr_dict)
```

**Continuous variable function: Impact function for TC** import scipy as sp def imp\_fun\_tc(G, v\_half, vmin, k, \_id=1):

```
imp_fun = ImpactFunc() imp_fun.haz_type = 'TC' imp_fun.id = _id imp_fun.intensity_unit = 'm/s' imp_fun.intensity = np.linspace(0, 150, num=100) imp_fun.mdd = np.repeat(1, len(imp_fun.intensity)) imp_fun.paa = np.array([sigmoid_function(v, G, v_half, vmin, k)
```

```
for v in imp_fun.intensity])
```

```
imp_fun.check() impf_set = ImpactFuncSet() impf_set.append(imp_fun) return impf_set
```

```
distr_dict = {"G": sp.stats.uniform(0.8, 1), "v_half": sp.stats.uniform(50, 100), "vmin": sp.stats.norm(loc=15, scale=30), "k": sp.stats.randint(low=1, high=9) }
```

```
iv_cont = InputVar(func=imp_fun_tc, distr_dict=distr_dict)
```

**\_\_init\_\_**(func, distr\_dict)

Initialize InputVar

### Parameters

- **func** (function) – Variable defined as a function of the uncertainty parameters
- **distr\_dict** (dict) – Dictionary of the probability density distributions of the uncertainty parameters, with keys matching the keyword arguments (i.e. uncertainty parameters) of the func function. The distribution must be of type scipy.stats <https://docs.scipy.org/doc/scipy/reference/stats.html>

**evaluate**(\*\*params)

Return the value of uncertainty input variable.

By default, the value of the average is returned.

**Parameters** **\*\*params** (*optional*) – Input parameters will be passed to self.InputVar\_func.

**Returns** **unc\_func(\*\*params)** – Output of the uncertainty variable.

**Return type** climada object

**plot**(*figsize=None*)

Plot the distributions of the parameters of the uncertainty variable.

**Parameters** **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3) figsize = (ncols \* FIG\_W, nrows \* FIG\_H)

**Returns** **axes** – The figure and axes handle of the plot.

**Return type** matplotlib.pyplot.figure, matplotlib.pyplot.axes

**static** **var\_to\_inputvar**(*var*)

Returns an uncertainty variable with no distribution if var is not an InputVar. Else, returns var.

**Parameters** **var** (*climada.uncertainty.InputVar or any other CLIMADA object*)

**Returns** var if var is InputVar, else InputVar with var and no distribution.

**Return type** *InputVar*

**static** **haz**(*haz, n\_ev=None, bounds\_int=None, bounds\_freq=None*)

Helper wrapper for basic hazard uncertainty input variable

The following types of uncertainties can be added: HE: sub-sampling events from the total event set

For each sub-sample, n\_ev events are sampled with replacement. HE is the value of the seed for the uniform random number generator.

**HI: scale the intensity of all events (homogeneously)** The intensity of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds\_int

**HF: scale the frequency of all events (homogeneously)** The frequency of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds\_freq

If a bounds is None, this parameter is assumed to have no uncertainty.

**Parameters**

- **haz** (*climada.hazard.Hazard*) – The base hazard
- **n\_ev** (*int, optional*) – Number of events to be subsampled per sample. Can be equal or larger than haz.size. The default is None.
- **bounds\_int** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous intensity scaling. The default is None.
- **bounds\_freq** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous frequency scaling. The default is None.

**Returns** Uncertainty input variable for a hazard object.

**Return type** *climada.engine.uncertainty\_quantification.input\_var.InputVar*

**static** **exp**(*exp, bounds\_totval=None, bounds\_noise=None*)

Helper wrapper for basic exposure uncertainty input variable

The following types of uncertainties can be added: ET: scale the total value (homogeneously)

The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds\_totvalue

**EN: multiplicative noise (inhomogeneous)** The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds\_noise. EN is the value of the seed for the uniform random number generator.

If a bounds is None, this parameter is assumed to have no uncertainty.

#### Parameters

- **exp** (*climada.entity.exposures.Exposures*) – The base exposure.
- **bounds\_totval** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous total value scaling. The default is None.
- **bounds\_noise** ((*float, float*), *optional*) – Bounds of the uniform distribution to scale each exposure point independently. The default is None.

**Returns** Uncertainty input variable for an exposure object.

**Return type** *climada.engine.uncertainty\_quantification.input\_var.InputVar*

**static impfset**(*impf\_set, haz\_id\_dict=None, bounds\_mdd=None, bounds\_paa=None, bounds\_impfi=None*)

Helper wrapper for basic impact function set uncertainty input variable.

One impact function (chosen with haz\_type and fun\_id) is characterized.

The following types of uncertainties can be added: MDD: scale the mdd (homogeneously)

The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds\_mdd

**PAA: scale the paa (homogeneously)** The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds\_paa

**IFi: shift the intensity (homogeneously)** The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds\_int

If a bounds is None, this parameter is assumed to have no uncertainty.

#### Parameters

- **impf\_set** (*climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet*) – The base impact function set.
- **bounds\_mdd** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous mdd scaling. The default is None.
- **bounds\_paa** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous paa scaling. The default is None.
- **bounds\_impfi** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous shift of intensity. The default is None.
- **haz\_id\_dict** (*dict()*, *optional*) – Dictionary of the impact functions affected by uncertainty. Keys are hazard types (str), values are a list of impact function id (int). The default is { 'TC': [1] }

**Returns** DESCRIPTION.

**Return type** *climada.engine.uncertainty\_quantification.input\_var.InputVar*

```
static ent(impf_set, disc_rate, exp, meas_set, haz_id_dict=None, bounds_disc=None, bounds_cost=None,
bounds_totval=None, bounds_noise=None, bounds_mdd=None, bounds_paa=None,
bounds_impfi=None)
```

Helper wrapper for basic entity set uncertainty input variable.

Important: only the impact function defined by `haz_type` and `fun_id` will be affected by `bounds_impfi`, `bounds_mdd`, `bounds_paa`.

The following types of uncertainties can be added: DR: value of constant discount rate (homogeneously)

The value of the discounts in each year is sampled uniformly from a distribution with (min, max) = `bounds_disc`

**CO: scale the cost (homogeneously)** The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = `bounds_cost`

**ET: scale the total value (homogeneously)** The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = `bounds_totval`

**EN: mutliplicative noise (inhomogeneous)** The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = `bounds_noise`. EN is the value of the seed for the uniform random number generator.

**MDD: scale the mdd (homogeneously)** The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = `bounds_mdd`

**PAA: scale the paa (homogeneously)** The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = `bounds_paa`

**IFi: shift the intensity (homogeneously)** The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = `bounds_int`

If a bounds is None, this parameter is assumed to have no uncertainty.

#### Parameters

- **bounds\_disk** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous discount rate scaling. The default is None.
- **bounds\_cost** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous cost of all measures scaling. The default is None.
- **bounds\_totval** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous total exposure value scaling. The default is None.
- **bounds\_noise** ((*float, float*), *optional*) – Bounds of the uniform distribution to scale each exposure point independently. The default is None.
- **bounds\_mdd** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous mdd scaling. The default is None.
- **bounds\_paa** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous paa scaling. The default is None.
- **bounds\_int** ((*float, float*), *optional*) – Bounds of the uniform distribution for the homogeneous shift of intensity. The default is None.
- **impf\_set** (*climada.engine.impact\_funcs.impact\_func\_set.ImpactFuncSet*) – The base impact function set.
- **disc\_rate** (*climada.entity.disc\_rates.base.DiscRates*) – The base discount rates.
- **exp** (*climada.entity.exposures.base.Exposure*) – The base exposure.

- **meas\_set** (*climada.entity.measures.measure\_set.MeasureSet*) – The base measures.
- **haz\_id\_dict** (*dict()*, *optional*) – Dictionary of the impact functions affected by uncertainty. Keys are hazard types (str), values are a list of impact function id (int). The default is { 'TC': [1] }

**Returns** Entity uncertainty input variable

**Return type** *climada.engine.uncertainty\_quantification.input\_var.InputVar*

**static entfut**(*impf\_set, exp, meas\_set, haz\_id\_dict=None, bounds\_cost=None, bounds\_eg=None, bounds\_noise=None, bounds\_impfi=None, bounds\_mdd=None, bounds\_paa=None*)

Helper wrapper for basic future entity set uncertainty input variable.

Important: only the impact function defined by *haz\_type* and *fun\_id* will be affected by *bounds\_impfi*, *bounds\_mdd*, *bounds\_paa*.

The following types of uncertainties can be added: CO: scale the cost (homogeneously)

The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = *bounds\_cost*

**EG: scale the exposures growth (homogeneously)** The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = *bounds\_eg*

**EN: mutliplicative noise (inhomogeneous)** The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = *bounds\_noise*. EN is the value of the seed for the uniform random number generator.

**MDD: scale the mdd (homogeneously)** The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = *bounds\_mdd*

**PAA: scale the paa (homogeneously)** The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = *bounds\_paa*

**IFi: shift the impact function intensity (homogeneously)** The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = *bounds\_impfi*

If a bounds is None, this parameter is assumed to have no uncertainty.

#### Parameters

- **bounds\_cost** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous cost of all measures scaling. The default is None.
- **bounds\_eg** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous total exposure growth scaling. The default is None.
- **bounds\_noise** (*(float, float), optional*) – Bounds of the uniform distribution to scale each exposure point independently. The default is None.
- **bounds\_mdd** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous mdd scaling. The default is None.
- **bounds\_paa** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous paa scaling. The default is None.
- **bounds\_impfi** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous shift of intensity. The default is None.
- **impf\_set** (*climada.engine.impact\_funcs.impact\_func\_set.ImpactFuncSet*) – The base impact function set.
- **exp** (*climada.entity.exposures.base.Exposure*) – The base exposure.

- **meas\_set** (*climada.entity.measures.measure\_set.MeasureSet*) – The base measures.
- **haz\_id\_dict** (*dict(), optional*) – Dictionary of the impact functions affected by uncertainty. Keys are hazard types (str), values are a list of impact function id (int). The default is { 'TC': [1] }

**Returns** Entity uncertainty input variable

**Return type** *climada.engine.uncertainty\_quantification.input\_var.InputVar*

### **climada.engine.uncertainty\_quantification.unc\_output module**

**class** *climada.engine.uncertainty\_quantification.unc\_output.UncOutput*(*samples\_df, unit=None*)  
Bases: object

Class to store and plot uncertainty and sensitivity analysis output data

This is the base class to store uncertainty and sensitivity outputs of an analysis done on *climada.engine.impact.Impact()* or *climada.engine.costbenefit.CostBenefit()* object.

**samples\_df**

Values of the sampled uncertainty parameters. It has *n\_samples* rows and one column per uncertainty parameter.

**Type** *pandas.DataFrame*

**sampling\_method**

Name of the sampling method from SALib. <https://salib.readthedocs.io/en/latest/api.html#>

**Type** str

**n\_samples**

Effective number of samples (number of rows of *samples\_df*)

**Type** int

**param\_labels**

Name of all the uncertainty parameters

**Type** list

**distr\_dict**

Comon flattened dictionary of all the *distr\_dict* of all input variables. It represents the distribution of all the uncertainty parameters.

**Type** dict

**problem\_sa**

The description of the uncertainty variables and their distribution as used in SALib. <https://salib.readthedocs.io/en/latest/basics.html>.

**Type** dict

**\_\_init\_\_**(*samples\_df, unit=None*)

Initialize Uncertainty Data object.

**Parameters**

- **samples\_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str, optional*) – value unit

**get\_samples\_df()**

**get\_unc\_df**(*metric\_name*)

**set\_unc\_df**(*metric\_name*, *unc\_df*)

**get\_sens\_df**(*metric\_name*)

**set\_sens\_df**(*metric\_name*, *sens\_df*)

**check\_salib**(*sensitivity\_method*)

Checks whether the chosen sensitivity method and the sampling method used to generate self.samples\_df respect the pairing recommendation by the SALib package.

<https://salib.readthedocs.io/en/latest/api.html>

**Parameters** *sensitivity\_method* (*str*) – Name of the sensitivity analysis method.

**Returns** True if sampling and sensitivity methods respect the recommended pairing.

**Return type** bool

**property sampling\_method**

Returns the sampling method used to generate self.samples\_df

**Returns** Sampling method name

**Return type** str

**property sampling\_kwargs**

Returns the kwargs of the sampling method that generate self.samples\_df

**Returns** Dictionary of arguments for SALib sampling method

**Return type** dict

**property n\_samples**

The effective number of samples

**Returns** effective number of samples

**Return type** int

**property param\_labels**

Labels of all uncertainty input parameters.

**Returns** Labels of all uncertainty input parameters.

**Return type** list of str

**property problem\_sa**

The description of the uncertainty variables and their distribution as used in SALib. <https://salib.readthedocs.io/en/latest/basics.html>

**Returns** Salib problem dictionary.

**Return type** dict

**property uncertainty\_metrics**

Retrieve all uncertainty output metrics names

**Returns** *unc\_metric\_list* – List of names of attributes containing metrics uncertainty values, without the trailing ‘\_unc\_df’

**Return type** [str]

**property sensitivity\_metrics**

Retrieve all sensitivity output metrics names

**Returns** `sens_metric_list` – List of names of attributes containing metrics sensitivity values, without the trailing ‘\_sens\_df’

**Return type** [str]

**get\_uncertainty**(*metric\_list=None*)

Returns uncertainty dataframe with values for each sample

**Parameters** `metric_list` ([str], optional) – List of uncertainty metrics to consider. The default returns all uncertainty metrics at once.

**Returns** Joint dataframe of all uncertainty values for all metrics in the `metric_list`.

**Return type** pandas.DataFrame

**See also:**

`uncertainty_metrics` list of all available uncertainty metrics

**get\_sensitivity**(*salib\_si, metric\_list=None*)

Returns sensitivity index

E.g. For the sensitivity analysis method ‘sobol’, the choices are [‘S1’, ‘ST’], for ‘delta’ the choices are [‘delta’, ‘S1’].

For more information see the SALib documentation: <https://salib.readthedocs.io/en/latest/basics.html>

**Parameters**

- `salib_si` (str) – Sensitivity index
- `metric_list` ([str], optional) – List of sensitivity metrics to consider. The default returns all sensitivity indices at once.

**Returns** Joint dataframe of the sensitivity indices for all metrics in the `metric_list`

**Return type** pandas.DataFrame

**See also:**

`sensitivity_metrics` list of all available sensitivity metrics

**get\_largest\_si**(*salib\_si, metric\_list=None*)

Get largest si per metric

**Parameters**

- `salib_si` (str) – The name of the sensitivity index to plot.
- `metric_list` (list of strings, optional) – List of metrics to plot the sensitivity. Default is None.

**Returns** `max_si_df` – Dataframe with the largest si and its value per metric

**Return type** pandas.dataframe

**plot\_sample**(*figsize=None*)

Plot the sample distributions of the uncertainty input parameters

For each uncertainty input variable, the sample distributions is shown in a separate axes.

**Parameters** `figsize` (tuple(int or float, int or float), optional) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:



```
nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3)
figsize = (ncols * FIG_W, nrows * FIG_H)
```

**Raises** **ValueError** – If no sample was computed the plot cannot be made.

**Returns** **axes** – The axis handle of the plot.

**Return type** matplotlib.pyplot.axes

**plot\_uncertainty**(*metric\_list=None, orig\_list=None, figsize=None, log=False, axes=None*)

Plot the uncertainty distribution

For each risk metric, a separate axes is used to plot the uncertainty distribution of the output values obtained over the sampled input parameters.

#### Parameters

- **metric\_list** (*list[str], optional*) – List of metrics to plot the distribution. The default is None.
- **orig\_list** (*list[float], optional*) – List of the original (without uncertainty) values for each sub-metric of the metrics in metric\_list. The ordering is identical. The default is None.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as: `nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3) figsize = (ncols * FIG_W, nrows * FIG_H)`
- **log** (*boolean, optional*) – Use log10 scale for x axis. Default is False.
- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.

**Raises** **ValueError** – If no metric distribution was computed the plot cannot be made.

**Returns** **axes** – The axes handle of the plot.

**Return type** matplotlib.pyplot.axes

See also:

[\*\*uncertainty\\_metrics\*\*](#) list of all available uncertainty metrics

**plot\_rp\_uncertainty**(*figsize=(16, 6), axes=None*)

Plot the distribution of return period uncertainty

#### Parameters

- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is (8, 6)
- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.

**Raises** **ValueError** – If no metric distribution was computed the plot cannot be made.

**Returns** **ax** – The axis handle of the plot.

**Return type** matplotlib.pyplot.axes

**plot\_sensitivity**(*salib\_si='SI', salib\_si\_conf='SI\_conf', metric\_list=None, figsize=None, axes=None, \*\*kwargs*)

Bar plot of a first order sensitivity index

For each metric, the sensitivity indices are plotted in a separate axes.

This requires that a sensitivity analysis was already performed.

E.g. For the sensitivity analysis method ‘sobel’, the choices are [‘S1’, ‘ST’], for ‘delta’ the choices are [‘delta’, ‘S1’].

Note that not all sensitivity indices have a confidence interval.

For more information see the SALib documentation: <https://salib.readthedocs.io/en/latest/basics.html>

#### Parameters

- **salib\_si** (*string, optional*) – The first order (one value per metric output) sensitivity index to plot. The default is S1.
- **salib\_si\_conf** (*string, optional*) – The confidence value for the first order sensitivity index to plot. The default is S1\_conf.
- **metric\_list** (*list of strings, optional*) – List of metrics to plot the sensitivity. If a metric is not found it is ignored.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:  
nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3) figsize = (ncols \* FIG\_W, nrows \* FIG\_H)
- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **kwargs** – Keyword arguments passed on to pandas.DataFrame.plot(kind=‘bar’)

**Raises ValueError** : – If no sensitivity is available the plot cannot be made.

**Returns axes** – The axes handle of the plot.

**Return type** matplotlib.pyplot.axes

**See also:**

**sensitivity\_metrics** list of all available sensitivity metrics

**plot\_sensitivity\_second\_order**(salib\_si=‘S2’, salib\_si\_conf=‘S2\_conf’, metric\_list=None, figsize=None, axes=None, \*\*kwargs)

Plot second order sensitivity indices as matrix.

For each metric, the sensitivity indices are plotted in a separate axes.

E.g. For the sensitivity analysis method ‘sobel’, the choices are [‘S2’, ‘S2\_conf’].

Note that not all sensitivity indices have a confidence interval.

For more information see the SALib documentation: <https://salib.readthedocs.io/en/latest/basics.html>

#### Parameters

- **salib\_si** (*string, optional*) – The second order sensitivity index to plot. The default is S2.
- **salib\_si\_conf** (*string, optional*) – The confidence value for the sensitivity index salib\_si to plot. The default is S2\_conf.
- **metric\_list** (*list of strings, optional*) – List of metrics to plot the sensitivity. If a metric is not found it is ignored. Default is all 1D metrics.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3) figsize = (ncols \* 5, nrows \* 5)

- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **kwargs** – Keyword arguments passed on to matplotlib.pyplot.imshow()

**Raises ValueError** : – If no sensitivity is available the plot cannot be made.

**Returns axes** – The axes handle of the plot.

**Return type** matplotlib.pyplot.axes

**See also:**

**sensitivity\_metrics** list of all available sensitivity metrics

**plot\_sensitivity\_map**(*exp, salib\_si='S1', \*\*kwargs*)

Plot a map of the largest sensitivity index in each exposure point

Requires the uncertainty distribution for eai\_exp.

**Parameters**

- **exp** (*climada.exposure*) – The exposure from which to take the coordinates.
- **salib\_si** (*str, optional*) – The name of the sensitivity index to plot. The default is 'S1'.
- **kwargs** – Keyword arguments passed on to climada.util.plot.geo\_scatter\_categorical

**Raises ValueError** : – If no sensitivity data is found, raise error.

**Returns ax** – The axis handle of the plot.

**Return type** matplotlib.pyplot.axes

**See also:**

**climada.util.plot.geo\_scatter\_categorical** geographical plot for categorical variable

**to\_hdf5**(*filename=None*)

Save output to .hdf5

**Parameters filename** (*str or pathlib.Path, optional*) – The filename with absolute or relative path. The default name is “unc\_output + datetime.now() + .hdf5” and the default path is taken from climada.config

**Returns save\_path** – Path to the saved file

**Return type** pathlib.Path

**static from\_hdf5**(*filename*)

Load a uncertainty and uncertainty output data from .hdf5 file

**Parameters filename** (*str or pathlib.Path*) – The filename with absolute or relative path.

**Returns unc\_output** – Uncertainty and sensitivity data loaded from .hdf5 file.

**Return type** climada.engine.uncertainty.unc\_output.UncOutput

```
class climada.engine.uncertainty_quantification.unc_output.UncCostBenefitOutput(samples_df,
 unit,
 imp_meas_present_unc_df,
 imp_meas_future_unc_df,
 tot_climate_risk_unc_df,
 bene-
 fit_unc_df,
 cost_ben_ratio_unc_df,
 cost_benefit_kwargs)
```

Bases: [climada.engine.uncertainty\\_quantification.unc\\_output.UncOutput](#)

Extension of UncOutput specific for CalcCostBenefit, returned by the uncertainty() method.

```
__init__(samples_df, unit, imp_meas_present_unc_df, imp_meas_future_unc_df, tot_climate_risk_unc_df,
 benefit_unc_df, cost_ben_ratio_unc_df, cost_benefit_kwargs)
```

Constructor

Uncertainty output values from cost\_benefit.calc for each sample

#### Parameters

- **samples\_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str*) – value unit
- **imp\_meas\_present\_unc\_df** (*pandas.DataFrame*) – Each row contains the values of `imp_meas_present` for one sample (row of `samples_df`)
- **imp\_meas\_future\_unc\_df** (*pandas.DataFrame*) – Each row contains the values of `imp_meas_future` for one sample (row of `samples_df`)
- **tot\_climate\_risk\_unc\_df** (*pandas.DataFrame*) – Each row contains the values of `tot_climate_risk` for one sample (row of `samples_df`)
- **benefit\_unc\_df** (*pandas.DataFrame*) – Each row contains the values of `benefit` for one sample (row of `samples_df`)
- **cost\_ben\_ratio\_unc\_df** (*pandas.DataFrame*) – Each row contains the values of `cost_ben_ratio` for one sample (row of `samples_df`)
- **cost\_benefit\_kwargs** (*pandas.DataFrame*) – Each row contains the value of `cost_benefit` for one sample (row of `samples_df`)

```
class climada.engine.uncertainty_quantification.unc_output.UncImpactOutput(samples_df, unit,
 aai_agg_unc_df,
 freq_curve_unc_df,
 eai_exp_unc_df,
 at_event_unc_df,
 tot_value_unc_df)
```

Bases: [climada.engine.uncertainty\\_quantification.unc\\_output.UncOutput](#)

Extension of UncOutput specific for CalcImpact, returned by the uncertainty() method.

```
__init__(samples_df, unit, aai_agg_unc_df, freq_curve_unc_df, eai_exp_unc_df, at_event_unc_df,
 tot_value_unc_df)
```

Constructor

Uncertainty output values from impact.calc for each sample

#### Parameters

- **samples\_df** (*pandas.DataFrame*) – input parameters samples

- **unit** (*str*) – value unit
- **aai\_agg\_unc\_df** (*pandas.DataFrame*) – Each row contains the value of aai\_aag for one sample (row of samples\_df)
- **freq\_curve\_unc\_df** (*pandas.DataFrame*) – Each row contains the values of the impact exceedence frequency curve for one sample (row of samples\_df)
- **eai\_exp\_unc\_df** (*pandas.DataFrame*) – Each row contains the values of eai\_exp for one sample (row of samples\_df)
- **at\_event\_unc\_df** (*pandas.DataFrame*) – Each row contains the values of at\_event for one sample (row of samples\_df)
- **tot\_value\_unc\_df** (*pandas.DataFrame*) – Each row contains the value of tot\_value for one sample (row of samples\_df)

### climada.engine.calibration\_opt module

`climada.engine.calibration_opt.calib_instance(hazard, exposure, impact_func, df_out=Empty DataFrame Columns: [] Index: [], yearly_impact=False, return_cost=False)`

calculate one impact instance for the calibration algorithm and write to given DataFrame

#### Parameters

- **hazard** (*Hazard*)
- **exposure** (*Exposure*)
- **impact\_func** (*ImpactFunc*)
- **df\_out** (*Dataframe, optional*) – Output DataFrame with headers of columns defined and optionally with first row (index=0) defined with values. If columns “impact”, “event\_id”, or “year” are not included, they are created here. Data like reported impacts or impact function parameters can be given here; values are preserved.
- **yearly\_impact** (*boolean, optional*) – if set True, impact is returned per year, not per event
- **return\_cost** (*str, optional*) – if not ‘False’ but any of ‘R2’, ‘logR2’, cost is returned instead of df\_out

**Returns** **df\_out** – DataFrame with modelled impact written to rows for each year or event.

**Return type** DataFrame

`climada.engine.calibration_opt.init_impf(impf_name_or_instance, param_dict, df_out=Empty DataFrame Columns: [] Index: [0])`

create an ImpactFunc based on the parameters in param\_dict using the method specified in impf\_parameterisation\_name and document it in df\_out.

#### Parameters

- **impf\_name\_or\_instance** (*str or ImpactFunc*) – method of impact function parameterisation e.g. ‘emanuel’ or an instance of ImpactFunc
- **param\_dict** (*dict, optional*) – dict of parameter\_names and values e.g. {‘v\_thresh’: 25.7, ‘v\_half’: 70, ‘scale’: 1} or {‘mdd\_shift’: 1.05, ‘mdd\_scale’: 0.8, ‘paa\_shift’: 1, ‘paa\_scale’: 1}

#### Returns

- **imp\_fun** (*ImpactFunc*) – The Impact function based on the parameterisation

- **df\_out** (*DataFrame*) – Output DataFrame with headers of columns defined and with first row (index=0) defined with values. The impact function parameters from param\_dict are represented here.

`climada.engine.calibration_opt.change_impf`(*impf\_instance*, *param\_dict*)

apply a shifting or a scaling defined in param\_dict to the impact function in impf\_instance and return it as a new ImpactFunc object.

#### Parameters

- **impf\_instance** (*ImpactFunc*) – an instance of ImpactFunc
- **param\_dict** (*dict*) – dict of parameter\_names and values (interpreted as factors, 1 = neutral) e.g. {'mdd\_shift': 1.05, 'mdd\_scale': 0.8, 'paa\_shift': 1, 'paa\_scale': 1}

#### Returns ImpactFunc

**Return type** The Impact function based on the parameterisation

`climada.engine.calibration_opt.init_impact_data`(*hazard\_type*, *region\_ids*, *year\_range*, *source\_file*,  
*reference\_year*, *impact\_data\_source*='emdat',  
*yearly\_impact*=True)

creates a dataframe containing the recorded impact data for one hazard type and one area (countries, country or local split)

#### Parameters

- **hazard\_type** (*str*) – default = 'TC', type of hazard 'WS','FL' etc.
- **region\_ids** (*str*) – name the region\_ids or country names
- **year\_range** (*list*) – list containing start and end year. e.g. [1980, 2017]
- **source\_file** (*str*)
- **reference\_year** (*int*) – impacts will be scaled to this year
- **impact\_data\_source** (*str*, *optional*) – default 'emdat', others maybe possible
- **yearly\_impact** (*bool*, *optional*) – if set True, impact is returned per year, not per event

**Returns df\_out** – DataFrame with recorded impact written to rows for each year or event.

**Return type** pd.DataFrame

`climada.engine.calibration_opt.calib_cost_calc`(*df\_out*, *cost\_function*)

calculate the cost function of the modelled impact impact\_CLIMADA and the reported impact impact\_scaled in df\_out

#### Parameters

- **df\_out** (*pd.DataFrame*) – DataFrame as created in calib\_instance
- **cost\_function** (*str*) – chooses the cost function e.g. 'R2' or 'logR2'

**Returns cost** – The results of the cost function when comparing modelled and reported impact

**Return type** float

`climada.engine.calibration_opt.calib_all`(*hazard*, *exposure*, *impf\_name\_or\_instance*, *param\_full\_dict*,  
*impact\_data\_source*, *year\_range*, *yearly\_impact*=True)

portrait the difference between modelled and reported impacts for all impact functions described in param\_full\_dict and impf\_name\_or\_instance :Parameters: \* **hazard** (*list or Hazard*)

- **exposure** (*list or Exposures*) – list or instance of exposure of full countries
- **impf\_name\_or\_instance** (*string or ImpactFunc*) – the name of a parameterisation or an instance of class ImpactFunc e.g. ‘emanuel’
- **param\_full\_dict** (*dict*) – a dict containing keys used for f\_name\_or\_instance and values which are iterable (lists) e.g. {‘v\_thresh’: [25.7, 20], ‘v\_half’: [70], ‘scale’: [1, 0.8]}
- **impact\_data\_source** (*dict or pd.DataFrame*) – with name of impact data source and file location or dataframe
- **year\_range** (*list*)
- **yearly\_impact** (*bool, optional*)

**Returns** **df\_result** – df with modelled impact written to rows for each year or event.

**Return type** pd.DataFrame

```
climada.engine.calibration_opt.calib_optimize(hazard, exposure, Impf_name_or_instance, param_dict,
 impact_data_source, year_range, yearly_impact=True,
 cost_fucntion='R2', show_details=False)
```

portrait the difference between modelled and reported impacts for all impact functions described in param\_full\_dict and Impf\_name\_or\_instance

#### Parameters

- **hazard** (*list or Hazard*)
- **exposure** (*list or Exposures*) – list or instance of exposure of full countries
- **impf\_name\_or\_instance** (*string or ImpactFunc*) – the name of a parameterisation or an instance of class ImpactFunc e.g. ‘emanuel’
- **param\_dict** (*dict*) – a dict containing keys used for Impf\_name\_or\_instance and one set of values e.g. {‘v\_thresh’: 25.7, ‘v\_half’: 70, ‘scale’: 1}
- **impact\_data\_source** (*dict or pd. dataframe*) – with name of impact data source and file location or dataframe
- **year\_range** (*list*)
- **yearly\_impact** (*bool, optional*)
- **cost\_function** (*str, optional*) – the argument for function calib\_cost\_calc, default ‘R2’
- **show\_details** (*bool, optional*) – if True, return a tuple with the parameters AND the details of the optimization like success, status, number of iterations etc

**Returns** **param\_dict\_result** – the parameters with the best calibration results (or a tuple with (1) the parameters and (2) the optimization output)

**Return type** dict or tuple

**climada.engine.cost\_benefit module****class** climada.engine.cost\_benefit.**CostBenefit**

Bases: object

Impact definition. Compute from an entity (exposures and impact functions) and hazard.

**present\_year**

present reference year

**Type** int**future\_year**

future year

**Type** int**tot\_climate\_risk**

total climate risk without measures

**Type** float**unit**

unit used for impact

**Type** str**color\_rgb**

color code RGB for each measure.

**Type** dict**Key**

measure name ('no measure' used for case without measure),

**Type** str**Value****Type** np.array**benefit**

benefit of each measure. Key: measure name, Value: float benefit

**Type** dict**cost\_ben\_ratio**

cost benefit ratio of each measure. Key: measure name, Value: float cost benefit ratio

**Type** dict**imp\_meas\_future**

impact of each measure at future or default. Key: measure name ('no measure' used for case without measure), Value: dict with: 'cost' (tuple): (cost measure, cost factor insurance), 'risk' (float): risk measurement, 'risk\_transf' (float): annual expected risk transfer, 'efc' (ImpactFreqCurve): impact exceedance freq (optional) 'impact' (Impact): impact instance

**Type** dict**imp\_meas\_present**

impact of each measure at present. Key: measure name ('no measure' used for case without measure), Value: dict with: 'cost' (tuple): (cost measure, cost factor insurance), 'risk' (float): risk measurement, 'risk\_transf' (float): annual expected risk transfer, 'efc' (ImpactFreqCurve): impact exceedance freq (optional) 'impact' (Impact): impact instance



Type dict

`__init__()`

Initialization

**calc**(*hazard*, *entity*, *haz\_future*=None, *ent\_future*=None, *future\_year*=None, *risk\_func*=<function *risk\_aai\_agg*>, *imp\_time\_depen*=None, *save\_imp*=False)

Compute cost-benefit ratio for every measure provided current and, optionally, future conditions. Present and future measures need to have the same name. The measures costs need to be discounted by the user. If future entity provided, only the costs of the measures of the future and the discount rates of the present will be used.

#### Parameters

- **hazard** (*climada.Hazard*)
- **entity** (*climada.entity*)
- **haz\_future** (*climada.Hazard*, *optional*) – hazard in the future (future year provided at *ent\_future*)
- **ent\_future** (*Entity*, *optional*) – entity in the future. Default is None
- **future\_year** (*int*, *optional*) – future year to consider if no *ent\_future*. Default is None provided. The benefits are added from the *entity.exposures.ref\_year* until *ent\_future.exposures.ref\_year*, or until *future\_year* if no *ent\_future* given. Default: *entity.exposures.ref\_year*+1
- **risk\_func** (*func optional*) – function describing risk measure to use to compute the annual benefit from the Impact. Default: average annual impact (aggregated).
- **imp\_time\_depen** (*float, optional*) – parameter which represents time evolution of impact (super- or sublinear). If None: all years count the same when there is no future hazard nor entity and 1 (linear annual change) when there is future hazard or entity. Default is None.
- **save\_imp** (*bool, optional*)
- **True if Impact of each measure is saved. Default is False.**

**combine\_measures**(*in\_meas\_names*, *new\_name*, *new\_color*, *disc\_rates*, *imp\_time\_depen*=None, *risk\_func*=<function *risk\_aai\_agg*>)

Compute cost-benefit of the combination of measures previously computed by **calc** with *save\_imp*=True. The benefits of the measures per event are added. To combine with risk transfer options use **apply\_risk\_transfer**.

#### Parameters

- **in\_meas\_names** (*list(str)*)
- **list with names of measures to combine**
- **new\_name** (*str*) – name to give to the new resulting measure *new\_color* (*np.array*): color code RGB for new measure, e.g. *np.array([0.1, 0.1, 0.1])*
- **disc\_rates** (*DiscRates*) – discount rates instance
- **imp\_time\_depen** (*float, optional*) – parameter which represents time evolution of impact (super- or sublinear). If None: all years count the same when there is no future hazard nor entity and 1 (linear annual change) when there is future hazard or entity. Default is None.
- **risk\_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).

#### Returns

**Return type** climada.CostBenefit

**apply\_risk\_transfer**(*meas\_name*, *attachment*, *cover*, *disc\_rates*, *cost\_fix*=0, *cost\_factor*=1, *imp\_time\_depen*=None, *risk\_func*=<function risk\_aai\_agg>)

Applies risk transfer to given measure computed before with saved impact and compares it to when no measure is applied. Appended to dictionaries of measures.

**Parameters**

- **meas\_name** (*str*) – name of measure where to apply risk transfer
- **attachment** (*float*) – risk transfer values attachment (deductible)
- **cover** (*float*) – risk transfer cover
- **cost\_fix** (*float*) – fixed cost of implemented innsurance, e.g. transaction costs
- **cost\_factor** (*float*, *optional*) – factor to which to multiply the insurance layer to compute its cost. Default is 1
- **imp\_time\_depen** (*float*, *optional*) – parameter which represents time evolution of impact (super- or sublinear). If None: all years count the same when there is no future hazard nor entity and 1 (linear annual change) when there is future hazard or entity. Default is None.
- **risk\_func** (*func*, *optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).

**remove\_measure**(*meas\_name*)

Remove computed values of given measure

**Parameters** **meas\_name** (*str*) – name of measure to remove

**plot\_cost\_benefit**(*cb\_list*=None, *axis*=None, *\*\*kwargs*)

Plot cost-benefit graph. Call after calc().

**Parameters**

- **cb\_list** (*list(CostBenefit)*, *optional*) – if other CostBenefit provided, overlay them all. Used for uncertainty visualization.
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot*, *optional*) – axis to use
- **kwargs** (*optional*) – arguments for Rectangle matplotlib, e.g. alpha=0.5 (color is set by measures color attribute)

**Returns**

**Return type** matplotlib.axes.\_subplots.AxesSubplot

**plot\_event\_view**(*return\_per*=(10, 25, 100), *axis*=None, *\*\*kwargs*)

Plot averted damages for return periods. Call after calc().

**Parameters**

- **return\_per** (*list*, *optional*) – years to visualize. Default 10, 25, 100
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot*, *optional*) – axis to use
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5 (color is set by measures color attribute)

**Returns**

**Return type** matplotlib.axes.\_subplots.AxesSubplot

**static plot\_waterfall**(*hazard, entity, haz\_future, ent\_future, risk\_func=<function risk\_aai\_agg>, axis=None, \*\*kwargs*)

Plot waterfall graph at future with given risk metric. Can be called before and after calc().

#### Parameters

- **hazard** (*climada.Hazard*)
- **entity** (*climada.Entity*)
- **haz\_future** (*Hazard*) – hazard in the future (future year provided at ent\_future)
- **ent\_future** (*climada.Entity*) – entity in the future
- **risk\_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5

#### Returns

**Return type** matplotlib.axes.\_subplots.AxesSubplot

**plot\_arrow\_averted**(*axis, in\_meas\_names=None, accumulate=False, combine=False, risk\_func=<function risk\_aai\_agg>, disc\_rates=None, imp\_time\_depen=1, \*\*kwargs*)

Plot waterfall graph with accumulated values from present to future year. Call after calc() with save\_imp=True.

#### Parameters

- **axis** (*matplotlib.axes.\_subplots.AxesSubplot*) – axis from plot\_waterfall or plot\_waterfall\_accumulated where arrow will be added to last bar
- **in\_meas\_names** (*list(str), optional*) – list with names of measures to represented total averted damage. Default: all measures
- **accumulate** (*bool, optional*) – accumulated averted damage (True) or averted damage in future (False). Default: False
- **combine** (*bool, optional*) – use combine\_measures to compute total averted damage (True) or just add benefits (False). Default: False
- **risk\_func** (*func, optional*) – function describing risk measure given an Impact used in combine\_measures. Default: average annual impact (aggregated).
- **disc\_rates** (*DiscRates, optional*) – discount rates used in combine\_measures
- **imp\_time\_depen** (*float, optional*) – parameter which represent time evolution of impact used in combine\_measures. Default: 1 (linear).
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5

**plot\_waterfall\_accumulated**(*hazard, entity, ent\_future, risk\_func=<function risk\_aai\_agg>, imp\_time\_depen=1, axis=None, \*\*kwargs*)

Plot waterfall graph with accumulated values from present to future year. Call after calc() with save\_imp=True. Provide same inputs as in calc.

#### Parameters

- **hazard** (*climada.Hazard*)
- **entity** (*climada.Entity*)

- **ent\_future** (*climada.Entity*) – entity in the future
- **risk\_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).
- **imp\_time\_depen** (*float, optional*) – parameter which represent time evolution of impact used in combine\_measures. Default: 1 (linear).
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5

**Returns**

**Return type** matplotlib.axes.\_subplots.AxesSubplot

`climada.engine.cost_benefit.risk_aai_agg(impact)`

Risk measurement as average annual impact aggregated.

**Parameters** **impact** (*climada.engine.Impact*) – an Impact instance

**Returns**

**Return type** float

`climada.engine.cost_benefit.risk_rp_100(impact)`

Risk measurement as exceedance impact at 100 years return period.

**Parameters** **impact** (*climada.engine.Impact*) – an Impact instance

**Returns**

**Return type** float

`climada.engine.cost_benefit.risk_rp_250(impact)`

Risk measurement as exceedance impact at 250 years return period.

**Parameters** **impact** (*climada.engine.Impact*) – an Impact instance

**Returns**

**Return type** float

## climada.engine.forecast module

**class** `climada.engine.forecast.Forecast`(*hazard\_dict, exposure, impact\_funcs, haz\_model='NWP', exposure\_name=None*)

Bases: object

Forecast definition. Compute an impact forecast with predefined hazard originating from a forecast (like numerical weather prediction models), exposure and impact. Use the `calc()` method to calculate a forecasted impact. Then use the plotting methods to illustrate the forecasted impacts. By default plots are saved under in a `'/forecast/plots'` folder in the configurable `save_dir` in `local_data` (see `climada.util.config`) under a name summarizing the Hazard type, haz model name, initialization time of the forecast run, event date, exposure name and the plot title. As the class is relatively new, there might be future changes to the attributes, the methods, and the parameters used to call the methods. It was discovered at some point, that there might be a memory leak in matplotlib even when figures are closed (<https://github.com/matplotlib/matplotlib/issues/8519>). Due to this reason the plotting functions in this module have the flag `close_fig`, to close figures within the function scope, which might mitigate that problem if a script runs this plotting functions many times.

**run\_datetime**

initialization time of the forecast model run used to create the Hazard

**Type** list of `datetime.datetime`

**event\_date**

Date on which the Hazard event takes place

**Type** datetime.datetime

**hazard**

List of the hazard forecast with different lead times.

**Type** list of CLIMADA Hazard

**haz\_model**

Short string specifying the model used to create the hazard, if possible three big letters.

**Type** str

**exposure**

an CLIMADA Exposures containing values at risk

**Type** Exposure

**exposure\_name**

string specifying the exposure (e.g. 'EU'), which is used to name output files.

**Type** str

**vulnerability**

Set of impact functions used in the impact calculation.

**Type** *ImpactFuncSet*

**\_\_init\_\_**(*hazard\_dict, exposure, impact\_funcs, haz\_model='NWP', exposure\_name=None*)

Initialization with hazard, exposure and vulnerability.

**Parameters**

- **hazard\_dict** (*dict*) – Dictionary of the format {run\_datetime: Hazard} with run\_datetime being the initialization time of a weather forecast run and Hazard being a CLIMADA Hazard derived from that forecast for one event. A probabilistic representation of that one event is possible, as long as the attribute Hazard.date is the same for all events. Several run\_datetime:Hazard combinations for the same event can be provided.
- **exposure** (*Exposure*)
- **impact\_funcs** (*ImpactFuncSet*)
- **haz\_model** (*str, optional*) – Short string specifying the model used to create the hazard, if possible three big letters. Default is 'NWP' for numerical weather prediction.
- **exposure\_name** (*str, optional*) – string specifying the exposure (e.g. 'EU'), which is used to name output files.

**ei\_exp**(*run\_datetime=None*)

Expected impact per exposure

**Parameters** **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run\_datetime, default is first element of attribute run\_datetime.

**Returns**

**Return type** float

**ai\_agg**(*run\_datetime=None*)

average impact aggregated over all exposures

**Parameters** **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run\_datetime, default is first element of attribute run\_datetime.

**Returns****Return type** float**haz\_summary\_str**(*run\_datetime=None*)

provide a summary string for the hazard part of the forecast

**Parameters** **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run\_datetime, default is first element of attribute run\_datetime.**Returns** summarizing the most important information about the hazard**Return type** str**summary\_str**(*run\_datetime=None*)

provide a summary string for the impact forecast

**Parameters** **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run\_datetime, default is first element of attribute run\_datetime.**Returns** summarizing the most important information about the impact forecast**Return type** str**lead\_time**(*run\_datetime=None*)

provide the lead time for the impact forecast

**Parameters**

- **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run\_datetime, default is first element of attribute run\_datetime.

**Returns**

- **\_\_\_\_\_**
- **datetime.timedelta** – the difference between the initialization time of the forecast model run and the date of the event, commonly named lead time

**calc**(*force\_reassign=False*)

calculate the impacts for all lead times using exposure, all hazards of all run\_datetime, and ImpactFunctionSet.

**Parameters** **force\_reassign** (*bool, optional*) – Reassign hazard centroids to the exposure for all hazards, default is false.**plot\_imp\_map**(*run\_datetime=None, save\_fig=True, close\_fig=False, polygon\_file=None, polygon\_file\_crs='epsg:4326', proj=<cartopy.crs.PlateCarree object>, figsize=(9, 13), adapt\_fontsize=True*)

plot a map of the impacts

**Parameters**

- **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run\_datetime, default is first element of attribute run\_datetime.
- **save\_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable save\_dir and filename is derived from the method summary\_str() (for more details see class docstring). Default is True.
- **close\_fig** (*bool, optional*) – Figure not drawn if True. Default is False.
- **polygon\_file** (*str, optional*) – Points to a .shp-file with polygons to be drawn as outlines on the plot, default is None to not draw the lines. please also specify the crs in the parameter polygon\_file\_crs.

- **polygon\_file\_crs** (*str, optional*) – String of pattern <provider>:<code> specifying the crs. has to be readable by pyproj.Proj. Default is 'epsg:4326'.
- **proj** (*ccrs*) – coordinate reference system used in coordinates The default is `ccrs.PlateCarree()`
- **figsize** (*tuple*) – figure size for `plt.subplots`, width, height in inches The default is (9, 13)
- **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.

**Returns axes**

**Return type** `cartopy.mpl.geoaxes.GeoAxesSubplot`

**plot\_hist**(*run\_datetime=None, save\_fig=True, close\_fig=False, figsize=(9, 8)*)

plot histogram of the forecasted impacts all ensemble members

**Parameters**

- **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the `run_datetime`, default is first element of attribute `run_datetime`.
- **save\_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable `save_dir` and filename is derived from the method `summary_str()` (for more details see class `docstring`). Default is True.
- **close\_fig** (*bool, optional*) – Figure is not drawn if True. Default is False.
- **figsize** (*tuple*) – figure size for `plt.subplots`, width, height in inches The default is (9, 8)

**Returns axes**

**Return type** `matplotlib.axes.Axes`

**plot\_exceedence\_prob**(*threshold, explain\_str=None, run\_datetime=None, save\_fig=True, close\_fig=False, polygon\_file=None, polygon\_file\_crs='epsg:4326', proj=<cartopy.crs.PlateCarree object>, figsize=(9, 13), adapt\_fontsize=True*)

plot exceedence map

**Parameters**

- **threshold** (*float*) – Threshold of impact unit for which exceedence probability should be plotted.
- **explain\_str** (*str, optional*) – Short str which explains threshold, `explain_str` is included in the title of the figure.
- **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the `run_datetime`, default is first element of attribute `run_datetime`.
- **save\_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable `save_dir` and filename is derived from the method `summary_str()` (for more details see class `docstring`). Default is True.
- **close\_fig** (*bool, optional*) – Figure not drawn if True. Default is False.
- **polygon\_file** (*str, optional*) – Points to a .shp-file with polygons do be drawn as outlines on the plot, default is None to not draw the lines. please also specify the crs in the parameter `polygon_file_crs`.
- **polygon\_file\_crs** (*str, optional*) – String of pattern <provider>:<code> specifying the crs. has to be readable by pyproj.Proj. Default is 'epsg:4326'.

- **proj** (*ccrs*) – coordinate reference system used in coordinates The default is `ccrs.PlateCarree()`
- **figsize** (*tuple*) – figure size for `plt.subplots`, width, height in inches The default is (9, 13)
- **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.

**Returns axes**

**Return type** `cartopy.mpl.geoaxes.GeoAxesSubplot`

**plot\_warn\_map**(*polygon\_file=None, polygon\_file\_crs='epsg:4326', thresholds='default', decision\_level='exposure\_point', probability\_aggregation=0.5, area\_aggregation=0.5, title='WARNINGS', explain\_text='warn level based on thresholds', run\_datetime=None, proj=<cartopy.crs.PlateCarree object>, figsize=(9, 13), save\_fig=True, close\_fig=False, adapt\_fontsize=True*)

plot map colored with 5 warning colors for all regions in provided shape file.

**Parameters**

- **polygon\_file** (*str, optional*) – path to shp-file containing warning region polygons
- **polygon\_file\_crs** (*str, optional*) – String of pattern <provider>:<code> specifying the crs. has to be readable by `pyproj.Proj`. Default is 'epsg:4326'.
- **thresholds** (*list of 4 floats, optional*) – Thresholds for coloring region in second, third, forth and fifth warning color.
- **decision\_level** (*str, optional*) – Either 'exposure\_point' or 'polygon'. Default value is 'exposure\_point'.
- **probability\_aggregation** (*float or str, optional*) – Either a float between [0..1] specifying a quantile or 'mean' or 'sum'. Default value is 0.5.
- **area\_aggregation** (*float or str*) – Either a float between [0..1] specifying a quantile or 'mean' or 'sum'. Default value is 0.5.
- **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the `run_datetime`, default is first element of attribute `run_datetime`.
- **title** (*str, optional*) – Default is 'WARNINGS'.
- **explain\_text** (*str, optional*) – Default is 'warn level based on thresholds'.
- **proj** (*ccrs*) – coordinate reference system used in coordinates
- **figsize** (*tuple*) – figure size for `plt.subplots`, width, height in inches The default is (9, 13)
- **save\_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable `save_dir` and filename is derived from the method `summary_str()` (for more details see class `docstring`). Default is True.
- **close\_fig** (*bool, optional*) – Figure is not drawn if True. The default is False.
- **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.

**Returns axes**

**Return type** `cartopy.mpl.geoaxes.GeoAxesSubplot`

**plot\_hexbin\_ei\_exposure**(*run\_datetime=None, figsize=(9, 13)*)  
plot the expected impact

**Parameters**



- **run\_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run\_datetime, default is first element of attribute run\_datetime.
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 13)

**Returns** axes

**Return type** cartopy.mpl.geoaxes.GeoAxesSubplot

## climada.engine.impact module

**class** climada.engine.impact.**ImpactFreqCurve**

Bases: object

Impact exceedence frequency curve.

**tag**

dictionary of tags of exposures, impact functions set and hazard: {'exp': Tag(), 'impf\_set': Tag(), 'haz': TagHazard()}

**Type** dict

**return\_per**

return period

**Type** np.array

**impact**

impact exceeding frequency

**Type** np.array

**unit**

value unit used (given by exposures unit)

**Type** str

**label**

string describing source data

**Type** str

**\_\_init\_\_()**

**plot**(*axis=None, log\_frequency=False, \*\*kwargs*)

Plot impact frequency curve.

**Parameters**

- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **log\_frequency** (*boolean, optional*) – plot logarithmic exceedance frequency on x-axis
- **kwargs** (*optional*) – arguments for plot matplotlib function, e.g. color='b'

**Returns**

**Return type** matplotlib.axes.\_subplots.AxesSubplot

**class** climada.engine.impact.**Impact**

Bases: object

Impact definition. Compute from an entity (exposures and impact functions) and hazard.

**tag**  
dictionary of tags of exposures, impact functions set and hazard: {'exp': Tag(), 'impf\_set': Tag(), 'haz': TagHazard()}  
**Type** dict

**event\_id**  
np.array id (>0) of each hazard event

**event\_name**  
list name of each hazard event

**date**  
date if events as integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)  
**Type** np.array

**coord\_exp**  
exposures coordinates [lat, lon] (in degrees)  
**Type** np.array

**eai\_exp**  
expected annual impact for each exposure  
**Type** np.array

**at\_event**  
impact for each hazard event  
**Type** np.array

**frequency**  
annual frequency of event  
**Type** np.array

**tot\_value**  
total exposure value affected  
**Type** float

**aai\_agg**  
average annual impact (aggregated)  
**Type** float

**unit**  
value unit used (given by exposures unit)  
**Type** str

**imp\_mat**  
matrix num\_events x num\_exp with impacts. only filled if save\_mat is True in calc()  
**Type** sparse.csr\_matrix

**\_\_init\_\_()**  
Empty initialization.

**calc\_freq\_curve**(*return\_per=None*)  
Compute impact exceedance frequency curve.  
**Parameters** **return\_per** (*np.array, optional*) – return periods where to compute the exceedance impact. Use impact's frequencies if not provided

**Returns****Return type** *ImpactFreqCurve***calc**(*exposures, impact\_funcs, hazard, save\_mat=False*)

Compute impact of an hazard to exposures.

**Parameters**

- **exposures** (*climada.entity.Exposures*)
- **impact\_funcs** (*climada.entity.ImpactFuncSet*) – impact functions
- **hazard** (*climada.Hazard*)
- **save\_mat** (*bool*) – self impact matrix: events x exposures

**Examples**

Use Entity class:

```
>>> haz = Hazard('TC') # Set hazard
>>> haz.read_mat(HAZ_DEMO_MAT)
>>> haz.check()
>>> ent = Entity() # Load entity with default values
>>> ent.read_excel(ENT_TEMPLATE_XLS) # Set exposures
>>> ent.check()
>>> imp = Impact()
>>> imp.calc(ent.exposures, ent.impact_funcs, haz)
>>> imp.calc_freq_curve().plot()
```

Specify only exposures and impact functions:

```
>>> haz = Hazard('TC') # Set hazard
>>> haz.read_mat(HAZ_DEMO_MAT)
>>> haz.check()
>>> funcs = ImpactFuncSet()
>>> funcs.read_excel(ENT_TEMPLATE_XLS) # Set impact functions
>>> funcs.check()
>>> exp = Exposures(pd.read_excel(ENT_TEMPLATE_XLS)) # Set exposures
>>> exp.check()
>>> imp = Impact()
>>> imp.calc(exp, funcs, haz)
>>> imp.aai_agg
```

**calc\_risk\_transfer**(*attachment, cover*)

Compute traditional risk transfer over impact. Returns new impact with risk transfer applied and the insurance layer resulting Impact metrics.

**Parameters**

- **attachment** (*float*) – (deductible)
- **cover** (*float*)

**Returns****Return type** *climada.engine.Impact*

**plot\_hexbin\_eai\_exposure**(*mask=None, ignore\_zero=True, pop\_name=True, buffer=0.0, extend='neither', axis=None, adapt\_fontsize=True, \*\*kwargs*)

Plot hexbin expected annual impact of each exposure.

#### Parameters

- **mask** (*np.array, optional*) – mask to apply to eai\_exp plotted.
- **ignore\_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop\_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 1.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. [ 'neither' | 'both' | 'min' | 'max' ]
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for hexbin matplotlib function

#### Returns

**Return type** cartopy.mpl.geoaxes.GeoAxesSubplot

**plot\_scatter\_eai\_exposure**(*mask=None, ignore\_zero=True, pop\_name=True, buffer=0.0, extend='neither', axis=None, adapt\_fontsize=True, \*\*kwargs*)

Plot scatter expected annual impact of each exposure.

#### Parameters

- **mask** (*np.array, optional*) – mask to apply to eai\_exp plotted.
- **ignore\_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop\_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 1.0.
- **extend** (*str*) – optional extend border colorbar with arrows. [ 'neither' | 'both' | 'min' | 'max' ]
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **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** (*optional*) – arguments for hexbin matplotlib function

#### Returns

**Return type** cartopy.mpl.geoaxes.GeoAxesSubplot

**plot\_raster\_eai\_exposure**(*res=None, raster\_res=None, save\_tiff=None, raster\_f=<function Impact.<lambda>>, label='value (log10)', axis=None, adapt\_fontsize=True, \*\*kwargs*)

Plot raster expected annual impact of each exposure.

#### Parameters

- **res** (*float, optional*) – resolution of current data in units of latitude and longitude, approximated if not provided.
- **raster\_res** (*float, optional*) – desired resolution of the raster

- **save\_tiff** (*str, optional*) – file name to save the raster in tiff format, if provided
- **raster\_f** (*lambda function*) – transformation to use to data. Default: log10 adding 1.
- **label** (*str colorbar label*)
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **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** (*optional*) – arguments for imshow matplotlib function

**Returns**

**Return type** cartopy.mpl.geoaxes.GeoAxesSubplot

**plot\_basemap\_eai\_exposure**(*mask=None, ignore\_zero=False, pop\_name=True, buffer=0.0, extend='neither', zoom=10, url='http://tile.stamen.com/terrain/tileZ/tileX/tileY.png', axis=None, \*\*kwargs*)

Plot basemap expected annual impact of each exposure.

**Parameters**

- **mask** (*np.array, optional*) – mask to apply to eai\_exp plotted.
- **ignore\_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop\_name** (*bool, optional*) – add names of the populated places buffer : float, optional border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. [ 'neither' | 'both' | 'min' | 'max' ]
- **zoom** (*int, optional*) – zoom coefficient used in the satellite image
- **url** (*str, optional*) – image source, e.g. ctx.sources.OSM\_C
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'. Default: 'Wistia'

**Returns**

**Return type** cartopy.mpl.geoaxes.GeoAxesSubplot

**plot\_hexbin\_impact\_exposure**(*event\_id=1, mask=None, ignore\_zero=True, pop\_name=True, buffer=0.0, extend='neither', axis=None, adapt\_fontsize=True, \*\*kwargs*)

Plot hexbin impact of an event at each exposure. Requires attribute imp\_mat.

**Parameters**

- **event\_id** (*int, optional*) – id of the event for which to plot the impact. Default: 1.
- **mask** (*np.array, optional*) – mask to apply to impact plotted.
- **ignore\_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop\_name** (*bool, optional*) – add names of the populated places buffer : float, optional border to add to coordinates. Default: 1.0.

- **extend** (*str, optional*) – extend border colorbar with arrows. [ ‘neither’ | ‘both’ | ‘min’ | ‘max’ ]
- **kwargs** (*optional*) – arguments for hexbin matplotlib function
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot*) – optional axis to use
- **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.

#### Returns

**Return type** matplotlib.figure.Figure, cartopy.mpl.geoaxes.GeoAxesSubplot

**plot\_basemap\_impact\_exposure**(*event\_id=1, mask=None, ignore\_zero=True, pop\_name=True, buffer=0.0, extend='neither', zoom=10, url='http://tile.stamen.com/terrain/tileZ/tileX/tileY.png', axis=None, \*\*kwargs*)

Plot basemap impact of an event at each exposure. Requires attribute `imp_mat`.

#### Parameters

- **event\_id** (*int, optional*) – id of the event for which to plot the impact. Default: 1.
- **mask** (*np.array, optional*) – mask to apply to impact plotted.
- **ignore\_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop\_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. [ ‘neither’ | ‘both’ | ‘min’ | ‘max’ ]
- **zoom** (*int, optional*) – zoom coefficient used in the satellite image
- **url** (*str, optional*) – image source, e.g. `ctx.sources.OSM_C`
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional axis to use*)
- **kwargs** (*optional arguments for scatter matplotlib function, e.g.*) – `cmap='Greys'`. Default: ‘Wistia’

#### Returns

**Return type** cartopy.mpl.geoaxes.GeoAxesSubplot

**write\_csv**(*file\_name*)

Write data into csv file. `imp_mat` is not saved.

**Parameters** **file\_name** (*str*) – absolute path of the file

**write\_excel**(*file\_name*)

Write data into Excel file. `imp_mat` is not saved.

**Parameters** **file\_name** (*str*) – absolute path of the file

**write\_sparse\_csr**(*file\_name*)

Write `imp_mat` matrix in numpy’s npz format.

**calc\_impact\_year\_set**(*all\_years=True, year\_range=None*)

Calculate yearly impact from impact data.

#### Parameters

- **all\_years** (*boolean*) – return values for all years between first and last year with event, including years without any events.
- **year\_range** (*tuple or list with integers*) – start and end year

**Returns**

**Return type** Impact year set of type `numpy.ndarray` with summed impact per year.

**local\_exceedance\_imp**(*return\_periods=(25, 50, 100, 250)*)

Compute exceedance impact map for given return periods. Requires attribute `imp_mat`.

**Parameters** **return\_periods** (*np.array return periods to consider*)

**Returns**

**Return type** `np.array`

**plot\_rp\_imp**(*return\_periods=(25, 50, 100, 250), log10\_scale=True, smooth=True, axis=None, \*\*kwargs*)

Compute and plot exceedance impact maps for different return periods. Calls `local_exceedance_imp`.

**Parameters**

- **return\_periods** (*tuple(int), optional*) – return periods to consider
- **log10\_scale** (*boolean, optional*) – plot impact as  $\log_{10}(\text{impact})$
- **smooth** (*bool, optional*) – smooth plot to `plot.RESOLUTIONxplot.RESOLUTION`
- **kwargs** (*optional*) – arguments for `pcolormesh` matplotlib function used in event plots

**Returns**

- `matplotlib.axes._subplots.AxesSubplot`,
- `np.ndarray` (*return\_periods.size x num\_centroids*)

**static read\_sparse\_csr**(*file\_name*)

Read `imp_mat` matrix from numpy's npz format.

**Parameters** **file\_name** (*str file name*)

**Returns**

**Return type** `sparse.csr_matrix`

**read\_csv**(*file\_name*)

Read csv file containing impact data generated by `write_csv`.

**Parameters** **file\_name** (*str absolute path of the file*)

**read\_excel**(*file\_name*)

Read excel file containing impact data generated by `write_excel`.

**Parameters** **file\_name** (*str absolute path of the file*)

**static video\_direct\_impact**(*exp, impf\_set, haz\_list, file\_name="", writer=<matplotlib.animation.PillowWriter object>, imp\_thresh=0, args\_exp=None, args\_imp=None*)

Computes and generates video of accumulated impact per input events over exposure.

**Parameters**

- **exp** (*Exposures*) – exposures instance, constant during all video
- **impf\_set** (*ImpactFuncSet*) – impact functions

- **haz\_list** (*(list(Hazard))*) – every Hazard contains an event; all hazards use the same centroids
- **file\_name** (*str, optional*) – file name to save video, if provided
- **writer** (*matplotlib.animation., optional\**) – video writer. Default: pillow with bitrate=500
- **imp\_thresh** (*float*) – represent damages greater than threshold
- **args\_exp** (*optional*) – arguments for scatter (points) or hexbin (raster) matplotlib function used in exposures
- **args\_imp** (*optional*) – arguments for scatter (points) or hexbin (raster) matplotlib function used in impact

#### Returns

**Return type** *list(Impact)*

**select** (*event\_ids=None, event\_names=None, dates=None, coord\_exp=None*)

Select a subset of events and/or exposure points from the impact. If multiple input variables are not None, it returns all the impacts matching at least one of the conditions.

---

**Note:** the frequencies are NOT adjusted. Method to adjust frequencies

---

**and obtain correct eai\_exp:** 1- Select subset of impact according to your choice `imp = impact.select(...)`  
2- Adjust manually the frequency of the subset of impact `imp.frequency = [...]` 3- Use select without arguments to select all events and recompute the `eai_exp` with the updated frequencies. `imp = imp.select()`

#### Parameters

- **event\_ids** (*list[int], optional*) – Selection of events by their id. The default is None.
- **event\_names** (*list[str], optional*) – Selection of events by their name. The default is None.
- **dates** (*tuple(), optional*) – (start-date, end-date), events are selected if they are  $\geq$  than start-date and  $\leq$  than end-date. Dates in same format as `impact.date` (ordinal format of datetime library) The default is None.
- **coord\_exp** (*np.ndarray, optional*) – Selection of exposures coordinates [lat, lon] (in degrees) The default is None.

**Raises ValueError** – If the impact matrix is missing, the `eai_exp` and `aai_agg` cannot be updated for a selection of events and/or exposures.

**Returns imp** – A new impact object with a selection of events and/or exposures

**Return type** `climada.engine.Impact`



**climada.engine.impact\_data module**

`climada.engine.impact_data.assign_hazard_to_emdat`(*certainty\_level*, *intensity\_path\_haz*,  
*names\_path\_haz*, *reg\_id\_path\_haz*, *date\_path\_haz*,  
*emdat\_data*, *start\_time*, *end\_time*,  
*keep\_checks=False*)

`assign_hazard_to_emdat`: link EMdat event to hazard

**Parameters**

- **certainty\_level** (*str*) – ‘high’ or ‘low’
- **intensity\_path\_haz** (*sparse matrix*) – with hazards as rows and grid points as cols, values only at location with impacts
- **names\_path\_haz** (*str*) – identifier for each hazard (i.e. IBtracID) (rows of the matrix)
- **reg\_id\_path\_haz** (*str*) – ISO country ID of each grid point (cols of the matrix)
- **date\_path\_haz** (*str*) – start date of each hazard (rows of the matrix)
- **emdat\_data** (*pd.DataFrame*) – dataframe with EMdat data
- **start\_time** (*str*) – start date of events to be assigned ‘yyyy-mm-dd’
- **end\_time** (*str*) – end date of events to be assigned ‘yyyy-mm-dd’
- **keep\_checks** (*bool, optional*)

**Returns**

**Return type** *pd.dataframe* with EMdat entries linked to a hazard

`climada.engine.impact_data.hit_country_per_hazard`(*intensity\_path*, *names\_path*, *reg\_id\_path*,  
*date\_path*)

`hit_country_per_hazard`: create list of hit countries from hazard set

**Parameters**

- **intensity\_path** (*str*) – Path to file containing sparse matrix with hazards as rows and grid points as cols, values only at location with impacts
- **names\_path** (*str*) – Path to file with identifier for each hazard (i.e. IBtracID) (rows of the matrix)
- **reg\_id\_path** (*str*) – Path to file with ISO country ID of each grid point (cols of the matrix)
- **date\_path** (*str*) – Path to file with start date of each hazard (rows of the matrix)

**Returns**

**Return type** *pd.DataFrame* with all hit countries per hazard

`climada.engine.impact_data.create_lookup`(*emdat\_data*, *start*, *end*, *disaster\_subtype='Tropical cyclone'*)  
`create_lookup`: prepare a lookup table of EMdat events to which hazards can be assigned

**Parameters**

- **emdat\_data** (*pd.DataFrame*) – with EMdat data
- **start** (*str*) – start date of events to be assigned ‘yyyy-mm-dd’
- **end** (*str*) – end date of events to be assigned ‘yyyy-mm-dd’
- **disaster\_subtype** (*str*) – EMdat disaster subtype

**Returns**

**Return type** `pd.DataFrame`

`climada.engine.impact_data.emdat_possible_hit(lookup, hit_countries, delta_t)`  
relate EM disaster to hazard using hit countries and time

**Parameters**

- **lookup** (*pd.DataFrame*) – to relate EMdatID to hazard
- **delta\_t** – max time difference of start of EMdat event and hazard
- **hit\_countries**

**Returns**

**Return type** list with possible hits

`climada.engine.impact_data.match_em_id(lookup, poss_hit)`  
function to check if EM\_ID has been assigned already and combine possible hits

**Parameters**

- **lookup** (*pd.dataframe*) – to relate EMdatID to hazard
- **poss\_hit** (*list*) – with possible hits

**Returns** with all possible hits per EMdat ID

**Return type** list

`climada.engine.impact_data.assign_track_to_em(lookup, possible_tracks_1, possible_tracks_2, level)`  
function to assign a hazard to an EMdat event to get some confidence into the procedure, hazards get only assigned if there is no other hazard occurring at a bigger time interval in that country. Thus a track of possible\_tracks\_1 gets only assigned if there are no other tracks in possible\_tracks\_2. The confidence can be expressed with a certainty level

**Parameters**

- **lookup** (*pd.DataFrame*) – to relate EMdatID to hazard
- **possible\_tracks\_1** (*list*) – list of possible hits with smaller time horizon
- **possible\_tracks\_2** (*list*) – list of possible hits with larger time horizon
- **level** (*int*) – level of confidence

**Returns** lookup with assigned tracks and possible hits

**Return type** `pd.DataFrame`

`climada.engine.impact_data.check_assigned_track(lookup, checkset)`  
compare lookup with assigned tracks to a set with checked sets

**Parameters**

- **lookup** (*pd.DataFrame*) – dataframe to relate EMdatID to hazard
- **checkset** (*pd.DataFrame*) – dataframe with already checked hazards

**Returns**

**Return type** error scores

`climada.engine.impact_data.clean_emdat_df(emdat_file, countries=None, hazard=None, year_range=None, target_version=2020)`

Get a clean and standardized DataFrame from EM-DAT-CSV-file (1) load EM-DAT data from CSV to DataFrame and remove header/footer, (2) handle version, clean up, and add columns, and (3) filter by country, hazard type and year range (if any given)

**Parameters**

- **emdat\_file** (*str, Path, or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV
- **countries** (*list of str*) – country ISO3-codes or names, e.g. ['JAM', 'CUB']. countries=None for all countries (default)
- **hazard** (*list or str*) – List of Disaster (sub-)type according EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.
- **year\_range** (*list or tuple*) – Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)
- **target\_version** (*int*) – required EM-DAT data format version (i.e. year of download), changes naming of columns/variables (default: 2020)

**Returns** **df\_data** – DataFrame containing cleaned and filtered EM-DAT impact data

**Return type** pd.DataFrame

```
climada.engine.impact_data.emdat_countries_by_hazard(emdat_file_csv, hazard=None,
 year_range=None)
```

return list of all countries exposed to a chosen hazard type from EMDAT data as CSV.

**Parameters**

- **emdat\_file** (*str, Path, or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV
- **hazard** (*list or str*) – List of Disaster (sub-)type according EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.
- **year\_range** (*list or tuple*) – Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

**Returns**

- **countries\_iso3a** (*list*) – List of ISO3-codes of countries impacted by the disaster (sub-)types
- **countries\_names** (*list*) – List of names of countries impacted by the disaster (sub-)types

```
climada.engine.impact_data.scale_impact2refyear(impact_values, year_values, iso3a_values,
 reference_year=None)
```

Scale give impact values proportional to GDP to the according value in a reference year (for normalization of monetary values)

**Parameters**

- **impact\_values** (*list or array*) – Impact values to be scaled.
- **year\_values** (*list or array*) – Year of each impact (same length as impact\_values)
- **iso3a\_values** (*list or array*) – ISO3alpha code of country for each impact (same length as impact\_values)
- **Optional Parameters**
- \_\_\_\_\_

- **reference\_year** (*int*) – Impact is scaled proportional to GDP to the value of the reference year. No scaling for reference\_year=None (default)

`climada.engine.impact_data.emdat_impact_yearlysum(emdat_file_csv, countries=None, hazard=None, year_range=None, reference_year=None, imp_str="Total Damages ('000 US$)", version=2020)`

function to load EM-DAT data and sum impact per year

**Parameters** `emdat_file_csv` (*str or DataFrame*) – Either string with full path to CSV-file or `pandas.DataFrame` loaded from EM-DAT CSV

**countries** [list of *str*] country ISO3-codes or names, e.g. ['JAM', 'CUB']. `countries=None` for all countries (default)

**hazard** [list or *str*] List of Disaster (sub-)type according EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.

**year\_range** [list or tuple] Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

**version** [*int*] required EM-DAT data format version (i.e. year of download), changes naming of columns/variables (default: 2020)

**Returns** `out` – `DataFrame` with summed impact and scaled impact per year and country.

**Return type** `pd.DataFrame`

`climada.engine.impact_data.emdat_impact_event(emdat_file_csv, countries=None, hazard=None, year_range=None, reference_year=None, imp_str="Total Damages ('000 US$)", version=2020)`

function to load EM-DAT data return impact per event

**Parameters** `emdat_file_csv` (*str or DataFrame*) – Either string with full path to CSV-file or `pandas.DataFrame` loaded from EM-DAT CSV

**countries** [list of *str*] country ISO3-codes or names, e.g. ['JAM', 'CUB']. default: `countries=None` for all countries

**hazard** [list or *str*] List of Disaster (sub-)type according EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.

**year\_range** [list or tuple] Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

**reference\_year** [*int*] reference year of exposures. Impact is scaled] proportional to GDP to the value of the reference year. Default: No scaling for 0

**imp\_str** [*str*] Column name of impact metric in EMDAT CSV, default = "Total Damages ('000 US\$)"

**version** [*int*] EM-DAT version to take variable/column names from (default: 2020)

**Returns** `out` – EMDAT `DataFrame` with new columns "year", "region\_id", and "impact" and "impact\_scaled" total impact per event with same unit as chosen impact, but multiplied by 1000 if impact is given as 1000 US\$ (e.g. `imp_str="Total Damages ('000 US$) scaled"`).

**Return type** `pd.DataFrame`

```
climada.engine.impact_data.emdat_to_impact(emdat_file_csv, hazard_type_climada, year_range=None,
 countries=None, hazard_type_emdat=None,
 reference_year=None, imp_str='Total Damages')
```

function to load EM-DAT data return impact per event

**Parameters** **emdat\_file\_csv** (*str or pd.DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV

**countries** [list of str] country ISO3-codes or names, e.g. ['JAM', 'CUB']. default: countries=None for all countries

**hazard\_type\_climada** [list or str] List of Disaster (sub-)type according EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.

**year\_range** [list or tuple] Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

**reference\_year** [int reference year of exposures. Impact is scaled] proportional to GDP to the value of the reference year. Default: No scaling for 0

**imp\_str** [str] Column name of impact metric in EMDAT CSV, default = “Total Damages (‘000 US\$)”

#### Returns

- **impact\_instance** (*instance of climada.engine.Impact*)
- *impact object of same format as output from CLIMADA*
- *impact computation.*
- *Values scaled with GDP to reference\_year if reference\_year is given.*
- *i.e. current US\$ for imp\_str=“Total Damages (‘000 US\$) scaled” (factor 1000 is applied)*
- *impact\_instance.eai\_exp holds expected annual impact for each country.*
- *impact\_instance.coord\_exp holds rough central coordinates for each country.*
- **countries (list)** (*ISO3-codes of countries in same order as in impact\_instance.eai\_exp*)

## 7.1.2 climada.entity package

### climada.entity.disc\_rates package

#### climada.entity.disc\_rates.base module

**class** climada.entity.disc\_rates.base.DiscRates

Bases: object

Defines discount rates and basic methods. Loads from files with format defined in FILE\_EXT.

**tag**

information about the source data

**Type** *Tag*

**years**

list of years

**Type** np.array

**rates**

list of discount rates for each year (between 0 and 1)

**Type** np.array

**\_\_init\_\_()**

Empty initialization.

**Examples**

Fill discount rates with values and check consistency data:

```
>>> disc_rates = DiscRates()
>>> disc_rates.years = np.array([2000, 2001])
>>> disc_rates.rates = np.array([0.02, 0.02])
>>> disc_rates.check()
```

Read discount rates from year\_2050.mat and checks consistency data.

```
>>> disc_rates = DiscRates(ENT_TEMPLATE_XLS)
```

**clear()**

Reinitialize attributes.

**check()**

Check attributes consistency.

**Raises ValueError** –

**select(year\_range)**

Select discount rates in given years.

**Parameters**

- **year\_range** (np.array) – continuous sequence of selected years.
- **Returns** (climada.entity.DiscRates) – The selected discrates in the year\_range

**append(disc\_rates)**

Check and append discount rates to current DiscRates. Overwrite discount rate if same year.

**Parameters** **disc\_rates** (climada.entity.DiscRates) – DiscRates instance to append

**Raises ValueError** –

**net\_present\_value(ini\_year, end\_year, val\_years)**

Compute net present value between present year and future year.

**Parameters**

- **ini\_year** (float) – initial year
- **end\_year** (float) – end year
- **val\_years** (np.array) – cash flow at each year btw ini\_year and end\_year (both included)

**Returns** **net\_present\_value** – net present value between present year and future year.

**Return type** float

**plot(axis=None, figsize=(6, 8), \*\*kwargs)**

Plot discount rates per year.

**Parameters**

- **axis** (*matplotlib.axes.\_subplots.AxesSubplot*, *optional*) – axis to use
- **figsize** (*tuple(int, int)*, *optional*) – size of the figure. The default is (6,8)
- **kwargs** (*optional*) – keyword arguments passed to plotting function `axis.plot`

**Returns** **axis** – axis handles of the plot

**Return type** `matplotlib.axes._subplots.AxesSubplot`

**read\_mat**(*file\_name*, *description*="", *var\_names*={'*field\_name*': '*discount*', '*sup\_field\_name*': '*entity*', '*var\_name*': {'*disc*': '*discount\_rate*', '*year*': '*year*'}}})

Read MATLAB file generated with previous MATLAB CLIMADA version.

#### Parameters

- **file\_name** (*str*) – filename including path and extension
- **description** (*str*, *optional*) – description of the data. The default is ''
- **var\_names** (*dict*, *optional*) – name of the variables in the file. The Default is `DEF_VAR_MAT = {'sup_field_name': 'entity', 'field_name': 'discount', 'var_name': {'year': 'year', 'disc': 'discount_rate'}}`

**read\_excel**(*file\_name*, *description*="", *var\_names*={'*col\_name*': {'*disc*': '*discount\_rate*', '*year*': '*year*'}, '*sheet\_name*': '*discount*'})

Read excel file following template and store variables.

#### Parameters

- **file\_name** (*str*) – filename including path and extension
- **description** (*str*, *optional*) – description of the data. The default is ''
- **var\_names** (*dict*, *optional*) – name of the variables in the file. The Default is `DEF_VAR_EXCEL = {'sheet_name': 'discount', 'col_name': {'year': 'year', 'disc': 'discount_rate'}}`

**write\_excel**(*file\_name*, *var\_names*={'*col\_name*': {'*disc*': '*discount\_rate*', '*year*': '*year*'}, '*sheet\_name*': '*discount*'})

Write excel file following template.

#### Parameters

- **file\_name** (*str*) – filename including path and extension
- **var\_names** (*dict*, *optional*) – name of the variables in the file. The Default is `DEF_VAR_EXCEL = {'sheet_name': 'discount', 'col_name': {'year': 'year', 'disc': 'discount_rate'}}`

## climada.entity.exposures package

## climada.entity.exposures.litpop package

**climada.entity.exposures.litpop.gpw\_population module**

```
climada.entity.exposures.litpop.gpw_population.load_gpw_pop_shape(geometry, reference_year,
 gpw_version,
 data_dir=PosixPath('/home/docs/climada/data'),
 layer=0, verbatim=True)
```

Read gridded population data from TIFF and crop to given shape(s).

Note: A (free) NASA Earthdata login is necessary to download the data. Data can be downloaded e.g. for `gpw_version=11` from [https://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/gpw-v4-population-count-rev11/gpw-v4-population-count-rev11\\_2015\\_30\\_sec.tif.zip](https://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/gpw-v4-population-count-rev11/gpw-v4-population-count-rev11_2015_30_sec.tif.zip)

**Parameters**

- **geometry** (*shape(s) to crop data to in degree lon/lat.*) – for example `shapely.geometry.(Multi)Polygon` or `shapefile.Shape` from polygon(s) defined in a (country) shapefile.
- **reference\_year** (*int*) – target year for data extraction
- **gpw\_version** (*int*) – Version number of GPW population data, i.e. 11 for v4.11. The default is `CONFIG.exposures.litpop.gpw_population.gpw_version.int()`
- **data\_dir** (*Path, optional*) – Path to data directory holding GPW data folders. The default is `SYSTEM_DIR`.
- **layer** (*int, optional*) – relevant data layer in input TIFF file to return. The default is 0 and should not be changed without understanding the different data layers in the given TIFF file.
- **verbatim** (*bool (optional):*) – if False, output in `LOGGER` is suppressed. Default is True.

**Returns**

- **pop\_data** (*2D numpy array*) – contains extracted population count data per grid point in shape first dimension is lat, second dimension is lon.
- **meta** (*dict*) – contains meta data per array, including “transform” with meta data on coordinates.
- **global\_transform** (*Affine instance*) – contains six numbers, providing transform info for global GWP grid. `global_transform` is required for resampling on a globally consistent grid

```
climada.entity.exposures.litpop.gpw_population.get_gpw_file_path(gpw_version, reference_year,
 data_dir=PosixPath('/home/docs/climada/data'),
 verbatim=True)
```

Check available GPW population data versions and year closest to `reference_year` and return full path to TIFF file.

**Parameters**

- **gpw\_version** (*int (optional)*) – Version number of GPW population data, i.e. 11 for v4.11.
- **reference\_year** (*int (optional)*) – Data year is selected as close to `reference_year` as possible. The default is 2020.
- **data\_dir** (*pathlib.Path (optional)*) – Absolute path where files are stored. Default: `SYSTEM_DIR`

**Raises** `FileExistsError` –

**Returns** `pathlib.Path`

**Return type** path to input file with population data



**climada.entity.exposures.litpop.litpop module**

`climada.entity.exposures.litpop.litpop.GPW_VERSION = 11`

Version of Gridded Population of the World (GPW) input data. Check for updates.

**class** `climada.entity.exposures.litpop.litpop.LitPop(*args, meta=None, tag=None, ref_year=2018, value_unit='USD', crs=None, **kwargs)`

Bases: `climada.entity.exposures.base.Exposures`

Holds geopandas GeoDataFrame with metadata and columns (pd.Series) defined in Attributes of Exposures class. LitPop exposure values are disaggregated proportional to a combination of nightlight intensity (NASA) and Gridded Population data (SEDAC). Total asset values can be produced capital, population count, GDP, or non-financial wealth.

Calling sequence example: `exp = LitPop() country_names = ['CHE', 'Austria'] exp.set_countries(country_names) exp.plot()`

**exponents**

Defining powers (m, n) with which lit (nightlights) and pop (gpw) go into  $\text{Lit}^{**m} * \text{Pop}^{**n}$ . The default is (1,1).

**Type** tuple of two integers, optional

**fin\_mode**

Socio-economic value to be used as an asset base that is disaggregated. The default is 'pc'.

**Type** str, optional

**gpw\_version**

Version number of GPW population data, e.g. 11 for v4.11. The default is defined in GPW\_VERSION.

**Type** int, optional

**set\_countries**(*countries*, *res\_arcsec*=30, *exponents*=(1, 1), *fin\_mode*='pc', *total\_values*=None, *admin1\_calc*=False, *reference\_year*=2018, *gpw\_version*=11, *data\_dir*=PosixPath('/home/docs/climada/data'))

init LitPop exposure object for a list of countries (admin 0). Sets attributes *ref\_year*, *tag*, *crs*, *value*, *geometry*, *meta*, *value\_unit*, *exponents*, *fin\_mode*, *gpw\_version*, and *admin1\_calc*.

Alias: `set_country()`

**Parameters**

- **countries** (*list with str or int*) – list containing country identifiers: iso3alpha (e.g. 'JPN'), iso3num (e.g. 92) or name (e.g. 'Togo')
- **res\_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arcsec, this corresponds to roughly 1 km.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (nightlights) and pop (gpw) go into LitPop. To get nightlights<sup>3</sup> without population count: (3, 0). To use population count alone: (0, 1). Default: (1, 1)
- **fin\_mode** (*str, optional*) – Socio-economic value to be used as an asset base that is disaggregated to the grid points within the country \* 'pc': produced capital (Source: World Bank), incl. manufactured or  
built assets such as machinery, equipment, and physical structures *pc* is in constant 2014 USD.  
– **'pop': population count (source: GPW, same as gridded population).** The unit is 'people'.

- ‘gdp’: gross-domestic product (Source: World Bank) [USD]
- ‘income\_group’: **gdp multiplied by country’s income group+1** [USD]. Income groups are 1 (low) to 4 (high income).
- ‘nfw’: non-financial wealth (Source: Credit Suisse, of households only) [USD]
- ‘tw’: total wealth (Source: Credit Suisse, of households only) [USD]
- ‘norm’: normalized by country (no unit)
- ‘none’: LitPop per pixel is returned unchanged (no unit)

The default is ‘pc’.

- **total\_values** (*list containing numerics, same length as countries, optional*) – Total values to be disaggregated to grid in each country. The default is None. If None, the total number is extracted from other sources depending on the value of `fin_mode`.
- **admin1\_calc** (*boolean, optional*) – If True, distribute admin1-level GDP (if available). Default: False
- **reference\_year** (*int, optional*) – Reference year. Default: `CONFIG.exposures.def_ref_year`.
- **gpw\_version** (*int, optional*) – Version number of GPW population data. The default is `GPW_VERSION`
- **data\_dir** (*Path, optional*) – redefines path to input data directory. The default is `SYSTEM_DIR`.

**Raises ValueError –**

**set\_nightlight\_intensity**(*countries=None, shape=None, res\_arcsec=15, reference\_year=2018, data\_dir=PosixPath('/home/docs/clinada/data')*)

Wrapper around `set_countries / set_custom_shape`.

Initiate exposures instance with value equal to the original BlackMarble nightlight intensity resampled to the target resolution `res_arcsec`.

Provide either `countries` or `shape`.

#### Parameters

- **countries** (*list or str, optional*) – list containing country identifiers (name or iso3)
- **shape** (*Shape, Polygon or MultiPolygon, optional*) – geographical shape of target region, alternative to `countries`.
- **res\_arcsec** (*int, optional*) – Resolution in arc seconds. The default is 15.
- **reference\_year** (*int, optional*) – Reference year. The default is `CONFIG.exposures.def_ref_year`.
- **data\_dir** (*Path, optional*) – data directory. The default is None.

**set\_population**(*countries=None, shape=None, res\_arcsec=30, reference\_year=2018, gpw\_version=11, data\_dir=PosixPath('/home/docs/clinada/data')*)

Wrapper around `set_countries / set_custom_shape`.

Initiate exposures instance with value equal to GPW population count. Provide either `countries` or `shape`.

#### Parameters

- **countries** (*list or str, optional*) – list containing country identifiers (name or iso3)

- **shape** (*Shape, Polygon or MultiPolygon, optional*) – geographical shape of target region, alternative to *countries*.
- **res\_arcsec** (*int, optional*) – Resolution in arc seconds. The default is 30.
- **reference\_year** (*int, optional*) – Reference year (closest available GPW data year is used) The default is `CONFIG.exposures.def_ref_year`.
- **gpw\_version** (*int, optional*) – specify GPW data version. The default is 11.
- **data\_dir** (*Path, optional*) – data directory. The default is None.

**Raises ValueError** – Either countries or shape is required.

```
set_custom_shape_from_countries(shape, countries, res_arcsec=30, exponents=(1, 1), fin_mode='pc',
 admin1_calc=False, reference_year=2018, gpw_version=11,
 data_dir=PosixPath('/home/docs/climada/data'))
```

create LitPop exposure for *country* and then crop to given shape.

#### Parameters

- **shape** (*shapely.geometry.Polygon, MultiPolygon, shapereader.Shape,*) – or GeoSeries or list containing either Polygons or Multipolygons. Geographical shape for which LitPop Exposure is to be initiated.
- **countries** (*list with str or int*) – list containing country identifiers: iso3alpha (e.g. 'JPN'), iso3num (e.g. 92) or name (e.g. 'Togo')
- **res\_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arc-sec, this corresponds to roughly 1 km.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (night-lights) and pop (gpw) go into LitPop. Default: (1, 1)
- **fin\_mode** (*str, optional*) – Socio-economic value to be used as an asset base that is disaggregated to the grid points within the country \* 'pc': produced capital (Source: World Bank), incl. manufactured or

built assets such as machinery, equipment, and physical structures (pc is in constant 2014 USD)

- **'pop': population count (source: GPW, same as gridded population).** The unit is 'people'.
- **'gdp': gross-domestic product (Source: World Bank) [USD]**
- **'income\_group': gdp multiplied by country's income group+1 [USD]** Income groups are 1 (low) to 4 (high income).
- **'nfw': non-financial wealth (Source: Credit Suisse, of households only) [USD]**
- **'tw': total wealth (Source: Credit Suisse, of households only) [USD]**
- **'norm': normalized by country**
- **'none': LitPop per pixel is returned unchanged**

The default is 'pc'.

- **admin1\_calc** (*boolean, optional*) – If True, distribute admin1-level GDP (if available). Default: False
- **reference\_year** (*int, optional*) – Reference year for data sources. Default: 2020

- **gpw\_version** (*int, optional*) – Version number of GPW population data. The default is GPW\_VERSION
- **data\_dir** (*Path, optional*) – redefines path to input data directory. The default is SYSTEM\_DIR.

**Raises** `NotImplementedError` –

**Returns**

**Return type** `None`.

**set\_custom\_shape**(*shape, total\_value, res\_arcsec=30, exponents=(1, 1), value\_unit='USD', reference\_year=2018, gpw\_version=11, data\_dir=PosixPath('/home/docs/climada/data')*)

init LitPop exposure object for a custom shape. Requires user input regarding the total value to be disaggregated.

Sets attributes *ref\_year, tag, crs, value, geometry, meta, value\_unit, exponents, `fin\_mode`, gpw\_version*, and *admin1\_calc*.

This method can be used to initiated LitPop Exposure for sub-national regions such as states, districts, cantons, cities, ... but shapes and total value need to be provided manually. If these required input parameters are not known / available, better initiate Exposure for entire country and extract shape afterwards.

**Parameters**

- **shape** (*shapely.geometry.Polygon or MultiPolygon or shapereader.Shape.*) – Geographical shape for which LitPop Exposure is to be initiated.
- **total\_value** (*int, float or None type*) – Total value to be disaggregated to grid in shape. If None, no value is disaggregated.
- **res\_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default 30 arcsec corresponds to roughly 1 km.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (night-lights) and pop (gpw) go into LitPop.
- **value\_unit** (*str*) – Unit of exposure values. The default is USD.
- **reference\_year** (*int, optional*) – Reference year for data sources. Default: CONFIG.exposures.def\_ref\_year
- **gpw\_version** (*int, optional*) – Version number of GPW population data. The default is set in CONFIG.
- **data\_dir** (*Path, optional*) – redefines path to input data directory. The default is SYSTEM\_DIR.

**Raises**

- **NotImplementedError** –
- **ValueError** –
- **TypeError** –

**set\_country**(*countries, res\_arcsec=30, exponents=(1, 1), fin\_mode='pc', total\_values=None, admin1\_calc=False, reference\_year=2018, gpw\_version=11, data\_dir=PosixPath('/home/docs/climada/data')*)

init LitPop exposure object for a list of countries (admin 0). Sets attributes *ref\_year, tag, crs, value, geometry, meta, value\_unit, exponents, `fin\_mode`, gpw\_version*, and *admin1\_calc*.

Alias: `set_country()`

**Parameters**

- **countries** (*list with str or int*) – list containing country identifiers: iso3alpha (e.g. ‘JPN’), iso3num (e.g. 92) or name (e.g. ‘Togo’)
- **res\_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arc-sec, this corresponds to roughly 1 km.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (night-lights) and pop (gpw) go into LitPop. To get nightlights<sup>3</sup> without population count: (3, 0). To use population count alone: (0, 1). Default: (1, 1)
- **fin\_mode** (*str, optional*) – Socio-economic value to be used as an asset base that is disaggregated to the grid points within the country \* ‘pc’: produced capital (Source: World Bank), incl. manufactured or  
built assets such as machinery, equipment, and physical structures *pc* is in constant 2014 USD.
  - ‘**pop**’: population count (source: GPW, same as gridded population). The unit is ‘people’.
  - ‘**gdp**’: gross-domestic product (Source: World Bank) [USD]
  - ‘**income\_group**’: gdp multiplied by country’s income group+1 [USD]. Income groups are 1 (low) to 4 (high income).
  - ‘**nfw**’: non-financial wealth (Source: Credit Suisse, of households only) [USD]
  - ‘**tw**’: total wealth (Source: Credit Suisse, of households only) [USD]
  - ‘**norm**’: normalized by country (no unit)
  - ‘**none**’: LitPop per pixel is returned unchanged (no unit)

The default is ‘pc’.

- **total\_values** (*list containing numerics, same length as countries, optional*) – Total values to be disaggregated to grid in each country. The default is None. If None, the total number is extracted from other sources depending on the value of *fin\_mode*.
- **admin1\_calc** (*boolean, optional*) – If True, distribute admin1-level GDP (if available). Default: False
- **reference\_year** (*int, optional*) – Reference year. Default: CONFIG.exposures.def\_ref\_year.
- **gpw\_version** (*int, optional*) – Version number of GPW population data. The default is GPW\_VERSION
- **data\_dir** (*Path, optional*) – redefines path to input data directory. The default is SYSTEM\_DIR.

**Raises ValueError –**

`climada.entity.exposures.litpop.litpop.get_value_unit(fin_mode)`  
get *value\_unit* depending on *fin\_mode*

**Parameters** *fin\_mode* (Socio-economic value to be used as an asset base)

**Returns** *value\_unit*

**Return type** str

```
climada.entity.exposures.litpop.litpop.reproject_input_data(data_array_list, meta_list, i_align=0,
 target_res_arcsec=None,
 global_origins=(-180.0,
 89.99999999999991),
 resampling=Resampling.bilinear,
 conserve=None)
```

LitPop-specific wrapper around `u_coord.align_raster_data`.

Reprojects all arrays in `data_arrays` to a given resolution – all based on the population data grid.

#### Parameters

- **data\_array\_list** (*list or array of numpy arrays containing numbers*) – Data to be reprojected, i.e. list containing N (min. 1) 2D-arrays. The data with the reference grid used to align the global destination grid to should be first `data_array_list[i_align]`, e.g., pop (GPW population data) for LitPop.
- **meta\_list** (*list of dicts*) – meta data dictionaries of data arrays in same order as `data_array_list`. Required fields in each dict are 'dtype', 'width', 'height', 'crs', 'transform'. Example:

```
{'driver': 'GTiff', 'dtype': 'float32', 'nodata': 0, 'width': 2702,
 'height': 1939, 'count': 1, 'crs': CRS.from_epsg(4326), 'transform':
 Affine(0.008333333333333333, 0.0, -18.175000000000068,
 0.0, -0.008333333333333333, 43.79999999999993)}
```

The meta data with the reference grid used to define the global destination grid should be first in the list, e.g., GPW population data for LitPop.

- **i\_align** (*int, optional*) – Index/Position of meta in `meta_list` to which the global grid of the destination is to be aligned to (c.f. `u_coord.align_raster_data`) The default is 0.
- **target\_res\_arcsec** (*int, optional*) – target resolution in arcsec. The default is None, i.e. same resolution as reference data.
- **global\_origins** (*tuple with two numbers (lat, lon), optional*) – global lon and lat origins as basis for destination grid. The default is the same as for GPW population data:  
(-180.0, 89.99999999999991)
- **resampling** (*resampling function, optional*) – The default is `rasterio.warp.Resampling.bilinear`
- **conserve** (*str, optional, either 'mean' or 'sum'*) – Conserve mean or sum of data? The default is None (no conservation).

#### Returns

- **data\_array\_list** (*list*) – contains reprojected data sets
- **meta\_out** (*dict*) – contains meta data of new grid (same for all arrays)

```
climada.entity.exposures.litpop.litpop.gridpoints_core_calc(data_arrays, offsets=None,
 exponents=None,
 total_val_rescale=None)
```

Combines N dense numerical arrays by point-wise multiplication and optionally rescales to new total value: (1) An offset (1 number per array) is added to all elements in

the corresponding data array in `data_arrays` (optional).

(2) Numbers in each array are taken to the power of the corresponding exponent (optional).

- (3) Arrays are multiplied element-wise.
- (4) if `total_val_rescale` is provided, results are normalized and re-scaled with `total_val_rescale`.
- (5) One array with results is returned.

#### Parameters

- **data\_arrays** (*list or array of numpy arrays containing numbers*) – Data to be combined, i.e. list containing  $N$  (min. 1) arrays of same shape.
- **total\_val\_rescale** (*float or int, optional*) – Total value for optional rescaling of resulting array. All values in `result_array` are scaled so that the sum is equal to `total_val_rescale`. The default (None) implies no rescaling.
- **offsets** (*list or array containing  $N$  numbers  $\geq 0$ , optional*) – One numerical offset per array that is added (sum) to the corresponding array in `data_arrays`. The default (None) corresponds to `np.zeros(N)`.
- **exponents** (*list or array containing  $N$  numbers  $\geq 0$ , optional*) – One exponent per array used as power for the corresponding array. The default (None) corresponds to `np.ones(N)`.

**Raises ValueError** – If input lists don't have the same number of elements. Or: If arrays in `data_arrays` do not have the same shape.

**Returns result\_array** – Results from calculation described above.

**Return type** `np.array` of same shape as arrays in `data_arrays`

#### climada.entity.exposures.litpop.nightlight module

`climada.entity.exposures.litpop.nightlight.NOAA_RESOLUTION_DEG = 0.008333333333333333`  
NOAA nightlights coordinates resolution in degrees.

`climada.entity.exposures.litpop.nightlight.NASA_RESOLUTION_DEG = 0.004166666666666667`  
NASA nightlights coordinates resolution in degrees.

`climada.entity.exposures.litpop.nightlight.NASA_TILE_SIZE = (21600, 21600)`  
NASA nightlights tile resolution.

`climada.entity.exposures.litpop.nightlight.NOAA_BORDER = (-180, -65, 180, 75)`  
NOAA nightlights border (min\_lon, min\_lat, max\_lon, max\_lat)

`climada.entity.exposures.litpop.nightlight.BM_FILENAMES =`  
`['BlackMarble_%i_A1_geo_gray.tif', 'BlackMarble_%i_A2_geo_gray.tif',`  
`'BlackMarble_%i_B1_geo_gray.tif', 'BlackMarble_%i_B2_geo_gray.tif',`  
`'BlackMarble_%i_C1_geo_gray.tif', 'BlackMarble_%i_C2_geo_gray.tif',`  
`'BlackMarble_%i_D1_geo_gray.tif', 'BlackMarble_%i_D2_geo_gray.tif']`  
 Nightlight NASA files which generate the whole earth when put together.

`climada.entity.exposures.litpop.nightlight.load_nasa_n1_shape(geometry, year,`  
`data_dir=PosixPath('/home/docs/climada/data'),`  
`dtype='float32')`

Read nightlight data from NASA BlackMarble tiles cropped to given shape(s) and combine arrays from each tile.  
 1) check and download required blackmarble files 2) read and crop data from each file required in a bounding box around

the given *geometry*.

- 3) **combine data from all input files into one array. this array then** contains all data in the geographic bounding box around *geometry*.
- 4) return array with nightlight data

**Parameters**

- **geometry** (*shape(s) to crop data to in degree lon/lat.*) – for example `shapely.geometry.(Multi)Polygon` or `shapefile.Shape`. from polygon defined in a shapefile. The object should have attribute 'bounds' or 'points'
- **year** (*int*) – target year for nightlight data, e.g. 2016. Closest available year is selected.
- **data\_dir** (*Path (optional)*) – Path to directory with BlackMarble data. The default is `SYSTEM_DIR`.
- **dtype** (*dtype*) – data type for output default 'float32', required for LitPop, choose 'int8' for integer.

**Returns**

- **results\_array** (*numpy array*) – extracted and combined nightlight data for bounding box around shape
- **meta** (*dict*) – rasterio meta data for results\_array

`climada.entity.exposures.litpop.nightlight.get_required_nl_files(bounds)`

**Determines which of the satellite pictures are necessary for** a certain bounding box (e.g. country)

**Parameters** **bounds** (*1x4 tuple*) – bounding box from shape (min\_lon, min\_lat, max\_lon, max\_lat).

**Raises** **ValueError** – invalid *bounds*

**Returns** **req\_files** – Array indicating the required files for the current operation with a boolean value (1: file is required, 0: file is not required).

**Return type** numpy array

`climada.entity.exposures.litpop.nightlight.check_nl_local_file_exists(required_files=None,  
check_path=PosixPath('/home/docs/climada'),  
year=2016)`

Checks if BM Satellite files are available and returns a vector denoting the missing files.

**Parameters**

- **required\_files** (*numpy array, optional*) – boolean array of dimension (8,) with which some files can be skipped. Only files with value 1 are checked, with value zero are skipped. The default is `np.ones(len(BM_FILENAMES),)`
- **check\_path** (*str or Path*) – absolute path where files are stored. Default: `SYSTEM_DIR`
- **year** (*int*) – year of the image, e.g. 2016

**Returns** **files\_exist** – Boolean array that denotes if the required files exist.

**Return type** numpy array



```
climada.entity.exposures.litpop.nightlight.download_n1_files(req_files=array([1., 1., 1., 1., 1., 1.,
1., 1.]), files_exist=array([0., 0., 0., 0., 0., 0., 0., 0.]),
dwnl_path=PosixPath('/home/docs/climada/data'),
year=2016)
```

Attempts to download nightlight files from NASA webpage.

#### Parameters

- **req\_files** (*numpy array, optional*) –  
**Boolean array which indicates the files required (0-> skip, 1-> download).** The default is `np.ones(len(BM_FILENAMES),)`.
- **files\_exist** (*numpy array, optional*) –  
**Boolean array which indicates if the files already exist locally and should not be downloaded (0-> download, 1-> skip).** The default is `np.zeros(len(BM_FILENAMES),)`.
- **dwnl\_path** (*str or path, optional*) – Download directory path. The default is `SYSTEM_DIR`.
- **year** (*int, optional*) – Data year to be downloaded. The default is 2016.

#### Raises

- **ValueError** –
- **RuntimeError** –

**Returns** `dwnl_path` – Download directory path.

**Return type** `str` or `path`

```
climada.entity.exposures.litpop.nightlight.load_nasa_n1_shape_single_tile(geometry, path,
layer=0)
```

Read nightlight data from single NASA BlackMarble tile and crop to given shape.

#### Parameters

- **geometry** (*shape or geometry object*) – shape(s) to crop data to in degree lon/lat. for example `shapely.geometry.Polygon` object or from polygon defined in a shapefile.
- **path** (*Path or str*) – full path to BlackMarble tif (including filename)
- **layer** (*int, optional*) – TIFF-layer to be returned. The default is 0. BlackMarble usually comes with 3 layers.

#### Returns

- **out\_image[layer, (:)]** : *2D numpy ndarray* – 2d array with data cropped to bounding box of shape
- **meta** (*dict*) – rasterio meta

```
climada.entity.exposures.litpop.nightlight.load_nightlight_nasa(bounds, req_files, year)
```

Get nightlight from NASA repository that contain input boundary.

Note: Legacy for BlackMarble, not required for litpop module

#### Parameters

- **bounds** (*tuple*) – min\_lon, min\_lat, max\_lon, max\_lat
- **req\_files** (*np.array*) – array with flags for NASA files needed

- **year** (*int*) – nightlight year

**Returns**

- **nightlight** (*sparse.csr\_matrix*)
- **coord\_nl** (*np.array*)

`climada.entity.exposures.litpop.nightlight.read_bm_file(bm_path, filename)`

Reads a single NASA BlackMarble GeoTiff and returns the data. Run all required checks first.

Note: Legacy for BlackMarble, not required for litpop module

**Parameters**

- **bm\_path** (*str*) – absolute path where files are stored.
- **filename** (*str*) – filename of the file to be read.

**Returns**

- **arr1** (*array*) – Raw BM data
- **curr\_file** (*gdal GeoTiff File*) – Additional info from which coordinates can be calculated.

`climada.entity.exposures.litpop.nightlight.unzip_tif_to_py(file_gz)`

Unzip image file, read it, flip the x axis, save values as pickle and remove tif.

**Parameters** **file\_gz** (*str*) – file fith .gz format to unzip

**Returns**

- **fname** (*str*) – file\_name of unzipped file
- **nightlight** (*sparse.csr\_matrix*)

`climada.entity.exposures.litpop.nightlight.untar_noaa_stable_nightlight(f_tar_ini)`

Move input tar file to SYSTEM\_DIR and extract stable light file. Returns absolute path of stable light file in format tif.gz.

**Parameters** **f\_tar\_ini** (*str*) – absolute path of file

**Returns** **f\_tif\_gz** – path of stable light file

**Return type** *str*

`climada.entity.exposures.litpop.nightlight.load_nightlight_noaa(ref_year=2013,  
sat_name=None)`

Get nightlight luminosites. Nightlight matrix, lat and lon ordered such that `nightlight[1][0]` corresponds to `lat[1]`, `lon[0]` point (the image has been flipped).

**Parameters**

- **ref\_year** (*int, optional*) – reference year. The default is 2013.
- **sat\_name** (*str, optional*) – satellite provider (e.g. 'F10', 'F18', ...)

**Returns**

- **nightlight** (*sparse.csr\_matrix*)
- **coord\_nl** (*np.array*)
- **fn\_light** (*str*)

**climada.entity.exposures.base module**

**class** climada.entity.exposures.base.**Exposures**(\*args, meta=None, tag=None, ref\_year=2018, value\_unit='USD', crs=None, \*\*kwargs)

Bases: object

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

**tag**

metadata - information about the source data

**Type** *Tag*

**ref\_year**

metadata - reference year

**Type** int

**value\_unit**

metadata - unit of the exposures values

**Type** str

**latitude**

latitude

**Type** pd.Series

**longitude**

longitude

**Type** pd.Series

**crs**

CRS information inherent to GeoDataFrame.

**Type** dict or crs

**value**

a value for each exposure

**Type** pd.Series

**impf\_**

e.g. `impf_TC`. impact functions id for hazard TC. There might be different hazards defined: `impf_TC`, `impf_FL`, ... If not provided, set to default '**impf\_**' with ids 1 in `check()`.

**Type** pd.Series, optional

**geometry**

geometry of type Point of each instance. Computed in method `set_geometry_points()`.

**Type** pd.Series, optional

**meta**

dictionary containing corresponding raster properties (if any): width, height, crs and transform must be present at least (transform needs to contain upper left corner!). Exposures might not contain all the points of the corresponding raster. Not used in internal computations.

**Type** dict

**deductible**

deductible value for each exposure

**Type** pd.Series, optional

**cover**

cover value for each exposure

**Type** pd.Series, optional

**category\_id**

category id for each exposure

**Type** pd.Series, optional

**region\_id**

region id for each exposure

**Type** pd.Series, optional

**centr\_**

e.g. centr\_TC. centroids index for hazard TC. There might be different hazards defined: centr\_TC, centr\_FL, ... Computed in method assign\_centroids().

**Type** pd.Series, optional

**vars\_oblig = ['value', 'latitude', 'longitude']**

Name of the variables needed to compute the impact.

**vars\_def = ['impf\_', 'if\_']**

Name of variables that can be computed.

**vars\_opt = ['centr\_', 'deductible', 'cover', 'category\_id', 'region\_id', 'geometry']**

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

**property crs**

Coordinate Reference System, refers to the crs attribute of the inherent GeoDataFrame

**\_\_init\_\_**(\*args, meta=None, tag=None, ref\_year=2018, value\_unit='USD', crs=None, \*\*kwargs)

Creates an Exposures object from a GeoDataFrame

**Parameters**

- **args** – Arguments of the GeoDataFrame constructor
- **kwargs** – Named arguments of the GeoDataFrame constructor, additionally
- **meta** (*dict, optional*) – Metadata dictionary. Default: {} (empty dictionary)
- **tag** (*climada.entity.exposures.tag.Tag, optional*) – Exposures tag. Defaults to the entry of the same name in *meta* or an empty Tag object.
- **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'.
- **crs** (*object, anything accepted by pyproj.CRS.from\_user\_input*) – Coordinate reference system. Defaults to the entry of the same name in *meta*, or to the CRS of the GeoDataFrame (if provided) or to 'epsg:4326'.

**check()**

Check Exposures consistency.

Reports missing columns in log messages. If no impf\_\* column is present in the dataframe, a default column '**impf\_**' is added with default impact function id 1.

**set\_crs**(*crs=None*)

Set the Coordinate Reference System. If the exposures GeoDataFrame has a 'geometry' column it will be updated too.

**Parameters** *crs* (*object, optional*) – anything anything accepted by pyproj.CRS.from\_user\_input if the original value is None it will be set to the default CRS.

**set\_gdf**(*gdf: geopandas.geodataframe.GeoDataFrame, crs=None*)

Set the *gdf* GeoDataFrame and update the CRS

**Parameters**

- **gdf** (*GeoDataFrame*)
- **crs** (*object, optional*) – anything anything accepted by pyproj.CRS.from\_user\_input, by default None, then *gdf.crs* applies or - if not set - the exposure's current crs

**get\_impf\_column**(*haz\_type=""*)

Find the best matching column name in the exposures dataframe for a given hazard type,

**Parameters** *haz\_type* (*str or None*) – hazard type, as in the hazard's tag.haz\_type which is the HAZ\_TYPE constant of the hazard's module

**Returns** a column name, the first of the following that is present in the exposures' dataframe:  
- *impf\_[haz\_type]* - *if\_[haz\_type]* - **impf\_** - **if\_**

**Return type** *str*

**Raises** **ValueError** – if none of the above is found in the dataframe.

**assign\_centroids**(*hazard, method='NN', distance='haversine', threshold=100*)

Assign for each exposure coordinate closest hazard coordinate. -1 used for disatances > threshold in point distances. If raster hazard, -1 used for centroids outside raster.

**Parameters**

- **hazard** (*Hazard*) – Hazard to match (with raster or vector centroids).
- **method** (*str, optional*) – Interpolation method to use in case of vector centroids. Currently, "NN" (nearest neighbor) is the only supported value, see *climada.util.interpolation.interpol\_index*.
- **distance** (*str, optional*) – Distance to use in case of vector centroids. Possible values are "haversine" and "approx", see *climada.util.interpolation.interpol\_index*. Default: "haversine"
- **threshold** (*float*) – If the distance to the nearest neighbor exceeds *threshold*, the index -1 is assigned. Set *threshold* to 0, to disable nearest neighbor matching. Default: 100 (km)

**set\_geometry\_points**(*scheduler=None*)

Set geometry attribute of GeoDataFrame with Points from latitude and longitude attributes.

**Parameters** *scheduler* (*str, optional*) – used for *dask map\_partitions*. "threads", "synchronous" or "processes"

**set\_lat\_lon**()

Set latitude and longitude attributes from geometry attribute.

**set\_from\_raster**(*file\_name, band=1, src\_crs=None, window=False, geometry=False, dst\_crs=False, transform=None, width=None, height=None, resampling=Resampling.nearest*)

Read raster data and set latitude, longitude, value and meta

**Parameters**

- **file\_name** (*str*) – file name containing values
- **band** (*int, optional*) – bands to read (starting at 1)
- **src\_crs** (*crs, optional*) – source CRS. Provide it if error without it.
- **window** (*rasterio.windows.Windows, optional*) – window where data is extracted
- **geometry** (*shapely.geometry, optional*) – consider pixels only in shape
- **dst\_crs** (*crs, optional*) – reproject to given crs
- **transform** (*rasterio.Affine*) – affine transformation to apply
- **width** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp..Resampling optional*) – resampling function used for re-projection to dst\_crs

**plot\_scatter**(*mask=None, ignore\_zero=False, pop\_name=True, buffer=0.0, extend='neither', axis=None, figsize=(9, 13), adapt\_fontsize=True, \*\*kwargs*)

Plot exposures geometry's value sum scattered over Earth's map. The plot will be projected according to the current crs.

#### Parameters

- **mask** (*np.array, optional*) – mask to apply to eai\_exp plotted.
- **ignore\_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop\_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. [ 'neither' | 'both' | 'min' | 'max' ]
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple, optional*) – figure size for plt.subplots
- **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** (*optional*) – arguments for scatter matplotlib function, e.g. `cmap='Greys'`. Default: 'Wistia'

#### Returns

**Return type** `cartopy.mpl.geoaxes.GeoAxesSubplot`

**plot\_hexbin**(*mask=None, ignore\_zero=False, pop\_name=True, buffer=0.0, extend='neither', axis=None, figsize=(9, 13), adapt\_fontsize=True, \*\*kwargs*)

Plot exposures geometry's value sum binned over Earth's map. An other function for the bins can be set through the key `reduce_C_function`. The plot will be projected according to the current crs.

#### Parameters

- **mask** (*np.array, optional*) – mask to apply to eai\_exp plotted.
- **ignore\_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False

- **pop\_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. [ 'neither' | 'both' | 'min' | 'max' ]
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple*) – figure size for plt.subplots
- **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** (*optional*) – arguments for hexbin matplotlib function, e.g. `reduce_C_function=np.average`. Default: `reduce_C_function=np.sum`

### Returns

**Return type** `cartopy.mpl.geoaxes.GeoAxesSubplot`

**plot\_raster**(*res=None, raster\_res=None, save\_tiff=None, raster\_f=<function Exposures.<lambda>>, label='value (log10)', scheduler=None, axis=None, figsize=(9, 13), fill=True, adapt\_fontsize=True, \*\*kwargs*)

Generate raster from points geometry and plot it using log10 scale: `np.log10((np.fmax(raster+1, 1)))`.

### Parameters

- **res** (*float, optional*) – resolution of current data in units of latitude and longitude, approximated if not provided.
- **raster\_res** (*float, optional*) – desired resolution of the raster
- **save\_tiff** (*str, optional*) – file name to save the raster in tiff format, if provided
- **raster\_f** (*lambda function*) – transformation to use to data. Default: log10 adding 1.
- **label** (*str*) – colorbar label
- **scheduler** (*str*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple, optional*) – figure size for plt.subplots
- **fill** (*bool, optional*) – If false, the areas with no data will be plotted in white. If True, the areas with missing values are filled as 0s. The default is True.
- **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** (*optional*) – arguments for imshow matplotlib function

### Returns

**Return type** `matplotlib.figure.Figure, cartopy.mpl.geoaxes.GeoAxesSubplot`

**plot\_basemap**(*mask=None, ignore\_zero=False, pop\_name=True, buffer=0.0, extend='neither', zoom=10, url='http://tile.stamen.com/terrain/tileZ/tileX/tileY.png', axis=None, \*\*kwargs*)

Scatter points over satellite image using contextily

### Parameters

- **mask** (*np.array, optional*) – mask to apply to eai\_exp plotted. Same size of the exposures, only the selected indexes will be plot.
- **ignore\_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop\_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. [ ‘neither’ | ‘both’ | ‘min’ | ‘max’ ]
- **zoom** (*int, optional*) – zoom coefficient used in the satellite image
- **url** (*str, optional*) – image source, e.g. ctx.sources.OSM\_C
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for scatter matplotlib function, e.g. cmap=‘Greys’. Default: ‘Wistia’

### Returns

**Return type** matplotlib.figure.Figure, cartopy.mpl.geoaxes.GeoAxesSubplot

### **write\_hdf5**(*file\_name*)

Write data frame and metadata in hdf5 format

**Parameters** **file\_name** (*str*) – (path and) file name to write to.

### **read\_hdf5**(*file\_name*)

Read data frame and metadata in hdf5 format

### **Parameters**

- **file\_name** (*str*) – (path and) file name to read from.
- **additional\_vars** (*list*) – list of additional variable names to read that are not in exposures.base.\_metadata

### **read\_mat**(*file\_name, var\_names=None*)

Read MATLAB file and store variables in exposures.

### **Parameters**

- **file\_name** (*str*) – absolute path file
- **var\_names** (*dict, optional*) – dictionary containing the name of the MATLAB variables. Default: DEF\_VAR\_MAT.

### **to\_crs**(*crs=None, epsg=None, inplace=False*)

Wrapper of the GeoDataFrame.to\_crs method.

Transform geometries to a new coordinate reference system. Transform all geometries in a GeoSeries to a different coordinate reference system. The crs attribute on the current GeoSeries must be set. Either crs in string or dictionary form or an EPSG code may be specified for output. This method will transform all points in all objects. It has no notion or projecting entire geometries. All segments joining points are assumed to be lines in the current projection, not geodesics. Objects crossing the dateline (or other projection boundary) will have undesirable behavior.

### **Parameters**

- **crs** (*dict or str*) – Output projection parameters as string or in dictionary form.
- **epsg** (*int*) – EPSG code specifying output projection.



- **inplace** (*bool, optional, default: False*) – Whether to return a new GeoDataFrame or do the transformation in place.

### Returns

**Return type** None if inplace is True else a transformed copy of the exposures object

**plot**(\*args, \*\*kwargs)

Plot a GeoDataFrame.

Generate a plot of a GeoDataFrame with matplotlib. If a column is specified, the plot coloring will be based on values in that column.

### Parameters

- **column** (*str, np.array, pd.Series (default None)*) – The name of the dataframe column, np.array, or pd.Series to be plotted. If np.array or pd.Series are used then it must have same length as dataframe. Values are used to color the plot. Ignored if *color* is also set.

- **kind** (*str*) –

#### The kind of plots to produce:

- ‘geo’: Map (default)

Pandas Kinds - ‘line’: line plot - ‘bar’: vertical bar plot - ‘barh’: horizontal bar plot - ‘hist’: histogram - ‘box’: BoxPlot - ‘kde’: Kernel Density Estimation plot - ‘density’: same as ‘kde’ - ‘area’: area plot - ‘pie’: pie plot - ‘scatter’: scatter plot - ‘hexbin’: hexbin plot.

- **cmap** (*str (default None)*) – The name of a colormap recognized by matplotlib.
- **color** (*str (default None)*) – If specified, all objects will be colored uniformly.
- **ax** (*matplotlib.pyplot.Artist (default None)*) – axes on which to draw the plot
- **cax** (*matplotlib.pyplot Artist (default None)*) – axes on which to draw the legend in case of color map.
- **categorical** (*bool (default False)*) – If False, cmap will reflect numerical values of the column being plotted. For non-numerical columns, this will be set to True.
- **legend** (*bool (default False)*) – Plot a legend. Ignored if no *column* is given, or if *color* is given.
- **scheme** (*str (default None)*) – Name of a choropleth classification scheme (requires mapclassify). A mapclassify.MapClassifier object will be used under the hood. Supported are all schemes provided by mapclassify (e.g. ‘BoxPlot’, ‘EqualInterval’, ‘FisherJenks’, ‘FisherJenksSampled’, ‘HeadTailBreaks’, ‘JenksCaspall’, ‘JenksCaspallForced’, ‘JenksCaspallSampled’, ‘MaxP’, ‘MaximumBreaks’, ‘NaturalBreaks’, ‘Quantiles’, ‘Percentiles’, ‘StdMean’, ‘UserDefined’). Arguments can be passed in *classification\_kwds*.
- **k** (*int (default 5)*) – Number of classes (ignored if scheme is None)
- **vmin** (*None or float (default None)*) – Minimum value of cmap. If None, the minimum data value in the column to be plotted is used.
- **vmax** (*None or float (default None)*) – Maximum value of cmap. If None, the maximum data value in the column to be plotted is used.
- **markersize** (*str or float or sequence (default None)*) – Only applies to point geometries within a frame. If a str, will use the values in the column of the frame specified by

markersize to set the size of markers. Otherwise can be a value to apply to all points, or a sequence of the same length as the number of points.

- **figsize** (*tuple of integers (default None)*) – Size of the resulting matplotlib.figure.Figure. If the argument axes is given explicitly, figsize is ignored.
  - **legend\_kwds** (*dict (default None)*) – Keyword arguments to pass to matplotlib.pyplot.legend() or matplotlib.pyplot.colorbar(). Additional accepted keywords when *scheme* is specified:
- fmt** [string] A formatting specification for the bin edges of the classes in the legend. For example, to have no decimals: {"fmt": "{:.0f}"}.  
**labels** [list-like] A list of legend labels to override the auto-generated labels. Needs to have the same number of elements as the number of classes (*k*).  
**interval** [boolean (default False)] An option to control brackets from mapclassify legend. If True, open/closed interval brackets are shown in the legend.
- **categories** (*list-like*) – Ordered list-like object of categories to be used for categorical plot.
  - **classification\_kwds** (*dict (default None)*) – Keyword arguments to pass to mapclassify
  - **missing\_kwds** (*dict (default None)*) – Keyword arguments specifying color options (as style\_kwds) to be passed on to geometries with missing values in addition to or overwriting other style kwds. If None, geometries with missing values are not plotted.
  - **aspect** ('auto', 'equal', None or float (default 'auto')) – Set aspect of axis. If 'auto', the default aspect for map plots is 'equal'; if however data are not projected (coordinates are long/lat), the aspect is by default set to  $1/\cos(df\_y * \pi/180)$  with *df\_y* the y coordinate of the middle of the GeoDataFrame (the mean of the y range of bounding box) so that a long/lat square appears square in the middle of the plot. This implies an Equirectangular projection. If None, the aspect of *ax* won't be changed. It can also be set manually (float) as the ratio of y-unit to x-unit.
  - **\*\*style\_kwds** (*dict*) – Style options to be passed on to the actual plot function, such as edgecolor, facecolor, linewidth, markersize, alpha.

**Returns** *ax*

**Return type** matplotlib axes instance

## Examples

```
>>> df = geopandas.read_file(geopandas.datasets.get_path("naturalearth_lowres"))
>>> df.head()
```

	pop_est	continent	geometry	name	iso_a3	gdp_md_est
0	920938	Oceania	Fiji	FJI	8374.0	
1	53950935	Africa	Tanzania	TZA	150600.0	
2	603253	Africa	W. Sahara	ESH	906.5	
3	35623680	North America	Canada	CAN	1674000.0	

(continues on next page)

(continued from previous page)

```
4 326625791 North America United States of America USA 185600000.0
→MULTIPOLYGON (((-122.84000 49.00000, -120.00000...
```

```
>>> df.plot("pop_est", cmap="Blues")
```

See the User Guide page `../user_guide/mapping` for details.

**copy**(*deep=True*)

Make a copy of this Exposures object.

**Parameters** **deep** (bool) (*Make a deep copy, i.e. also copy data. Default True.*)

**Returns**

**Return type** *Exposures*

**write\_raster**(*file\_name*, *value\_name='value'*, *scheduler=None*)

Write value data into raster file with GeoTiff format

**Parameters** **file\_name** (*str*) – name output file in tif format

**static concat**(*exposures\_list*)

Concatenates Exposures or DataFrame objects to one Exposures object.

**Parameters** **exposures\_list** (*list of Exposures or DataFrames*) – The list must not be empty with the first item supposed to be an Exposures object.

**Returns** with the metadata of the first item in the list and the dataframes concatenated.

**Return type** *Exposures*

`climada.entity.exposures.base.add_sea`(*exposures*, *sea\_res*, *scheduler=None*)

Add sea to geometry's surroundings with given resolution. `region_id` set to -1 and other variables to 0.

**Parameters**

- **exposures** (*Exposures*) – the Exposures object without sea surroundings.
- **sea\_res** (*tuple (float,float)*) – (`sea_coast_km`, `sea_res_km`), where first parameter is distance from coast to fill with water and second parameter is resolution between sea points
- **scheduler** (*str, optional*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”

**Returns**

**Return type** *Exposures*

`climada.entity.exposures.base.INDICATOR_IMP_F = 'impf_'`

Name of the column containing the impact functions id of specified hazard

`climada.entity.exposures.base.INDICATOR_CENTR = 'centr_'`

Name of the column containing the centroids id of specified hazard

**climada.entity.impact\_funcs package****climada.entity.impact\_funcs.base module****class** climada.entity.impact\_funcs.base.**ImpactFunc**

Bases: object

Contains the definition of one impact function.

**haz\_type**

hazard type acronym (e.g. 'TC')

**Type** str**id**

id of the impact function. Exposures of the same type will refer to the same impact function id

**Type** int or str**name**

name of the ImpactFunc

**Type** str**intensity\_unit**

unit of the intensity

**Type** str**intensity**

intensity values

**Type** np.array**mdd**

mean damage (impact) degree for each intensity (numbers in [0,1])

**Type** np.array**paa**

percentage of affected assets (exposures) for each intensity (numbers in [0,1])

**Type** np.array**\_\_init\_\_()**

Empty initialization.

**calc\_mdr**(inten)

Interpolate impact function to a given intensity.

**Parameters** *inten* (float or np.array) – intensity, the x-coordinate of the interpolated values.**Returns****Return type** np.array**plot**(axis=None, \*\*kwargs)Plot the impact functions MDD, MDR and PAA in one graph, where  $MDR = PAA * MDD$ .**Parameters**

- **axis** (matplotlib.axes.\_subplots.AxesSubplot, optional) – axis to use
- **kwargs** (optional) – arguments for plot matplotlib function, e.g. marker='x'

**Returns**

**Return type** matplotlib.axes.\_subplots.AxesSubplot

**check()**

Check consistent instance data.

**Raises ValueError** –

**set\_step\_impf**(*intensity*, *mdd*=(0, 1), *paa*=(1, 1), *impf\_id*=1)

Step function type impact function.

By default, everything is destroyed above the step. Useful for high resolution modelling.

This method modifies self (climada.entity.impact\_funcs instance) by assigning an id, intensity, mdd and paa to the impact function.

**Parameters**

- **intensity** (*tuple(float, float, float)*) – tuple of 3-intensity numbers: (minimum, threshold, maximum)
- **mdd** (*tuple(float, float)*) – (min, max) mdd values. The default is (0, 1)
- **paa** (*tuple(float, float)*) – (min, max) paa values. The default is (1, 1)
- **impf\_id** (*int, optional, default=1*) – impact function id

**set\_sigmoid\_impf**(*intensity*, *L*, *k*, *x0*, *if\_id*=1)

Sigmoid type impact function hinging on three parameter.

This type of impact function is very flexible for any sort of study, hazard and resolution. The sigmoid is defined as:

$$f(x) =$$

$$\frac{L}{1 + \exp\{-k(x - x_0)\}}$$

For more information: [https://en.wikipedia.org/wiki/Logistic\\_function](https://en.wikipedia.org/wiki/Logistic_function)

This method modifies self (climada.entity.impact\_funcs instance) by assigning an id, intensity, mdd and paa to the impact function.

**intensity: tuple(float, float, float)** tuple of 3 intensity numbers along np.arange(min, max, step)

**L** [float] “top” of sigmoid

**k** [float] “slope” of sigmoid

**x0** [float] intensity value where  $f(x) = L/2$

**if\_id** [int, optional, default=1] impact function id

## climada.entity.impact\_funcs.impact\_func\_set module

**class** climada.entity.impact\_funcs.impact\_func\_set.**ImpactFuncSet**

Bases: object

Contains impact functions of type ImpactFunc. Loads from files with format defined in FILE\_EXT.

**tag**

information about the source data

**Type** *Tag*

**\_data**

contains ImpactFunc classes. It's not supposed to be directly accessed. Use the class methods instead.

**Type** dict

**\_\_init\_\_()**

Empty initialization.

**Examples**

Fill impact functions with values and check consistency data:

```
>>> fun_1 = ImpactFunc()
>>> fun_1.haz_type = 'TC'
>>> fun_1.id = 3
>>> fun_1.intensity = np.array([0, 20])
>>> fun_1.paa = np.array([0, 1])
>>> fun_1.mdd = np.array([0, 0.5])
>>> imp_fun = ImpactFuncSet()
>>> imp_fun.append(fun_1)
>>> imp_fun.check()
```

Read impact functions from file and checks consistency data.

```
>>> imp_fun = ImpactFuncSet()
>>> imp_fun.read(ENT_TEMPLATE_XLS)
```

**clear()**

Reinitialize attributes.

**append(func)**

Append a ImpactFunc. Overwrite existing if same id and haz\_type.

**Parameters** **func** (*ImpactFunc*) – ImpactFunc instance

**Raises** **ValueError** –

**remove\_func(haz\_type=None, fun\_id=None)**

Remove impact function(s) with provided hazard type and/or id. If no input provided, all impact functions are removed.

**Parameters**

- **haz\_type** (*str, optional*) – all impact functions with this hazard
- **fun\_id** (*int, optional*) – all impact functions with this id

**get\_func(haz\_type=None, fun\_id=None)**

Get ImpactFunc(s) of input hazard type and/or id. If no input provided, all impact functions are returned.

**Parameters**

- **haz\_type** (*str, optional*) – hazard type
- **fun\_id** (*int, optional*) – ImpactFunc id

**Returns**

- *ImpactFunc* (if *haz\_type* and *fun\_id*),
- *list(ImpactFunc)* (if *haz\_type* or *fun\_id*),

- **{ImpactFunc.haz\_type}** (*{ImpactFunc.id : ImpactFunc}*) (if None))

**get\_hazard\_types**(*fun\_id=None*)

Get impact functions hazard types contained for the id provided. Return all hazard types if no input id.

**Parameters** **fun\_id** (*int, optional*) – id of an impact function

**Returns**

**Return type** list(str)

**get\_ids**(*haz\_type=None*)

Get impact functions ids contained for the hazard type provided. Return all ids for each hazard type if no input hazard type.

**Parameters** **haz\_type** (*str, optional*) – hazard type from which to obtain the ids

**Returns**

- *list(ImpactFunc.id)* (if *haz\_type* provided),
- **{ImpactFunc.haz\_type}** (*list(ImpactFunc.id)*) (if no *haz\_type*)

**size**(*haz\_type=None, fun\_id=None*)

Get number of impact functions contained with input hazard type and /or id. If no input provided, get total number of impact functions.

**Parameters**

- **haz\_type** (*str, optional*) – hazard type
- **fun\_id** (*int, optional*) – ImpactFunc id

**Returns**

**Return type** int

**check()**

Check instance attributes.

**Raises ValueError** –

**extend**(*impact\_funcs*)

Append impact functions of input ImpactFuncSet to current ImpactFuncSet. Overwrite ImpactFunc if same id and haz\_type.

**Parameters** **impact\_funcs** (*ImpactFuncSet*) – ImpactFuncSet instance to extend

**Raises ValueError** –

**plot**(*haz\_type=None, fun\_id=None, axis=None, \*\*kwargs*)

Plot impact functions of selected hazard (all if not provided) and selected function id (all if not provided).

**Parameters**

- **haz\_type** (*str, optional*) – hazard type
- **fun\_id** (*int, optional*) – id of the function

**Returns**

**Return type** matplotlib.axes.\_subplots.AxesSubplot

**read\_excel**(*file\_name, description="", var\_names={'col\_name': {'func\_id': 'impact\_fun\_id', 'inten': 'intensity', 'mdd': 'mdd', 'name': 'name', 'paa': 'paa', 'peril': 'peril\_id', 'unit': 'intensity\_unit'}, 'sheet\_name': 'impact\_functions'})*)

Read excel file following template and store variables.

**Parameters**

- **file\_name** (*str*) – absolute file name
- **description** (*str, optional*) – description of the data
- **var\_names** (*dict, optional*) – name of the variables in the file

```
read_mat(file_name, description="", var_names={'field_name': 'damagefunctions', 'sup_field_name': 'entity',
 'var_name': {'fun_id': 'DamageFunID', 'inten': 'Intensity', 'mdd': 'MDD', 'name': 'name', 'paa':
 'PAA', 'peril': 'peril_ID', 'unit': 'Intensity_unit'}})
```

Read MATLAB file generated with previous MATLAB CLIMADA version.

**Parameters**

- **file\_name** (*str*) – absolute file name
- **description** (*str, optional*) – description of the data
- **var\_names** (*dict, optional*) – name of the variables in the file

```
write_excel(file_name, var_names={'col_name': {'func_id': 'impact_fun_id', 'inten': 'intensity', 'mdd':
 'mdd', 'name': 'name', 'paa': 'paa', 'peril': 'peril_id', 'unit': 'intensity_unit'}, 'sheet_name':
 'impact_functions'})
```

Write excel file following template.

**Parameters**

- **file\_name** (*str*) – absolute file name to write
- **var\_names** (*dict, optional*) – name of the variables in the file

**climada.entity.impact\_funcs.storm\_europe module**

```
class climada.entity.impact_funcs.storm_europe.ImpfStormEurope
```

Bases: [climada.entity.impact\\_funcs.base.ImpactFunc](#)

Impact functions for tropical cyclones.

```
__init__()
```

Empty initialization.

```
set_schwierz(impf_id=1)
```

Using the impact functions of Schwierz et al. 2010, doi:10.1007/s10584-009-9712-1

```
set_welker(Impf_id=1)
```

Using the impact functions of Welker et al. 2020, doi:10.5194/nhess-21-279-2021 It is the Schwierz function, calibrated with a simple multiplicative factor to minimize RMSE between modelled damages and reported damages.

**climada.entity.impact\_funcs.trop\_cyclone module**

```
class climada.entity.impact_funcs.trop_cyclone.ImpfTropCyclone
```

Bases: [climada.entity.impact\\_funcs.base.ImpactFunc](#)

Impact functions for tropical cyclones.

```
__init__()
```

Empty initialization.



```
set_emanuel_usa(impf_id=1, intensity=array([0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
85, 90, 95, 100, 105, 110, 115, 120]), v_thresh=25.7, v_half=74.7, scale=1.0)
```

Using the formula of Emanuele 2011.

#### Parameters

- **impf\_id** (*int, optional*) – impact function id. Default: 1
- **intensity** (*np.array, optional*) – intensity array in m/s. Default: 5 m/s step array from 0 to 120m/s
- **v\_thresh** (*float, optional*) – first shape parameter, wind speed in m/s below which there is no damage. Default: 25.7(Emanuel 2011)
- **v\_half** (*float, optional*) – second shape parameter, wind speed in m/s at which 50% of max. damage is expected. Default: v\_threshold + 49 m/s (mean value of Sealy & Strobl 2017)
- **scale** (*float, optional*) – scale parameter, linear scaling of MDD.  $0 \leq \text{scale} \leq 1$ . Default: 1.0

**Raises** `ValueError` –

### climada.entity.measures package

#### climada.entity.measures.base module

**class** climada.entity.measures.base.Measure

Bases: object

Contains the definition of one measure.

**name**

name of the measure

**Type** str

**haz\_type**

related hazard type (peril), e.g. TC

**Type** str

**color\_rgb**

integer array of size 3. Color code of this measure in RGB

**Type** np.array

**cost**

discounted cost (in same units as assets)

**Type** float

**hazard\_set**

file name of hazard to use (in h5 format)

**Type** str

**hazard\_freq\_cutoff**

hazard frequency cutoff

**Type** float

**exposures\_set**

file name of exposure to use (in h5 format) or Exposure instance

**Type** str or climada.entity.Exposure

**imp\_fun\_map**

change of impact function id of exposures, e.g. '1to3'

**Type** str

**hazard\_inten\_imp**

parameter a and b of hazard intensity change

**Type** tuple(float, float)

**mdd\_impact**

parameter a and b of the impact over the mean damage degree

**Type** tuple(float, float)

**paa\_impact**

parameter a and b of the impact over the percentage of affected assets

**Type** tuple(float, float)

**exp\_region\_id**

region id of the selected exposures to consider ALL the previous parameters

**Type** int

**risk\_transf\_attach**

risk transfer attachment

**Type** float

**risk\_transf\_cover**

risk transfer cover

**Type** float

**risk\_transf\_cost\_factor**

factor to multiply to resulting insurance layer to get the total cost of risk transfer

**Type** float

**\_\_init\_\_()**

Empty initialization.

**check()**

Check consistent instance data.

**Raises ValueError –**

**calc\_impact(exposures, imp\_fun\_set, hazard)**

Apply measure and compute impact and risk transfer of measure implemented over inputs.

**Parameters**

- **exposures** (*climada.entity.Exposures*) – exposures instance
- **imp\_fun\_set** (*climada.entity.ImpactFuncSet*) – impact function set instance
- **hazard** (*climada.hazard.Hazard*) – hazard instance

**Returns**

- *climada.engine.Impact*

- *resulting impact and risk transfer of measure*

**apply**(*exposures*, *imp\_fun\_set*, *hazard*)

Implement measure with all its defined parameters.

#### Parameters

- **exposures** (*climada.entity.Exposures*) – exposures instance
- **imp\_fun\_set** (*climada.entity.ImpactFuncSet*) – impact function set instance
- **hazard** (*climada.hazard.Hazard*) – hazard instance

#### Returns

**new\_exp**, **new\_ifs**, **new\_haz** – **climada.entity.ImpactFuncSet**, **climada.hazard.Hazard**

Exposure, impact function set with implemented measure with all defined parameters.

**Return type** **climada.entity.Exposure**,

### climada.entity.measures.measure\_set module

**class** **climada.entity.measures.measure\_set.MeasureSet**

Bases: **object**

Contains measures of type **Measure**. Loads from files with format defined in **FILE\_EXT**.

#### **tag**

information about the source data

**Type** *Tag*

#### **\_data**

contains **Measure** classes. It's not supposed to be directly accessed. Use the class methods instead.

**Type** **dict**

#### **\_\_init\_\_()**

Empty initialization.

### Examples

Fill **MeasureSet** with values and check consistency data:

```
>>> act_1 = Measure()
>>> act_1.name = 'Seawall'
>>> act_1.color_rgb = np.array([0.1529, 0.2510, 0.5451])
>>> act_1.hazard_intensity = (1, 0)
>>> act_1.mdd_impact = (1, 0)
>>> act_1.paa_impact = (1, 0)
>>> meas = MeasureSet()
>>> meas.append(act_1)
>>> meas.tag.description = "my dummy MeasureSet."
>>> meas.check()
```

Read measures from file and checks consistency data:

```
>>> meas = MeasureSet()
>>> meas.read_excel(ENT_TEMPLATE_XLS)
```

**clear()**

Reinitialize attributes.

**append(*meas*)**

Append an Measure. Override if same name and haz\_type.

**Parameters** *meas* (*Measure*) – Measure instance

**Raises** **ValueError** –

**remove\_measure(*haz\_type=None, name=None*)**

Remove impact function(s) with provided hazard type and/or id. If no input provided, all impact functions are removed.

**Parameters**

- **haz\_type** (*str, optional*) – all impact functions with this hazard
- **name** (*str, optional*) – measure name

**get\_measure(*haz\_type=None, name=None*)**

Get ImpactFunc(s) of input hazard type and/or id. If no input provided, all impact functions are returned.

**Parameters**

- **haz\_type** (*str, optional*) – hazard type
- **name** (*str, optional*) – measure name

**Returns**

- *Measure* (if *haz\_type* and *name*),
- *list(Measure)* (if *haz\_type* or *name*),
- **{Measure.haz\_type (Measure.name : Measure)}** (if *None*)

**get\_hazard\_types(*meas=None*)**

Get measures hazard types contained for the name provided. Return all hazard types if no input name.

**Parameters** *name* (*str, optional*) – measure name

**Returns**

**Return type** *list(str)*

**get\_names(*haz\_type=None*)**

Get measures names contained for the hazard type provided. Return all names for each hazard type if no input hazard type.

**Parameters** *haz\_type* (*str, optional*) – hazard type from which to obtain the names

**Returns**

- *list(Measure.name)* (if *haz\_type* provided),
- **{Measure.haz\_type (list(Measure.name))}** (if no *haz\_type*)

**size(*haz\_type=None, name=None*)**

Get number of measures contained with input hazard type and /or id. If no input provided, get total number of impact functions.

**Parameters**

- **haz\_type** (*str, optional*) – hazard type
- **name** (*str, optional*) – measure name

**Returns****Return type** int**check()**

Check instance attributes.

**Raises ValueError** –**extend(meas\_set)**

Extend measures of input MeasureSet to current MeasureSet. Overwrite Measure if same name and haz\_type.

**Parameters** **impact\_funcs** (*MeasureSet*) – ImpactFuncSet instance to extend**Raises ValueError** –

**read\_mat**(*file\_name*, *description*="", *var\_names*={'field\_name': 'measures', 'sup\_field\_name': 'entity', 'var\_name': {'color': 'color', 'cost': 'cost', 'exp\_reg': 'Region\_ID', 'exp\_set': 'assets\_file', 'fun\_map': 'damagefunctions\_map', 'haz': 'peril\_ID', 'haz\_freq': 'hazard\_high\_frequency\_cutoff', 'haz\_int\_a': 'hazard\_intensity\_impact\_a', 'haz\_int\_b': 'hazard\_intensity\_impact\_b', 'haz\_set': 'hazard\_event\_set', 'mdd\_a': 'MDD\_impact\_a', 'mdd\_b': 'MDD\_impact\_b', 'name': 'name', 'paa\_a': 'PAA\_impact\_a', 'paa\_b': 'PAA\_impact\_b', 'risk\_att': 'risk\_transfer\_attachement', 'risk\_cov': 'risk\_transfer\_cover'}})

Read MATLAB file generated with previous MATLAB CLIMADA version.

**Parameters**

- **file\_name** (*str*) – absolute file name
- **description** (*str*, *optional*) – description of the data
- **var\_names** (*dict*, *optional*) – name of the variables in the file

**read\_excel**(*file\_name*, *description*="", *var\_names*={'col\_name': {'color': 'color', 'cost': 'cost', 'exp\_reg': 'Region\_ID', 'exp\_set': 'assets\_file', 'fun\_map': 'damagefunctions map', 'haz': 'peril\_ID', 'haz\_freq': 'hazard high frequency cutoff', 'haz\_int\_a': 'hazard intensity impact a', 'haz\_int\_b': 'hazard intensity impact b', 'haz\_set': 'hazard event set', 'mdd\_a': 'MDD impact a', 'mdd\_b': 'MDD impact b', 'name': 'name', 'paa\_a': 'PAA impact a', 'paa\_b': 'PAA impact b', 'risk\_att': 'risk transfer attachment', 'risk\_cov': 'risk transfer cover', 'risk\_fact': 'risk transfer cost factor'}, 'sheet\_name': 'measures'})

Read excel file following template and store variables.

**Parameters**

- **file\_name** (*str*) – absolute file name
- **description** (*str*, *optional*) – description of the data
- **var\_names** (*dict*, *optional*) – name of the variables in the file

**write\_excel**(*file\_name*, *var\_names*={'col\_name': {'color': 'color', 'cost': 'cost', 'exp\_reg': 'Region\_ID', 'exp\_set': 'assets\_file', 'fun\_map': 'damagefunctions map', 'haz': 'peril\_ID', 'haz\_freq': 'hazard high frequency cutoff', 'haz\_int\_a': 'hazard intensity impact a', 'haz\_int\_b': 'hazard intensity impact b', 'haz\_set': 'hazard event set', 'mdd\_a': 'MDD impact a', 'mdd\_b': 'MDD impact b', 'name': 'name', 'paa\_a': 'PAA impact a', 'paa\_b': 'PAA impact b', 'risk\_att': 'risk transfer attachment', 'risk\_cov': 'risk transfer cover', 'risk\_fact': 'risk transfer cost factor'}, 'sheet\_name': 'measures'})

Write excel file following template.

**Parameters**

- **file\_name** (*str*) – absolute file name to write

- **var\_names** (*dict, optional*) – name of the variables in the file

### climada.entity.entity\_def module

**class** climada.entity.entity\_def.**Entity**

Bases: object

Collects exposures, impact functions, measures and discount rates. Default values set when empty constructor.

**exposures**

exposures

**Type** *Exposures*

**impact\_funcs**

impact functions

**Type** ImpactFucs

**measures**

measures

**Type** *MeasureSet*

**disc\_rates**

discount rates

**Type** *DiscRates*

**def\_file**

Default file from configuration file

**Type** str

**\_\_init\_\_()**

Empty initializer

**read\_mat**(*file\_name, description=""*)

Read MATLAB file of climada.

**Parameters**

- **file\_name** (*str, optional*) – file name(s) or folder name containing the files to read
- **description** (*str or list(str), optional*) – one description of the data or a description of each data file

**Raises ValueError** –

**read\_excel**(*file\_name, description=""*)

Read csv or xls or xlsx file following climada's template.

**Parameters**

- **file\_name** (*str, optional*) – file name(s) or folder name containing the files to read
- **description** (*str or list(str), optional*) – one description of the data or a description of each data file

**Raises ValueError** –

**write\_excel**(*file\_name*)

Write excel file following template.

**check()**

Check instance attributes.

**Raises** `ValueError` –

### climada.entity.tag module

**class** `climada.entity.tag.Tag`(*file\_name*="", *description*="")

Bases: `object`

Source data tag for Exposures, DiscRates, ImpactFuncSet, MeasureSet.

**file\_name**

name of the source file

**Type** `str`

**description**

description of the data

**Type** `str`

**\_\_init\_\_**(*file\_name*="", *description*="")

Initialize values.

**Parameters**

- **file\_name** (*str, optional*) – file name to read
- **description** (*str, optional*) – description of the data

**append**(*tag*)

Append input Tag instance information to current Tag.

## 7.1.3 climada.hazard package

### climada.hazard.centroids package

#### climada.hazard.centroids.centri module

**class** `climada.hazard.centroids.centri.Centroids`

Bases: `object`

Contains raster or vector centroids.

Raster data can be set with `set_raster_file()` or `set_meta()`. Vector data can be set with `set_lat_lon()` or `set_vector_file()`.

**meta**

rasterio meta dictionary containing raster properties: width, height, crs and transform must be present at least. The affine transformation needs to be shearless (only stretching) and have positive x- and negative y-orientation.

**Type** `dict`, optional

**lat**

latitude of size size

**Type** `np.array`, optional

**lon**  
longitude of size size  
**Type** np.array, optional

**geometry**  
contains lat and lon crs. Might contain geometry points for lat and lon  
**Type** gpd.GeoSeries, optional

**area\_pixel**  
area of size size  
**Type** np.array, optional

**dist\_coast**  
distance to coast of size size  
**Type** np.array, optional

**on\_land**  
on land (True) and on sea (False) of size size  
**Type** np.array, optional

**region\_id**  
country region code of size size  
**Type** np.array, optional

**elevation**  
elevation of size size  
**Type** np.array, optional

**vars\_check** = {'area\_pixel', 'dist\_coast', 'elevation', 'geometry', 'lat', 'lon', 'on\_land', 'region\_id'}  
Variables whose size will be checked

**\_\_init\_\_()**  
Initialize to None raster and vector. Default crs=DEF\_CRS

**check()**  
Check integrity of stored information.  
  
Checks that either *meta* attribute is set, or *lat*, *lon* and *geometry.crs*. Checks sizes of (optional) data attributes.

**equal(centr)**  
Return True if two centroids equal, False otherwise  
**Parameters** **centr** (*Centroids*) – centroids to compare  
**Returns** **eq**  
**Return type** bool

**static from\_base\_grid(land=False, res\_as=360, base\_file=None)**  
Initialize from base grid data provided with CLIMADA  
**Parameters**

- **land** (*bool, optional*) – If True, restrict to grid points on land. Default: False.
- **res\_as** (*int, optional*) – Base grid resolution in arc-seconds (one of 150, 360). Default: 360.



- **base\_file** (*str, optional*) – If set, read this file instead of one provided with climada.

**static from\_geodataframe**(*gdf, geometry\_alias='geom'*)

Create Centroids instance from GeoDataFrame.

The geometry, lat, and lon attributes are set from the GeoDataFrame.geometry attribute, while the columns are copied as attributes to the Centroids object in the form of numpy.ndarrays using pandas.Series.to\_numpy. The Series dtype will thus be respected.

Columns named lat or lon are ignored, as they would overwrite the coordinates extracted from the point features. If the geometry attribute bears an alias, it can be dropped by setting the geometry\_alias parameter.

If the GDF includes a region\_id column, but no on\_land column, then on\_land=True is inferred for those centroids that have a set region\_id.

### Example

```
>>> gdf = geopandas.read_file('centroids.shp')
>>> gdf.region_id = gdf.region_id.astype(int) # type coercion
>>> centroids = Centroids.from_geodataframe(gdf)
```

### Parameters

- **gdf** (*GeoDataFrame*) – Where the geometry column needs to consist of point features. See above for details on processing.
- **geometry\_alias** (*str, opt*) – Alternate name for the geometry column; dropped to avoid duplicate assignment.

**set\_raster\_from\_pix\_bounds**(*xf\_lat, xo\_lon, d\_lat, d\_lon, n\_lat, n\_lon, crs='EPSG:4326'*)

Set raster metadata (meta attribute) from pixel border data

### Parameters

- **xf\_lat** (*float*) – upper latitude (top)
- **xo\_lon** (*float*) – left longitude
- **d\_lat** (*float*) – latitude step (negative)
- **d\_lon** (*float*) – longitude step (positive)
- **n\_lat** (*int*) – number of latitude points
- **n\_lon** (*int*) – number of longitude points
- **crs** (*dict()* or *rasterio.crs.CRS, optional*) – CRS. Default: DEF\_CRS

**set\_raster\_from\_pnt\_bounds**(*points\_bounds, res, crs='EPSG:4326'*)

Set raster metadata (meta attribute) from points border data.

Raster border = point\_border + res/2

### Parameters

- **points\_bounds** (*tuple*) – points' lon\_min, lat\_min, lon\_max, lat\_max
- **res** (*float*) – desired resolution in same units as points\_bounds
- **crs** (*dict()* or *rasterio.crs.CRS, optional*) – CRS. Default: DEF\_CRS

**set\_lat\_lon**(*lat, lon, crs='EPSG:4326'*)

Set Centroids points from given latitude, longitude and CRS.

**Parameters**

- **lat** (*np.array*) – latitude
- **lon** (*np.array*) – longitude
- **crs** (*dict()* or *rasterio.crs.CRS*, *optional*) – CRS. Default: DEF\_CRS

**set\_raster\_file**(*file\_name*, *band=None*, *src\_crs=None*, *window=False*, *geometry=False*, *dst\_crs=False*, *transform=None*, *width=None*, *height=None*, *resampling=Resampling.nearest*)

Read raster of bands and set 0 values to the masked ones.

Each band is an event. Select region using window or geometry. Reproject input by providing *dst\_crs* and/or (*transform*, *width*, *height*).

**Parameters**

- **file\_path** (*str*) – path of the file
- **band** (*list(int)*, *optional*) – band number to read. Default: [1]
- **src\_crs** (*crs*, *optional*) – source CRS. Provide it if error without it.
- **window** (*rasterio.windows.Window*, *optional*) – window to read
- **geometry** (*shapely.geometry*, *optional*) – consider pixels only in shape
- **dst\_crs** (*crs*, *optional*) – reproject to given crs
- **transform** (*rasterio.Affine*) – affine transformation to apply
- **width** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp.Resampling* *optional*) – resampling function used for re-projection to *dst\_crs*

**Raises ValueError** –

**Returns inten** – Each row is an event.

**Return type** *scipy.sparse.csr\_matrix*

**set\_vector\_file**(*file\_name*, *inten\_name=None*, *dst\_crs=None*)

Read vector file format supported by fiona.

Each intensity name is considered an event.

**Parameters**

- **file\_name** (*str*) – vector file with format supported by fiona and ‘geometry’ field.
- **inten\_name** (*list(str)*, *optional*) – list of names of the columns of the intensity of each event. default: [‘intensity’]
- **dst\_crs** (*crs*, *optional*) – reproject to given crs

**Returns inten** – Sparse intensity array of shape (len(inten\_name), len(geometry)).

**Return type** *scipy.sparse.csr\_matrix*

**read\_mat**(*file\_name*, *var\_names=None*)

Read centroids from CLIMADA’s MATLAB version.

**Parameters**

- **file\_name** (*str*) – absolute or relative file name

- **var\_names** (*dict, optional*) – name of the variables

**Raises** **KeyError** –

**read\_excel**(*file\_name, var\_names=None*)

Read centroids from excel file with column names in var\_names.

**Parameters**

- **file\_name** (*str*) – absolute or relative file name
- **var\_names** (*dict, default*) – name of the variables

**Raises** **KeyError** –

**clear**()

Clear vector and raster data.

**append**(*centr*)

Append centroids points.

If centr or self are rasters they are converted to points first using `Centroids.set_meta_to_lat_lon`. Note that self is modified in-place, and meta is set to {}. Thus, raster information in self is lost.

Note: this is a wrapper for `centroids.union`.

**Parameters** **centr** (*Centroids*) – Centroids to append. The centroids need to have the same CRS.

**See also:**

**union** Union of Centroid objects.

**union**(\**others*)

Create the union of centroids from the inputs.

The centroids are combined together point by point. Rasters are converted to points and raster information is lost in the output. All centroids must have the same CRS.

In any case, the attribute `.geometry` is computed for all centroids. This requires a CRS to be defined. If `Centroids.crs` is `None`, the default `DEF_CRS` is set for all centroids (self and others).

When at least one centroids has one of the following property defined, it is also computed for all others. `.area_pixel`, `.dist_coast`, `.on_land`, `.region_id`, `.elevetaion`

!Caution!: the input objects (self and others) are modified in place. Missing properties are added, existing ones are not overwritten.

**Parameters** **others** (*any number of climada.hazard.Centroids()*) – Centroids to form the union with

**Returns** **centroids** – Centroids containing the union of the centroids in others.

**Return type** `climada.hazard.Centroids()`

**Raises** **ValueError** –

**get\_closest\_point**(*x\_lon, y\_lat, scheduler=None*)

Returns closest centroid and its index to a given point.

**Parameters**

- **x\_lon** (*float*) – x coord (lon)
- **y\_lat** (*float*) – y coord (lat)

- **scheduler** (*str*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”

**Returns**

- **x\_close** (*float*) – x-coordinate (longitude) of closest centroid.
- **y\_close** (*float*) – y-coordinate (latitude) of closest centroids.
- **idx\_close** (*int*) – Index of centroid in internal ordering of centroids.

**set\_region\_id**(*scheduler=None*)

Set region\_id as country ISO numeric code attribute for every pixel or point.

**Parameters** **scheduler** (*str*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”

**set\_area\_pixel**(*min\_resol=1e-08, scheduler=None*)

Set area\_pixel attribute for every pixel or point (area in m\*m).

**Parameters**

- **min\_resol** (*float, optional*) – if centroids are points, use this minimum resolution in lat and lon. Default: 1.0e-8
- **scheduler** (*str*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”

**set\_area\_approx**(*min\_resol=1e-08*)

Set area\_pixel attribute for every pixel or point (approximate area in m\*m).

Values are differentiated per latitude. Faster than *set\_area\_pixel*.

**Parameters** **min\_resol** (*float, optional*) – if centroids are points, use this minimum resolution in lat and lon. Default: 1.0e-8

**set\_elevation**(*topo\_path*)

Set elevation attribute for every pixel or point in meters.

**Parameters** **topo\_path** (*str*) – Path to a raster file containing gridded elevation data.

**set\_dist\_coast**(*signed=False, precomputed=False, scheduler=None*)

Set dist\_coast attribute for every pixel or point in meters.

**Parameters**

- **signed** (*bool*) – If True, use signed distances (positive off shore and negative on land). Default: False.
- **precomputed** (*bool*) – If True, use precomputed distances (from NASA). Default: False.
- **scheduler** (*str*) – Used for dask map\_partitions. “threads”, “synchronous” or “processes”

**set\_on\_land**(*scheduler=None*)

Set on\_land attribute for every pixel or point.

**Parameters** **scheduler** (*str*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”

**remove\_duplicate\_points**(*scheduler=None*)

Return Centroids with removed duplicated points

**Parameters** **scheduler** (*str*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”

**Returns** `cen` – Sub-selection of this object.

**Return type** *Centroids*

**select**(*reg\_id=None, extent=None, sel\_cen=None*)

Return Centroids with points in the given *reg\_id* or within mask

**Parameters**

- **reg\_id** (*int*) – region to filter according to *region\_id* values
- **extent** (*tuple*) – Format (*min\_lon, max\_lon, min\_lat, max\_lat*) tuple. If *min\_lon* > *lon\_max*, the extend crosses the antimeridian and is [*lon\_max*, 180] + [-180, *lon\_min*] Borders are inclusive.
- **sel\_cen** (*np.array*) – 1-dim mask, overrides *reg\_id* and *extent*

**Returns** `cen` – Sub-selection of this object

**Return type** *Centroids*

**set\_lat\_lon\_to\_meta**(*min\_resol=1e-08*)

Compute meta from lat and lon values.

**Parameters** **min\_resol** (*float, optional*) – Minimum centroids resolution to use in the raster.  
Default: 1.0e-8.

**set\_meta\_to\_lat\_lon**()

Compute lat and lon of every pixel center from meta raster.

**plot**(*axis=None, figsize=(9, 13), \*\*kwargs*)

Plot centroids scatter points over earth.

**Parameters**

- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*((float, float), optional)*) – figure size for *plt.subplots* The default is (9, 13)
- **kwargs** (*optional*) – arguments for scatter *matplotlib* function

**Returns** `axis`

**Return type** *matplotlib.axes.\_subplots.AxesSubplot*

**calc\_pixels\_polygons**(*scheduler=None*)

Return a *gpd.GeoSeries* with a polygon for every pixel

**Parameters** **scheduler** (*str*) – used for *dask map\_partitions*. “threads”, “synchronous” or “processes”

**Returns** `geo`

**Return type** *gpd.GeoSeries*

**empty\_geometry\_points**()

Removes all points in geometry.

Useful when centroids is used in multiprocessing function.

**write\_hdf5**(*file\_data*)

Write centroids attributes into *hdf5* format.

**Parameters** **file\_data** (*str or h5*) – If string, path to write data. If *h5* object, the datasets will be generated there.

**read\_hdf5**(*file\_data*)

Read centroids attributes from hdf5.

**Parameters** *file\_data* (*str* or *h5*) – If string, path to read data. If h5 object, the datasets will be read from there.

**property crs**

Get CRS of raster or vector.

**property size**

Get number of pixels or points.

**property shape**

Get shape of rastered data.

**property total\_bounds**

Get total bounds (left, bottom, right, top).

**property coord**

Get [lat, lon] array.

**set\_geometry\_points**(*scheduler=None*)

Set *geometry* attribute with Points from *lat/lon* attributes.

**Parameters** *scheduler* (*str*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”

## climada.hazard.base module

**class** climada.hazard.base.**Hazard**(*haz\_type=""*, *pool=None*)

Bases: object

Contains events of some hazard type defined at centroids. Loads from files with format defined in FILE\_EXT.

**tag**

information about the source

**Type** TagHazard

**units**

units of the intensity

**Type** str

**centroids**

centroids of the events

**Type** *Centroids*

**event\_id**

id (>0) of each event

**Type** np.array

**event\_name**

name of each event (default: event\_id)

**Type** list(str)

**date**

integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)

**Type** np.array

**orig**

flags indicating historical events (True) or probabilistic (False)

**Type** np.array

**frequency**

frequency of each event in years

**Type** np.array

**intensity**

intensity of the events at centroids

**Type** sparse.csr\_matrix

**fraction**

fraction of affected exposures for each event at each centroid

**Type** sparse.csr\_matrix

**intensity\_thres = 10**

Intensity threshold per hazard used to filter lower intensities. To be set for every hazard type

**vars\_oblig** = {'centroids', 'event\_id', 'fraction', 'frequency', 'intensity', 'tag', 'units'}

scalar, str, list, 1dim np.array of size num\_events, scipy.sparse matrix of shape num\_events x num\_centroids, Centroids and Tag.

**Type** Name of the variables needed to compute the impact. Types

**vars\_def** = {'date', 'event\_name', 'orig'}

Name of the variables used in impact calculation whose value is descriptive and can therefore be set with default values. Types: scalar, string, list, 1dim np.array of size num\_events.

**vars\_opt** = {}

Name of the variables that aren't need to compute the impact. Types: scalar, string, list, 1dim np.array of size num\_events.

**\_\_init\_\_**(haz\_type="", pool=None)

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

**Examples**

Fill hazard values by hand:

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

Take hazard values from file:

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

**clear()**

Reinitialize attributes (except the process Pool).

**check()**

Check dimension of attributes.

**Raises ValueError** –

**set\_raster**(*files\_intensity*, *files\_fraction*=None, *attrs*=None, *band*=None, *src\_crs*=None, *window*=False, *geometry*=False, *dst\_crs*=False, *transform*=None, *width*=None, *height*=None, *resampling*=Resampling.nearest)

Set intensity and fraction to values from raster files

If raster files are masked, the masked values are set to 0.

Files can be partially read using either window or geometry. Additionally, the data is reprojected when custom *dst\_crs* and/or transform, width and height are specified.

**Parameters**

- **files\_intensity** (*list(str)*) – file names containing intensity
- **files\_fraction** (*list(str)*) – file names containing fraction
- **attrs** (*dict, optional*) – name of Hazard attributes and their values
- **band** (*list(int), optional*) – bands to read (starting at 1), default [1]
- **src\_crs** (*crs, optional*) – source CRS. Provide it if error without it.
- **window** (*rasterio.windows.Windows, optional*) – window where data is extracted
- **geometry** (*shapely.geometry, optional*) – consider pixels only in shape
- **dst\_crs** (*crs, optional*) – reproject to given crs
- **transform** (*rasterio.Affine*) – affine transformation to apply
- **width** (*float, optional*) – number of lons for transform
- **height** (*float, optional*) – number of lats for transform
- **resampling** (*rasterio.warp.Resampling, optional*) – resampling function used for re-projection to *dst\_crs*

**set\_vector**(*files\_intensity*, *files\_fraction*=None, *attrs*=None, *inten\_name*=None, *frac\_name*=None, *dst\_crs*=None)

Read vector files format supported by fiona. Each intensity name is considered an event.

**Parameters**

- **files\_intensity** (*list(str)*) – file names containing intensity, default: ['intensity']
- **files\_fraction** (*list(str)*) (*file names containing fraction,*) – default: ['fraction']
- **attrs** (*dict, optional*) – name of Hazard attributes and their values
- **inten\_name** (*list(str), optional*) – name of variables containing the intensities of each event
- **frac\_name** (*list(str), optional*) – name of variables containing the fractions of each event
- **dst\_crs** (*crs, optional*) – reproject to given crs



**reproject\_raster**(*dst\_crs=False, transform=None, width=None, height=None, resampl\_inten=Resampling.nearest, resampl\_fract=Resampling.nearest*)

Change current raster data to other CRS and/or transformation

#### Parameters

- **dst\_crs** (*crs, optional*) – reproject to given crs
- **transform** (*rasterio.Affine*) – affine transformation to apply
- **width** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampl\_inten** (*rasterio.warp.Resampling optional*) – resampling function used for reprojection to dst\_crs for intensity
- **resampl\_fract** (*rasterio.warp.Resampling optional*) – resampling function used for reprojection to dst\_crs for fraction

**reproject\_vector**(*dst\_crs, scheduler=None*)

Change current point data to a given projection

#### Parameters

- **dst\_crs** (*crs*) – reproject to given crs
- **scheduler** (*str, optional*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”

**raster\_to\_vector**()

Change current raster to points (center of the pixels)

**vector\_to\_raster**(*scheduler=None*)

Change current point data to a raster with same resolution

**Parameters scheduler** (*str, optional*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”

**read\_mat**(*file\_name, description="", var\_names=None*)

Read climada hazard generate with the MATLAB code.

#### Parameters

- **file\_name** (*str*) – absolute file name
- **description** (*str, optional*) – description of the data
- **var\_names** (*dict, optional*) – name of the variables in the file, default: DEF\_VAR\_MAT constant

**Raises KeyError** –

**read\_excel**(*file\_name, description="", var\_names=None*)

Read climada hazard generate with the MATLAB code.

#### Parameters

- **file\_name** (*str*) – absolute file name
- **description** (*str, optional*) – description of the data
- **centroids** (*Centroids, optional*) – provide centroids if not contained in the file
- **var\_names** (**dict, default**) (*name of the variables in the file,*) – default: DEF\_VAR\_EXCEL constant

**Raises `KeyError`** –

**`select`**(*event\_names=None, date=None, orig=None, reg\_id=None, reset\_frequency=False*)  
Select events matching provided criteria

The frequency of events may need to be recomputed (see *reset\_frequency*)!

**Parameters**

- **`event_names`** (*list of str, optional*) – Names of events.
- **`date`** (*array-like of length 2 containing str or int, optional*) – (initial date, final date) in string ISO format ('2011-01-02') or datetime ordinal integer.
- **`orig`** (*bool, optional*) – Select only historical (True) or only synthetic (False) events.
- **`reg_id`** (*int, optional*) – Region identifier of the centroids' `region_id` attribute.
- **`reset_frequency`** (*bool, optional*) – Change frequency of events proportional to difference between first and last year (old and new). Default: False.

**Returns `haz`** – If no event matching the specified criteria is found, None is returned.

**Return type** *Hazard* or None

**`local_exceedance_inten`**(*return\_periods=(25, 50, 100, 250)*)  
Compute exceedance intensity map for given return periods.

**Parameters** **`return_periods`** (*np.array*) – return periods to consider

**Returns `inten_stats`**

**Return type** *np.array*

**`plot_rp_intensity`**(*return\_periods=(25, 50, 100, 250), smooth=True, axis=None, figsize=(9, 13), adapt\_fontsize=True, \*\*kwargs*)

Compute and plot hazard exceedance intensity maps for different return periods. Calls `local_exceedance_inten`.

**Parameters**

- **`return_periods`** (*tuple(int), optional*) – return periods to consider
- **`smooth`** (*bool, optional*) – smooth plot to `plot.RESOLUTIONxplot.RESOLUTION`
- **`axis`** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **`figsize`** (*tuple, optional*) – figure size for `plt.subplots`
- **`kwargs`** (*optional*) – arguments for `pcolormesh` matplotlib function used in event plots

**Returns `axis, inten_stats`** – `intenstats` is `return_periods.size x num_centroids`

**Return type** *matplotlib.axes.\_subplots.AxesSubplot, np.ndarray*

**`plot_intensity`**(*event=None, centr=None, smooth=True, axis=None, adapt\_fontsize=True, \*\*kwargs*)

Plot intensity values for a selected event or centroid.

**Parameters**

- **`event`** (*int or str, optional*) – If `event > 0`, plot intensities of event with `id = event`. If `event = 0`, plot maximum intensity in each centroid. If `event < 0`, plot `abs(event)`-largest event. If event is string, plot events with that name.
- **`centr`** (*int or tuple, optional*) – If `centr > 0`, plot intensity of all events at centroid with `id = centr`. If `centr = 0`, plot maximum intensity of each event. If `centr < 0`, plot

abs(centr)-largest centroid where higher intensities are reached. If tuple with (lat, lon) plot intensity of nearest centroid.

- **smooth** (*bool, optional*) – Rescale data to RESOLUTIONxRESOLUTION pixels (see constant in module *climada.util.plot*)
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for pcolormesh matplotlib function used in event plots or for plot function used in centroids plots

#### Returns

**Return type** matplotlib.axes.\_subplots.AxesSubplot

**Raises** **ValueError** –

**plot\_fraction**(*event=None, centr=None, smooth=True, axis=None, \*\*kwargs*)

Plot fraction values for a selected event or centroid.

#### Parameters

- **event** (*int or str, optional*) – If event > 0, plot fraction of event with id = event. If event = 0, plot maximum fraction in each centroid. If event < 0, plot abs(event)-largest event. If event is string, plot events with that name.
- **centr** (*int or tuple, optional*) – If centr > 0, plot fraction of all events at centroid with id = centr. If centr = 0, plot maximum fraction of each event. If centr < 0, plot abs(centr)-largest centroid where highest fractions are reached. If tuple with (lat, lon) plot fraction of nearest centroid.
- **smooth** (*bool, optional*) – Rescale data to RESOLUTIONxRESOLUTION pixels (see constant in module *climada.util.plot*)
- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for pcolormesh matplotlib function used in event plots or for plot function used in centroids plots

#### Returns

**Return type** matplotlib.axes.\_subplots.AxesSubplot

**Raises** **ValueError** –

**sanitize\_event\_ids**()

Make sure that event ids are unique

**get\_event\_id**(*event\_name*)

Get an event id from its name. Several events might have the same name.

**Parameters** **event\_name** (*str*) – Event name

**Returns** **list\_id**

**Return type** np.array(int)

**get\_event\_name**(*event\_id*)

Get the name of an event id.

**Parameters** **event\_id** (*int*) – id of the event

**Returns**

**Return type** str

**Raises** **ValueError** –

**get\_event\_date**(*event=None*)

Return list of date strings for given event or for all events, if no event provided.

**Parameters** *event* (*str or int, optional*) – event name or id.

**Returns** *l\_dates*

**Return type** list(str)

**calc\_year\_set**()

From the dates of the original events, get number yearly events.

**Returns** *orig\_yearset* – key are years, values array with event\_ids of that year

**Return type** dict

**remove\_duplicates**()

Remove duplicate events (events with same name and date).

**set\_frequency**(*yearrange=None*)

Set hazard frequency from yearrange or intensity matrix.

**Parameters** *yearrange* (*tuple or list, optional*) – year range to be used to compute frequency per event. If yearrange is not given (None), the year range is derived from self.date

**property size**

Return number of events.

**write\_raster**(*file\_name, intensity=True*)

Write intensity or fraction as GeoTIFF file. Each band is an event

**Parameters**

- **file\_name** (*str*) – file name to write in tif format
- **intensity** (*bool*) – if True, write intensity, otherwise write fraction

**write\_hdf5**(*file\_name, todense=False*)

Write hazard in hdf5 format.

**Parameters** *file\_name* (*str*) – file name to write, with h5 format

**read\_hdf5**(*file\_name*)

Read hazard in hdf5 format.

**Parameters** *file\_name* (*str*) – file name to read, with h5 format

**append**(\**others*)

Append the events and centroids to this hazard object.

All of the given hazards must be of the same type and use the same units as self. The centroids of all hazards must have the same CRS.

The following kinds of object attributes are processed:

- All centroids are combined together using *Centroids.union*.
- Lists, 1-dimensional arrays (NumPy) and sparse CSR matrices (SciPy) are concatenated.

Sparse matrices are concatenated along the first (vertical) axis.

- All *tag* attributes are appended to *self.tag*.

For any other type of attribute: A ValueError is raised if an attribute of that name is not defined in all of the non-empty hazards at least. However, there is no check that the attribute value is identical among the given hazard objects. The initial attribute value of *self* will not be modified.

Note: Each of the hazard's *centroids* attributes might be modified in place in the sense that missing properties are added, but existing ones are not overwritten. In case of raster centroids, conversion to point centroids is applied so that raster information (meta) is lost. For more information, see *Centroids.union*.

**Parameters** *others* (one or more *climada.hazard.Hazard* objects) – Hazard instances to append to self

**Raises** *TypeError*, *ValueError* –

See also:

*Hazard.concat* concatenate 2 or more hazards

*Centroids.union* combine centroids

**classmethod** *concat*(*haz\_list*)

Concatenate events of several hazards of same type.

This function creates a new hazard of the same class as the first hazard in the given list and then applies the *append* method. Please refer to the docs of *Hazard.append* for caveats and limitations of the concatenation procedure.

For centroids, tags, lists, arrays and sparse matrices, the remarks in *Hazard.append* apply. All other attributes are copied from the first object in *haz\_list*.

Note that *Hazard.concat* can be used to concatenate hazards of a subclass. The result's type will be the subclass. However, calling *concat([])* (with an empty list) is equivalent to instantiation without init parameters. So, *Hazard.concat([])* is equivalent to *Hazard()*. If *HazardB* is a subclass of *Hazard*, then *HazardB.concat([])* is equivalent to *HazardB()* (unless *HazardB* overrides the *concat* method).

**Parameters** *haz\_list* (list of *climada.hazard.Hazard* objects) – Hazard instances of the same hazard type (subclass).

**Returns** *haz\_concat* – This will be of the same type (subclass) as all the hazards in *haz\_list*.

**Return type** instance of *climada.hazard.Hazard*

See also:

*Hazard.append* append hazards to a hazard in place

*Centroids.union* combine centroids

**change\_centroids**(*centroids*, *threshold=100*)

Assign (new) centroids to hazard.

Centroids of the hazard not in centroids are mapped onto the nearest point. Fails if a point is further than threshold from the closest centroid.

The centroids must have the same CRS as *self.centroids*.

**Parameters**

- **haz** (*climada.hazard.Hazard()*) – Hazard instance
- **centroids** (*climada.hazard.Centroids()*) – Centroids instance on which to map the hazard.
- **threshold** (*int or float*) – Threshold for mapping *haz.centroids* not in *centroids*. Argument is passed to *climada.util.coordinates.assign\_coordinates*. Default is 100.

**Returns** *haz\_new\_cent* – Hazard projected onto centroids

**Return type** *climada.hazard.Hazard* object

Raises `ValueError` –

See also:

`util.coordinates.assign_coordinates` algorithm to match centroids.

### `climada.hazard.isimip_data` module

### `climada.hazard.storm_europe` module

**class** `climada.hazard.storm_europe.StormEurope`

Bases: `climada.hazard.base.Hazard`

A hazard set containing european winter storm events. Historic storm events can be downloaded at <http://wisc.climate.copernicus.eu/> and read `read_footprints()`. Weather forecasts can be automatically downloaded from <https://opendata.dwd.de/> and read with `read_icon_grib()`. Weather forecast from the COSMO-Consortium <http://www.cosmo-model.org/> can be read with `read_cosmoe_file()`.

#### **ssi\_wisc**

Storm Severity Index (SSI) as recorded in the footprint files; apparently not reproducible from the footprint values only.

**Type** `np.array, float`

#### **ssi**

SSI as set by `set_ssi`; uses the Dawkins definition by default.

**Type** `np.array, float`

#### **intensity\_thres = 14.7**

Intensity threshold for storage in m/s; same as used by WISC SSI calculations.

#### **vars\_opt = {'ssi', 'ssi\_full\_area', 'ssi\_wisc'}**

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

#### **\_\_init\_\_()**

Calls the Hazard init dunder. Sets unit to 'm/s'.

#### **read\_footprints**(*path, description=None, ref\_raster=None, centroids=None, files\_omit='fp\_era20c\_1990012515\_701\_0.nc', combine\_threshold=None*)

Clear instance and read WISC footprints into it. Read Assumes that all footprints have the same coordinates as the first file listed/first file in dir.

#### **Parameters**

- **path** (*str, list(str)*) – A location in the filesystem. Either a path to a single netCDF WISC footprint, or a folder containing only footprints, or a globbing pattern to one or more footprints.
- **description** (*str, optional*) – description of the events, defaults to 'WISC historical hazard set'
- **ref\_raster** (*str, optional*) – Reference netCDF file from which to construct a new bare-bones Centroids instance. Defaults to the first file in path.
- **centroids** (*Centroids, optional*) – A Centroids struct, overriding `ref_raster`
- **files\_omit** (*str, list(str), optional*) – List of files to omit; defaults to one duplicate storm present in the WISC set as of 2018-09-10.

- **combine\_threshold** (*int, optional*) – threshold for combining events in number of days. if the difference of the dates (`self.date`) of two events is smaller or equal to this threshold, the two events are combined into one. Default is None, Advised for WISC is 2

**read\_cosmoe\_file**(*fp\_file, run\_datetime, event\_date=None, model\_name='COSMO-2E', description=None*)

Clear instance and read gust footprint from weather forecast into it. The function is designed for the COSMO ensemble model used by the COSMO Consortium <http://www.cosmo-model.org/> and postprocessed to a netcdf file using fieldextra. One event is one full day in UTC. Works for MeteoSwiss model output of COSMO-1E (11 members, resolution 1.1 km, forecast period 33-45 hours) COSMO-2E (21 members, resolution 2.2 km, forecast period 5 days)

#### Parameters

- **fp\_file** (*str*) – string directing to one netcdf file
- **run\_datetime** (*datetime*) – The starting timepoint of the forecast run of the cosmo model
- **event\_date** (*datetime, optional*) – one day within the forecast period, only this day (00H-24H) will be included in the hazard
- **model\_name** (*str, optional*) – provide the name of the COSMO model, for the description (e.g., 'COSMO-1E', 'COSMO-2E')
- **description** (*str, optional*) – description of the events, defaults to a combination of model\_name and run\_datetime

**read\_icon\_grib**(*run\_datetime, event\_date=None, model\_name='icon-eu-eps', description=None, grib\_dir=None, delete\_raw\_data=True*)

Clear instance and download and read dwd icon weather forecast footprints into it. New files are available for 24 hours on <https://opendata.dwd.de>, old files can be processed if they are already stored in grib\_dir. One event is one full day in UTC. Current setup works for runs starting at 00H and 12H. Otherwise the aggregation is inaccurate, because of the given file structure with 1-hour, 3-hour and 6-hour maxima provided.

#### Parameters

- **run\_datetime** (*datetime*) – The starting timepoint of the forecast run of the icon model
- **event\_date** (*datetime, optional*) – one day within the forecast period, only this day (00H-24H) will be included in the hazard
- **model\_name** (*str, optional*) – select the name of the icon model to be downloaded. Must match the url on <https://opendata.dwd.de> (see download\_icon\_grib for further info)
- **description** (*str, optional*) – description of the events, defaults to a combination of model\_name and run\_datetime
- **grib\_dir** (*str, optional*) – path to folder, where grib files are or should be stored
- **delete\_raw\_data** (*bool, optional*) – select if downloaded raw data in .grib.bz2 file format should be stored on the computer or removed

**calc\_ssi**(*method='dawkins', intensity=None, on\_land=True, threshold=None, sel\_cen=None*)

Calculate the SSI, method must either be 'dawkins' or 'wisc\_gust'.

'dawkins', after Dawkins et al. (2016), doi:10.5194/nhess-16-1999-2016, matches the MATLAB version.  

$$ssi = \sum_i (area\_cell\_i * intensity\_cell\_i^3)$$

‘wisc\_gust’, according to the WISC Tier 1 definition found at [https://wisc.climate.copernicus.eu/wisc/#/help/products#tier1\\_section](https://wisc.climate.copernicus.eu/wisc/#/help/products#tier1_section)  $ssi = \text{sum}(\text{area\_on\_land}) * \text{mean}(\text{intensity})^3$

In both definitions, only raster cells that are above the threshold are used in the computation. Note that this method does not reproduce `self.ssi_wisc`, presumably because the footprint only contains the maximum wind gusts instead of the sustained wind speeds over the 72 hour window. The deviation may also be due to differing definitions of what lies on land (i.e. Syria, Russia, Northern Africa and Greenland are exempt).

#### Parameters

- **method** (*str*) – Either ‘dawkins’ or ‘wisc\_gust’
- **intensity** (*scipy.sparse.csr*) – Intensity matrix; defaults to `self.intensity`
- **on\_land** (*bool*) – Only calculate the SSI for areas on land, ignoring the intensities at sea. Defaults to `true`, whereas the MATLAB version did not.
- **threshold** (*float, optional*) – Intensity threshold used in index definition. Cannot be lower than the read-in value.
- **sel\_cen** (*np.array, bool*) – A boolean vector selecting centroids. Takes precedence over `on_land`.

`self.ssi_dawkins`

SSI per event

**Type** `np.array`

`set_ssi(**kwargs)`

Wrapper around `calc_ssi` for setting the `self.ssi` attribute.

**Parameters** `kwargs` – passed on to `calc_ssi`

`ssi`

SSI per event

**Type** `np.array`

`plot_ssi(full_area=False)`

**Plot the distribution of SSIs versus their cumulative exceedance** frequencies, highlighting historical storms in red.

#### Returns

- **fig** (*matplotlib.figure.Figure*)
- **ax** (*matplotlib.axes.\_subplots.AxesSubplot*)

`generate_prob_storms(reg_id=528, spatial_shift=4, ssi_args=None, **kwargs)`

Generates a new hazard set with one original and 29 probabilistic storms per historic storm. This represents a partial implementation of the Monte-Carlo method described in section 2.2 of Schwierz et al. (2010), doi:10.1007/s10584-009-9712-1. It omits the rotation of the storm footprints, as well as the pseudo-random alterations to the intensity.

In a first step, the original intensity and five additional intensities are saved to an array. In a second step, those 6 possible intensity levels are shifted by `n` raster pixels into each direction (N/S/E/W).

#### Caveats:

- Memory safety is an issue; trial with the entire dataset resulted in 60GB of swap memory being used...



- Can only use numeric region\_id for country selection
- Drops event names as provided by WISC

#### Parameters

- **region\_id** (*int, list of ints, or None*) – iso\_n3 code of the countries we want the generated hazard set to be returned for.
- **spatial\_shift** (*int*) – amount of raster pixels to shift by
- **ssi\_args** (*dict*) – A dictionary of arguments passed to calc\_ssi
- **kwargs** – keyword arguments passed on to self.\_hist2prob()

**Returns** **new\_haz** – A new hazard set for the given country. Centroid attributes are preserved. self.orig attribute is set to True for original storms (event\_id ending in 00). Also contains a ssi\_prob attribute,

**Return type** *StormEurope*

### climada.hazard.tag module

**class** climada.hazard.tag.Tag(*haz\_type="", file\_name="", description=""*)

Bases: object

Contain information used to tag a Hazard.

#### file\_name

name of the source file(s)

**Type** str or list(str)

#### haz\_type

acronym defining the hazard type (e.g. ‘TC’)

**Type** str

#### description

description(s) of the data

**Type** str or list(str)

**\_\_init\_\_**(*haz\_type="", file\_name="", description=""*)

Initialize values.

#### Parameters

- **haz\_type** (*str, optional*) – acronym of the hazard type (e.g. ‘TC’).
- **file\_name** (*str or list(str), optional*) – file name(s) to read
- **description** (*str or list(str), optional*) – description of the data

#### append(*tag*)

Append input Tag instance information to current Tag.

#### join\_file\_names()

Get a string with the joined file names.

#### join\_descriptions()

Get a string with the joined descriptions.

### climada.hazard.tc\_clim\_change module

```
climada.hazard.tc_clim_change.TOT_RADIATIVE_FORCE =
PosixPath('/home/docs/climada/data/rcp_db.xls')
//www.iiasa.ac.at/web-apps/tnt/RcpDb. generated: 2018-07-04 10:47:59.
```

**Type** © RCP Database (Version 2.0.5) http

```
climada.hazard.tc_clim_change.get_knutson_criterion()
```

Fill changes in TCs according to Knutson et al. 2015 Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios.

**Returns criterion** – list of the criterion dictionary for frequency and intensity change per basin, per category taken from the Table 3 in Knutson et al. 2015. with items 'basin' (str), 'category' (list(int)), 'year' (int), 'change' (float), 'variable' ('intensity' or 'frequency')

**Return type** list(dict)

```
climada.hazard.tc_clim_change.calc_scale_knutson(ref_year=2050, rcp_scenario=45)
```

Comparison 2081-2100 (i.e., late twenty-first century) and 2001-20 (i.e., present day). Late twenty-first century effects on intensity and frequency per Saffir-Simpson-category and ocean basin is scaled to target year and target RCP proportional to total radiative forcing of the respective RCP and year.

**Parameters**

- **ref\_year** (*int, optional*) – year between 2000 ad 2100. Default: 2050
- **rcp\_scenario** (*int, optional*) – 26 for RCP 2.6, 45 for RCP 4.5. The default is 45 60 for RCP 6.0 and 85 for RCP 8.5.

**Returns factor** – factor to scale Knutson parameters to the give RCP and year

**Return type** float

### climada.hazard.tc\_tracks module

```
climada.hazard.tc_tracks.CAT_NAMES = {-1: 'Tropical Depression', 0: 'Tropical Storm', 1:
'Hurricane Cat. 1', 2: 'Hurricane Cat. 2', 3: 'Hurricane Cat. 3', 4: 'Hurricane Cat. 4',
5: 'Hurricane Cat. 5'}
```

Saffir-Simpson category names.

```
climada.hazard.tc_tracks.SAFFIR_SIM_CAT = [34, 64, 83, 96, 113, 137, 1000]
```

Saffir-Simpson Hurricane Wind Scale in kn based on NOAA

```
class climada.hazard.tc_tracks.TCTracks(pool=None)
```

Bases: object

Contains tropical cyclone tracks.

**data**

**List of tropical cyclone tracks. Each track contains following attributes:**

- time (coords)
- lat (coords)
- lon (coords)
- time\_step (in hours)
- radius\_max\_wind (in nautical miles)
- radius\_oci (in nautical miles)

- `max_sustained_wind` (in knots)
- `central_pressure` (in hPa/mbar)
- `environmental_pressure` (in hPa/mbar)
- `basin` (for each track position)
- `max_sustained_wind_unit` (attrs)
- `central_pressure_unit` (attrs)
- `name` (attrs)
- `sid` (attrs)
- `orig_event_flag` (attrs)
- `data_provider` (attrs)
- `id_no` (attrs)
- `category` (attrs)

**Computed during processing:**

- `on_land` (bool for each track position)
- `dist_since_lf` (in km)

**Type** `list(xarray.Dataset)`

**`__init__`** (*pool=None*)

Empty constructor. Read csv IBTrACS files if provided.

**`append`** (*tracks*)

Append tracks to current.

**Parameters** `tracks` (*xarray.Dataset or list(xarray.Dataset)*) – tracks to append.

**`get_track`** (*track\_name=None*)

Get track with provided name.

Returns the first matching track based on the assumption that no other track with the same name or sid exists in the set.

**Parameters** `track_name` (*str, optional*) – Name or sid (ibtracsID for IBTrACS) of track. If None (default), return all tracks.

**Returns result** – Usually, a single track is returned. If no track with the specified name is found, an empty list `[]` is returned. If called with *track\_name=None*, the list of all tracks is returned.

**Return type** `xarray.Dataset or list of xarray.Dataset`

**`subset`** (*filterdict*)

Subset tracks based on track attributes.

Select all tracks matching exactly the given attribute values.

**Parameters** `filterdict` (*dict or OrderedDict*) – Keys are attribute names, values are the corresponding attribute values to match. In case of an ordered dict, the filters are applied in the given order.

**Returns** `tc_tracks` – A new instance of `TCTracks` containing only the matching tracks.

**Return type** *TCTracks*

**tracks\_in\_exp**(*exposure*, *buffer*=1.0)

Select only the tracks that are in the vicinity (*buffer*) of an exposure.

Each exposure point/geometry is extended to a disc of radius *buffer*. Each track is converted to a line and extended by a radius *buffer*.

#### Parameters

- **exposure** (*Exposure*) – Exposure used to select tracks.
- **buffer** (*float, optional*) – Size of buffer around exposure geometries (in the units of *exposure.crs*), see *geopandas.distance*. Default: 1.0

**Returns** **filtered\_tracks** – TCTracks object with tracks from *tc\_tracks* intersecting the exposure within a buffer distance.

**Return type** *climada.hazard.TCTracks()*

**read\_ibtracs\_netcdf**(*provider*=None, *rescale\_windspeeds*=True, *storm\_id*=None, *year\_range*=None, *basin*=None, *genesis\_basin*=None, *interpolate\_missing*=True, *estimate\_missing*=False, *correct\_pres*=False, *discard\_single\_points*=True, *file\_name*='IBTrACS.ALL.v04r00.nc')

Read track data from IBTrACS database.

When using data from IBTrACS, make sure to be familiar with the scope and limitations of IBTrACS, e.g. by reading the official documentation ([https://www.ncdc.noaa.gov/ibtracs/pdf/IBTrACS\\_version4\\_Technical\\_Details.pdf](https://www.ncdc.noaa.gov/ibtracs/pdf/IBTrACS_version4_Technical_Details.pdf)). Reading the CLIMADA documentation can't replace a thorough understanding of the underlying data. This function only provides a (hopefully useful) interface for the data input, but cannot provide any guidance or make recommendations about if and how to use IBTrACS data for your particular project.

Resulting tracks are required to have both pressure and wind speed information at all time steps. Therefore, all track positions where one of wind speed or pressure are missing are discarded unless one of *interpolate\_missing* or *estimate\_missing* are active.

Some corrections are automatically applied, such as: *environmental\_pressure* is enforced to be larger than *central\_pressure*.

Note that the tracks returned by this function might contain irregular time steps since that is often the case for the original IBTrACS records. Apply the *equal\_timestep* function afterwards to enforce regular time steps.

#### Parameters

- **provider** (*str or list of str, optional*) – Either specify an agency, such as “usa”, “newdelhi”, “bom”, “cma”, “tokyo”, or the special values “official” and “official\_3h”:
  - \* “official” means using the (usually 6-hourly) officially reported values of the officially responsible agencies.
  - “official\_3h” means to include (inofficial) 3-hourly data of the officially responsible agencies (whenever available).

If you want to restrict to the officially reported values by the officially responsible agencies (*provider*=“official”) without any modifications to the original official data, make sure to also set *estimate\_missing*=False and *interpolate\_missing*=False. Otherwise, gaps in the official reporting will be filled using interpolation and/or statistical estimation procedures (see below). If a list is given, the following logic is applied: For each storm, the variables that are not reported by the first agency for this storm are taken from the next agency in the list that did report this variable for this storm. For different storms, the same variable might be taken from different agencies. Default:

['official\_3h', 'usa', 'tokyo', 'newdelhi', 'reunion', 'bom', 'nadi', 'wellington', 'cma', 'hko', 'ds824', 'td9636', 'td9635', 'neumann', 'mlc']

- **rescale\_windspeeds** (*bool, optional*) – If True, all wind speeds are linearly rescaled to 1-minute sustained winds. Note however that the IBTrACS documentation (Section 5.2, [https://www.ncdc.noaa.gov/ibtracs/pdf/IBTrACS\\_version4\\_Technical\\_Details.pdf](https://www.ncdc.noaa.gov/ibtracs/pdf/IBTrACS_version4_Technical_Details.pdf)) includes a warning about this kind of conversion: “While a multiplicative factor can describe the numerical differences, there are procedural and observational differences between agencies that can change through time, which confounds the simple multiplicative factor.” Default: True
- **storm\_id** (*str or list of str, optional*) – IBTrACS ID of the storm, e.g. 1988234N13299, [1988234N13299, 1989260N11316].
- **year\_range** (*tuple (min\_year, max\_year), optional*) – Year range to filter track selection. Default: None.
- **basin** (*str, optional*) – If given, select storms that have at least one position in the specified basin. This allows analysis of a given basin, but also means that basin-specific track sets should not be combined across basins since some storms will be in more than one set. If you would like to select storms by their (unique) genesis basin instead, use the parameter *genesis\_basin*. For possible values (basin abbreviations), see the parameter *genesis\_basin*. If None, this filter is not applied. Default: None.
- **genesis\_basin** (*str, optional*) – The basin where a TC is formed is not defined in IBTrACS. However, this filter option allows to restrict to storms whose first valid eye position is in the specified basin, which simulates the genesis location. Note that the resulting genesis basin of a particular track may depend on the selected *provider* and on *estimate\_missing* because only the first *valid* eye position is considered. Possible values are ‘NA’ (North Atlantic), ‘SA’ (South Atlantic), ‘EP’ (Eastern North Pacific, which includes the Central Pacific region), ‘WP’ (Western North Pacific), ‘SP’ (South Pacific), ‘SI’ (South Indian), ‘NI’ (North Indian). If None, this filter is not applied. Default: None.
- **interpolate\_missing** (*bool, optional*) – If True, interpolate temporal reporting gaps within a variable (such as pressure, wind speed, or radius) linearly if possible. Temporal interpolation is with respect to the time steps defined in IBTrACS for a particular storm. No new time steps are added that are not originally defined in IBTrACS. For each time step with a missing value, this procedure is only able to fill in that value if there are other time steps before and after this time step for which values have been reported. This procedure will be applied before the statistical estimations referred to by *estimate\_missing*. It is applied to all variables (eye position, wind speed, environmental and central pressure, storm radius and radius of maximum winds). Default: True
- **estimate\_missing** (*bool, optional*) – For each fixed time step, estimate missing pressure, wind speed and radius using other variables that are available at that time step. The relationships between the variables are purely statistical. In comparison to *interpolate\_missing*, this procedure is able to estimate values for variables that haven’t been reported by any agency at any time step, as long as other variables are available. A typical example are storms before 1950, for which there are often no reported values for pressure, but for wind speed. In this case, a rough statistical pressure-wind relationship is applied to estimate the missing pressure values from the available wind-speed values. Make sure to set *rescale\_windspeeds=True* when using this option because the statistical relationships are calibrated using rescaled wind speeds. Default: False
- **correct\_pres** (*bool, optional*) – For backwards compatibility, alias for *estimate\_missing*. This is deprecated, use *estimate\_missing* instead!

- **discard\_single\_points** (*bool, optional*) – Whether to discard tracks that consists of a single point. Recommended for full compatibility with other functions such as *equal\_timesteps*. Default: True.
- **file\_name** (*str, optional*) – Name of NetCDF file to be downloaded or located at climada/data/system. Default: 'IBTrACS.ALL.v04r00.nc'

**read\_processed\_ibtracs\_csv**(*file\_names*)

Fill from processed ibtracs csv file(s).

**Parameters** **file\_names** (*str or list of str*) – Absolute file name(s) or folder name containing the files to read.

**read\_simulations\_emanuel**(*file\_names, hemisphere=None*)

Fill from Kerry Emanuel tracks.

**Parameters**

- **file\_names** (*str or list of str*) – Absolute file name(s) or folder name containing the files to read.
- **hemisphere** (*str or None, optional*) – For global data sets, restrict to northern ('N') or southern ('S') hemisphere. Default: None (no restriction)

**read\_one\_gettelman**(*nc\_data, i\_track*)

Fill from Andrew Gettelman tracks.

**Parameters**

- **nc\_data** (*str*) – netCDF4.Dataset Objekt
- **i\_tracks** (*int*) – track number

**read\_simulations\_chaz**(*file\_names, year\_range=None, ensemble\_nums=None*)

Read track output from CHAZ simulations

Lee, C.-Y., Tippett, M.K., Sobel, A.H., Camargo, S.J. (2018): An Environmentally Forced Tropical Cyclone Hazard Model. J Adv Model Earth Sy 10(1): 223–241.

**Parameters**

- **file\_names** (*str or list of str*) – Absolute file name(s) or folder name containing the files to read.
- **year\_range** (*tuple (min\_year, max\_year), optional*) – Filter by year, if given.
- **ensemble\_nums** (*list, optional*) – Filter by ensembleNum, if given.

**read\_simulations\_storm**(*path, years=None*)

Read track output from STORM simulations

Bloemendaal et al. (2020): Generation of a global synthetic tropical cyclone hazard dataset using STORM. Scientific Data 7(1): 40.

Track data available for download from

<https://doi.org/10.4121/uuid:82c1dc0d-5485-43d8-901a-ce7f26cda35d>

**Parameters**

- **path** (*str*) – Full path to a txt-file as contained in the *data.zip* archive from the official source linked above.

- **years** (*list of int, optional*) – If given, only read the specified “years” from the txt-File. Note that a “year” refers to one ensemble of tracks in the data set that represents one sample year.

**equal\_timestep**(*time\_step\_h=1, land\_params=False*)

Generate interpolated track values to time steps of *time\_step\_h*.

**Parameters**

- **time\_step\_h** (*float or int, optional*) – Temporal resolution in hours (positive, may be non-integer-valued). Default: 1.
- **land\_params** (*bool, optional*) – If True, recompute *on\_land* and *dist\_since\_lf* at each node. Default: False.

**calc\_random\_walk**(*\*\*kwargs*)

Deprecated. Use *TCTracks.calc\_perturbed\_trajectories* instead.

**calc\_perturbed\_trajectories**(*\*\*kwargs*)

See function in *climada.hazard.tc\_tracks\_synth*.

**property size**

Get longitude from coord array.

**get\_bounds**(*deg\_buffer=0.1*)

Get bounds as (lon\_min, lat\_min, lon\_max, lat\_max) tuple.

**Parameters** **deg\_buffer** (*float*) – A buffer to add around the bounding box

**Returns** **bounds**

**Return type** tuple (lon\_min, lat\_min, lon\_max, lat\_max)

**property bounds**

Exact bounds of trackset as tuple, no buffer.

**get\_extent**(*deg\_buffer=0.1*)

Get extent as (lon\_min, lon\_max, lat\_min, lat\_max) tuple.

**Parameters** **deg\_buffer** (*float*) – A buffer to add around the bounding box

**Returns** **extent**

**Return type** tuple (lon\_min, lon\_max, lat\_min, lat\_max)

**property extent**

Exact extent of trackset as tuple, no buffer.

**generate\_centroids**(*res\_deg, buffer\_deg*)

Generate gridded centroids within padded bounds of tracks

**Parameters**

- **res\_deg** (*float*) – Resolution in degrees.
- **buffer\_deg** (*float*) – Buffer around tracks in degrees.

**Returns** **centroids** – Centroids instance.

**Return type** *Centroids*

**plot**(*axis=None, figsize=(9, 13), legend=True, adapt\_fontsize=True, \*\*kwargs*)

Track over earth. Historical events are blue, probabilistic black.

**Parameters**

- **axis** (*matplotlib.axes.\_subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*((float, float), optional)*) – figure size for plt.subplots The default is (9, 13)
- **legend** (*bool, optional*) – whether to display a legend of Tropical Cyclone categories. Default: True.
- **kwargs** (*optional*) – arguments for LineCollection matplotlib, e.g. alpha=0.5
- **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.

**Returns** axis

**Return type** matplotlib.axes.\_subplots.AxesSubplot

**write\_netcdf**(*folder\_name*)

Write a netcdf file per track with track.sid name in given folder.

**Parameters** **folder\_name** (*str*) – Folder name where to write files.

**read\_netcdf**(*folder\_name*)

Read all netcdf files contained in folder and fill a track per file.

**Parameters** **folder\_name** (*str*) – Folder name where to write files.

**to\_geodataframe**(*as\_points=False, split\_lines\_antimeridian=True*)

Transform this TCTracks instance into a GeoDataFrame.

**Parameters**

- **as\_points** (*bool, optional*) – If False (default), one feature (row) per track with a LineString or MultiLineString as geometry (or Point geometry for tracks of length one) and all track attributes (sid, name, orig\_event\_flag, etc) as dataframe columns. If True, one feature (row) per track time step, with variable values per time step (radius\_max\_wind, max\_sustained\_wind, etc) as columns in addition to attributes.
- **split\_lines\_antimeridian** (*bool, optional*) – If True, tracks that cross the antimeridian are split into multiple Lines as a MultiLineString, with each Line on either side of the meridian. This ensures all Lines are within (-180, +180) degrees longitude. Note that lines might be split at more locations than strictly necessary, due to the underlying splitting algorithm (<https://github.com/Toblerity/Shapely/issues/572>).

**Returns** gdf

**Return type** GeoDataFrame

`climada.hazard.tc_tracks.set_category(max_sus_wind, wind_unit='kn', saffir_scale=None)`

Add storm category according to Saffir-Simpson hurricane scale.

**Parameters**

- **max\_sus\_wind** (*np.array*) – Maximum sustained wind speed records for a single track.
- **wind\_unit** (*str, optional*) – Units of wind speed. Default: 'kn'.
- **saffir\_scale** (*list, optional*) – Saffir-Simpson scale in same units as wind (default scale valid for knots).

**Returns**

**category** –

**Intensity of given track according to the Saffir-Simpson hurricane scale:**



- -1 : tropical depression
- 0 : tropical storm
- 1 : Hurricane category 1
- 2 : Hurricane category 2
- 3 : Hurricane category 3
- 4 : Hurricane category 4
- 5 : Hurricane category 5

**Return type** int

### climada.hazard.tc\_tracks\_synth module

```
climada.hazard.tc_tracks_synth.LANDFALL_DECAY_V = {-1: 0.00012859077693295416, 0:
0.0017226346292718126, 1: 0.002309772914350468, 2: 0.0025968221565522698, 3:
0.002626252944053856, 4: 0.002550639312763181, 5: 0.003788695795963695}
```

Global landfall decay parameters for wind speed by TC category. Keys are TC categories with -1='TD', 0='TS', 1='Cat 1', ..., 5='Cat 5'. It is v\_rel as derived from: tracks = TCTracks() tracks.read\_ibtracs\_netcdf(year\_range=(1980,2019),

estimate\_missing=True)

extent = tracks.get\_extent() land\_geom = climada.util.coordinates.get\_land\_geometry(

extent=extent, resolution=10

) v\_rel, p\_rel = \_calc\_land\_decay(tracks.data, land\_geom,

pool=tracks.pool)

```
climada.hazard.tc_tracks_synth.LANDFALL_DECAY_P = {-1: (1.0088807492745373,
0.002117478217863062), 0: (1.0192813768091684, 0.003068578025845065), 1:
(1.0362982218631644, 0.003620816186262243), 2: (1.0468630800617038,
0.004067381088015585), 3: (1.0639055205005432, 0.003708174876364079), 4:
(1.0828373148889825, 0.003997492773076179), 5: (1.1088615145002092,
0.005224331234796362)}
```

Global landfall decay parameters for pressure by TC category. Keys are TC categories with -1='TD', 0='TS', 1='Cat 1', ..., 5='Cat 5'. It is p\_rel as derived from: tracks = TCTracks() tracks.read\_ibtracs\_netcdf(year\_range=(1980,2019),

estimate\_missing=True)

extent = tracks.get\_extent() land\_geom = climada.util.coordinates.get\_land\_geometry(

extent=extent, resolution=10

) v\_rel, p\_rel = \_calc\_land\_decay(tracks.data, land\_geom,

pool=tracks.pool)

```
climada.hazard.tc_tracks_synth.calc_perturbed_trajectories(tracks, nb_synth_tracks=9,
 max_shift_ini=0.75,
 max_dspeed_rel=0.3,
 max_ddirection=0.008726646259971648,
 autocorr_dspeed=0.85,
 autocorr_ddirection=0.5, seed=54,
 decay=True,
 use_global_decay_params=True)
```

Generate synthetic tracks based on directed random walk. An ensemble of `nb_synth_tracks` synthetic tracks is computed for every track contained in self.

The methodology perturbs the tracks locations, and if `decay` is `True` it additionally includes decay of wind speed and central pressure drop after landfall. No other track parameter is perturbed. The track starting point location is perturbed by random uniform values of magnitude up to `max_shift_ini` in both longitude and latitude. Then, each segment between two consecutive points is perturbed in direction and distance (i.e., translational speed). These perturbations can be correlated in time, i.e., the perturbation in direction applied to segment `i` is correlated with the perturbation in direction applied to segment `i-1` (and similarly for the perturbation in translational speed). Perturbations in track direction and temporal auto-correlations in perturbations are on an hourly basis, and the perturbations in translational speed is relative. Hence, the parameter values are relatively insensitive to the temporal resolution of the tracks. Note however that all tracks should be at the same temporal resolution, which can be achieved using `equal_timestep()`. `max_dspeed_rel` and `autocorr_dspeed` control the spread along the track ('what distance does the track run for'), while `max_ddirection` and `autocorr_ddirection` control the spread perpendicular to the track movement ('how does the track diverge in direction'). `max_dspeed_rel` and `max_ddirection` control the amplitude of perturbations at each track timestep but perturbations may tend to compensate each other over time, leading to a similar location at the end of the track, while `autocorr_dspeed` and `autocorr_ddirection` control how these perturbations persist in time and hence the amplitude of the perturbations towards the end of the track.

Note that the default parameter values have been only roughly calibrated so that the frequency of tracks in each 5x5degree box remains approximately constant. This is not an in-depth calibration and should be treated as such. The object is mutated in-place.

#### Parameters

- **tracks** (*climada.hazard.TCTracks*) – Tracks data.
- **nb\_synth\_tracks** (*int, optional*) – Number of ensemble members per track. Default: 9.
- **max\_shift\_ini** (*float, optional*) – Amplitude of max random starting point shift in decimal degree (up to +/-`max_shift_ini` for longitude and latitude). Default: 0.75.
- **max\_dspeed\_rel** (*float, optional*) – Amplitude of translation speed perturbation in relative terms (e.g., 0.2 for +/-20%). Default: 0.3.
- **max\_ddirection** (*float, optional*) – Amplitude of track direction (bearing angle) perturbation per hour, in radians. Default:  $\pi/360$ .
- **autocorr\_dspeed** (*float, optional*) – Temporal autocorrelation in translation speed perturbation at a lag of 1 hour. Default: 0.85.
- **autocorr\_ddirection** (*float, optional*) – Temporal autocorrelation of translational direction perturbation at a lag of 1 hour. Default: 0.5.
- **seed** (*int, optional*) – Random number generator seed for replicability of random walk. Put negative value if you don't want to use it. Default: configuration file.
- **decay** (*bool, optional*) – Whether to apply landfall decay in probabilistic tracks. Default: `True`.

- **use\_global\_decay\_params** (*bool, optional*) – Whether to use precomputed global parameter values for landfall decay obtained from IBTrACS (1980-2019). If False, parameters are fitted using historical tracks in input parameter ‘tracks’, in which case the landfall decay applied depends on the tracks passed as an input and may not be robust if few historical tracks make landfall in this object. Default: True.

## climada.hazard.trop\_cyclone module

**class** climada.hazard.trop\_cyclone.TropCyclone(*pool=None*)

Bases: *climada.hazard.base.Hazard*

Contains 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.array(int)

### 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(str)

### intensity\_thres = 17.5

intensity threshold for storage in m/s

### vars\_opt = {'category'}

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

### \_\_init\_\_(*pool=None*)

Initialize values.

**Parameters** **pool** (*pathos.pool, optional*) – Pool that will be used for parallel computation when applicable. Default: None

### set\_from\_tracks(\*args, \*\*kwargs)

This function is deprecated, use TropCyclone.from\_tracks instead.

**static from\_tracks**(*tracks, centroids=None, pool=None, description='', model='H08', ignore\_distance\_to\_coast=False, store\_windfields=False, metric='equirect', intensity\_thres=17.5*)

Clear and fill with windfields from specified tracks.

This function sets the *TropCyclone.intensity* attribute to contain, for each centroid, the maximum wind speed (1-minute sustained winds at 10 meters above ground) experienced over the whole period of each TC event in m/s. The wind speed is set to 0 if it doesn’t exceed the threshold in *TropCyclone.intensity\_thres*.

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

The *TropCyclone.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, vectorial winds can be stored using the *store\_windfields* function parameter (see below).

### Parameters

- **tracks** (*TCTracks*) – Tracks of storm events.
- **centroids** (*Centroids, optional*) – Centroids where to model TC. Default: global centroids at 360 arc-seconds resolution.
- **pool** (*pathos.pool, optional*) – Pool that will be used for parallel computation of wind fields. Default: None
- **description** (*str, optional*) – Description of the event set. Default: “”.
- **model** (*str, optional*) – Parametric wind field model to use: one of “H1980” (the prominent Holland 1980 model), “H08” (Holland 1980 with b-value from Holland 2008), or “H10” (Holland et al. 2010). Default: “H08”.
- **ignore\_distance\_to\_coast** (*boolean, optional*) – If True, centroids far from coast are not ignored. Default: False.
- **store\_windfields** (*boolean, optional*) – If True, the Hazard object gets a list *windfields* of sparse matrices. For each track, the full velocity vectors at each centroid and track position are stored in a sparse matrix of shape (npositions, ncentroids \* 2) that can be reshaped to a full ndarray of shape (npositions, ncentroids, 2). 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*) – Wind speeds (in m/s) below this threshold are stored as 0. Default: 17.5

**Raises ValueError –**

**set\_climate\_scenario\_knu**(*ref\_year=2050, rcp\_scenario=45*)

Compute future events for a given RCP scenario and year based on the parametrized values derived from Table 3 in Knutson et al 2015. <https://doi.org/10.1175/JCLI-D-15-0129.1> . The scaling for different years and RCP scenarios is obtained by linear interpolation.β

Note: The parametrized values are derived from the overall changes in statistical ensemble of tracks. Hence, this method should only be applied to sufficiently large tropical cyclone event sets that approximate the reference years 1981 - 2008 used in Knutson et. al.

The frequency and intensity changes are applied independently from one another. The mean intensity factors can thus slightly deviate from the Knutson value (deviation was found to be less than 1% for default IBTrACS event sets 1980-2020 for each basin).

#### Parameters

- **ref\_year** (*int*) – year between 2000 ad 2100. Default: 2050
- **rcp\_scenario** (*int*) – 26 for RCP 2.6, 45 for RCP 4.5, 60 for RCP 6.0 and 85 for RCP 8.5. The default is 45.

**Returns haz\_cc** – Tropical cyclone with frequencies and intensity scaled according to the Knutson criterion for the given year and RCP. Returns a new instance of `climada.hazard.TropCyclone`, self is not modified.

**Return type** `climada.hazard.TropCyclone`

**static video\_intensity**(*track\_name*, *tracks*, *centroids*, *file\_name*=None,  
*writer*=<matplotlib.animation.PillowWriter object>, *figsize*=(9, 13),  
*adapt\_fontsize*=True, *\*\*kwargs*)

Generate video of TC wind fields node by node and returns its corresponding TropCyclone instances and track pieces.

#### Parameters

- **track\_name** (*str*) – name of the track contained in tracks to record
- **tracks** (*climada.hazard.TCTracks*) – tropical cyclone tracks
- **centroids** (*climada.hazard.Centroids*) – centroids where wind fields are mapped
- **file\_name** (*str*, *optional*) – file name to save video (including full path and file extension)
- **writer** (*matplotlib.animation.*, *optional*\*) – video writer. Default is pillow with bitrate=500
- **figsize** (*tuple*, *optional*) – figure size for plt.subplots
- **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** (*optional*) – arguments for pcolormesh matplotlib function used in event plots

**Returns** *tc\_list*, *tc\_coord*

**Return type** *list(TropCyclone)*, *list(np.array)*

**Raises** *ValueError* –

**frequency\_from\_tracks**(*tracks*)

Set hazard frequency from tracks data.

**Parameters** *tracks* (*list of xarray.Dataset*)

**static from\_single\_track**(*track*, *centroids*, *coastal\_idx*, *model*='H08', *store\_windfields*=False,  
*metric*='equirect', *intensity\_thres*=17.5)

Generate windfield hazard from a single track dataset

#### Parameters

- **track** (*xr.Dataset*) – Single tropical cyclone track.
- **centroids** (*Centroids*) – Centroids instance.
- **coastal\_idx** (*np.array*) – Indices of centroids close to coast.
- **model** (*str*, *optional*) – Parametric wind field model, one of “H1980” (the prominent Holland 1980 model), “H08” (Holland 1980 with b-value from Holland 2008), or “H10” (Holland et al. 2010). Default: “H08”.
- **store\_windfields** (*boolean*, *optional*) – If True, store windfields. Default: False.
- **metric** (*str*, *optional*) – Specify an approximation method to use for earth distances: “equirect” (faster) or “geosphere” (more accurate). See *dist\_approx* function in *climada.util.coordinates*. Default: “equirect”.
- **intensity\_thres** (*float*, *optional*) – Wind speeds (in m/s) below this threshold are stored as 0. Default: 17.5

**Raises** *ValueError*, *KeyError* –

Returns `haz`

Return type *TropCyclone*

## 7.1.4 climada.util package

### climada.util.api\_client module

```
class climada.util.api_client.Download(*args, **kwargs)
```

Bases: `peewee.Model`

Database entry keeping track of downloaded files from the CLIMADA data API

`url = <CharField: Download.url>`

`path = <CharField: Download.path>`

`startdownload = <DateTimeField: Download.startdownload>`

`enddownload = <DateTimeField: Download.enddownload>`

**exception Failed**

Bases: `Exception`

The download failed for some reason.

**DoesNotExist**

alias of `climada.util.api_client.DownloadDoesNotExist`

`id = <AutoField: Download.id>`

```
class climada.util.api_client.FileInfo(uuid: str, url: str, file_name: str, file_format: str, file_size: int,
 check_sum: str)
```

Bases: `object`

file data from CLIMADA data API.

`uuid: str`

`url: str`

`file_name: str`

`file_format: str`

`file_size: int`

`check_sum: str`

`__init__(uuid: str, url: str, file_name: str, file_format: str, file_size: int, check_sum: str) → None`

```
class climada.util.api_client.DataTypeInfo(data_type: str, data_type_group: str, description: str,
 properties: Optional[list] = None)
```

Bases: `object`

data\_type data from CLIMADA data API.

`data_type: str`

`data_type_group: str`

`description: str`

`properties: list = None`

`__init__(data_type: str, data_type_group: str, description: str, properties: Optional[list] = None) → None`

```
class climada.util.api_client.DatasetInfo(uuid: str, data_type: climada.util.api_client.DataTypeInfo,
 name: str, version: str, status: str, properties: dict, files: list,
 doi: str, description: str, license: str, activation_date: str,
 expiration_date: str)
```

Bases: object

dataset data from CLIMADA data API.

**uuid:** str

**data\_type:** [climada.util.api\\_client.DataTypeInfo](#)

**name:** str

**version:** str

**status:** str

**properties:** dict

**files:** list

**doi:** str

**description:** str

**license:** str

**activation\_date:** str

**expiration\_date:** str

**static from\_json**(json)

creates a DatasetInfo object from the json object returned by the CLIMADA data api server.

**Parameters** json (dict)

**Returns**

**Return type** [DatasetInfo](#)

```
__init__(uuid: str, data_type: climada.util.api_client.DataTypeInfo, name: str, version: str, status: str,
 properties: dict, files: list, doi: str, description: str, license: str, activation_date: str,
 expiration_date: str) → None
```

```
climada.util.api_client.checksize(local_path, fileinfo)
```

Checks sanity of downloaded file simply by comparing actual and registered size.

**Parameters**

- **local\_path** (*Path*) – the downloaded file
- **fileinfo** (*FileInfo*) – file information from CLIMADA data API

**Raises** [Download.Failed](#) – if the file is not what it's supposed to be

```
climada.util.api_client.checkhash(local_path, fileinfo)
```

Checks sanity of downloaded file by comparing actual and registered check sum.

**Parameters**

- **local\_path** (*Path*) – the downloaded file
- **fileinfo** (*FileInfo*) – file information from CLIMADA data API

**Raises** [Download.Failed](#) – if the file is not what it's supposed to be

**class** climada.util.api\_client.**Client**

Bases: object

Python wrapper around REST calls to the CLIMADA data API server.

**MAX\_WAITING\_PERIOD** = 6

**exception** **AmbiguousResult**

Bases: Exception

Custom Exception for Non-Unique Query Result

**exception** **NoResult**

Bases: Exception

Custom Exception for No Query Result

**\_\_init\_\_**()

Constructor of Client.

Data API host and chunk\_size (for download) are configurable values. Default values are 'climada.ethz.ch' and 8096 respectively.

**get\_datasets**(data\_type=None, name=None, version=None, properties=None, status='active')

Find all datasets matching the given parameters.

#### Parameters

- **data\_type** (*str, optional*) – data\_type of the dataset, e.g., 'litpop' or 'draught'
- **name** (*str, optional*) – the name of the dataset
- **version** (*str, optional*) – the version of the dataset
- **properties** (*dict, optional*) – search parameters for dataset properties, by default None any property has a string for key and can be a string or a list of strings for value
- **status** (*str, optional*) – valid values are 'preliminary', 'active', 'expired', and 'test\_dataset', by default 'active'

#### Returns

**Return type** list of DatasetInfo

**get\_dataset**(data\_type=None, name=None, version=None, properties=None, status=None)

Find the one dataset that matches the given parameters.

#### Parameters

- **data\_type** (*str, optional*) – data\_type of the dataset, e.g., 'litpop' or 'draught'
- **name** (*str, optional*) – the name of the dataset
- **version** (*str, optional*) – the version of the dataset
- **properties** (*dict, optional*) – search parameters for dataset properties, by default None any property has a string for key and can be a string or a list of strings for value
- **status** (*str, optional*) – valid values are 'preliminary', 'active', 'expired', and 'test\_dataset', by default None

#### Returns

**Return type** [DatasetInfo](#)

#### Raises



- **AmbiguousResult** – when there is more than one dataset matching the search parameters
- **NoResult** – when there is no dataset matching the search parameters

**get\_dataset\_by\_uuid(uuid)**

Returns the data from 'https://climada/rest/dataset/{uuid}' as DatasetInfo object.

**Parameters** **uuid** (*str*) – the universal unique identifier of the dataset

**Returns**

**Return type** *DatasetInfo*

**Raises** **NoResult** – if the uuid is not valid

**get\_data\_types(data\_type\_group=None)**

Returns all data types from the climada data API belonging to a given data type group.

**Parameters** **data\_type\_group** (*str, optional*) – name of the data type group, by default None

**Returns**

**Return type** list of DataTypeInfo

**get\_data\_type(data\_type)**

Returns the data type from the climada data API with a given name.

**Parameters** **data\_type** (*str*) – data type name

**Returns**

**Return type** *DataTypeInfo*

**Raises** **NoResult** – if there is no such data type registered

**download\_file(local\_path, fileinfo, check=<function checksize>, retries=3)**

Download a file if it is not already present at the target destination.

**Parameters**

- **local\_path** (*Path*) – target destination, if it is a directory the original filename (fileinfo.filename) is kept
- **fileinfo** (*FileInfo*) – file object as retrieved from the data api
- **check** (*function, optional*) – how to check download success, by default checksize
- **retries** (*int, optional*) – how many times one should retry in case of failure, by default 3

**Returns** the path to the downloaded file

**Return type** *Path*

**Raises** **Exception** – when number of retries was exceeded or when a download is already running

**download\_dataset(dataset, target\_dir=PosixPath('/home/docs/climada/data'), organize\_path=True, check=<function checksize>)**

Download all files from a given dataset to a given directory.

**Parameters**

- **dataset** (*DatasetInfo*) – the dataset
- **target\_dir** (*Path, optional*) – target directory for download, by default *climada.util.constants.SYSTEM\_DIR*

- **organize\_path** (*bool, optional*) – if set to True the files will end up in subdirectories of target\_dir: [target\_dir]/[data\_type\_group]/[data\_type]/[name]/[version] by default True
- **check** (*function, optional*) – how to check download success for each file, by default Download.checksize

**Returns**

- **download\_dir** (*Path*) – the path to the directory containing the downloaded files, will be created if organize\_path is True
- **downloaded\_files** (*list of Path*) – the downloaded files themselves

**Raises Exception** – when one of the files cannot be downloaded

**static purge\_cache**(*local\_path*)

Removes entry from the sqlite database that keeps track of files downloaded by *cached\_download*. This may be necessary in case a previous attempt has failed in an uncontrolled way (power outage or the like).

**Parameters**

- **local\_path** (*Path*) – target destination
- **fileinfo** (*FileInfo*) – file object as retrieved from the data api

**get\_hazard**(*hazard\_type, dump\_dir=PosixPath('/home/docs/clinada/data'), max\_datasets=1, \*\*kwargs*)

Queries the data api for hazard datasets of the given type, downloads associated hdf5 files and turns them into a climada.hazard.Hazard object.

**Parameters**

- **hazard\_type** (*str*) – Type of climada hazard.
- **dump\_dir** (*str, optional*) – Directory where the files should be downloaded. Default: SYSTEM\_DIR If the directory is the SYSTEM\_DIR, the eventual target directory is organized into dump\_dir > hazard\_type > dataset name > version
- **max\_datasets** (*int, optional*) – Download limit for datasets. If a query matches is matched by more datasets than this number, a ValueError is raised. Setting it to 0 or a negative number inactivates the limit. Default is 10.
- **\*\*kwargs** – additional parameters passed on to get\_datasets

**Returns** The combined hazard object

**Return type** climada.hazard.Hazard

**to\_hazard**(*datasets, dump\_dir=PosixPath('/home/docs/clinada/data')*)

Downloads hdf5 files belonging to the given datasets reads them into Hazards and concatenates them into a single climada.Hazard object.

**Parameters**

- **datasets** (*list of DatasetInfo*) – Datasets to download and read into climada.Hazard objects.
- **dump\_dir** (*str, optional*) – Directory where the files should be downloaded. Default: SYSTEM\_DIR If the directory is the SYSTEM\_DIR, the eventual target directory is organized into dump\_dir > hazard\_type > dataset name > version

**Returns** The combined hazard object

**Return type** climada.hazard.Hazard

**get\_exposures**(*exposures\_type*, *dump\_dir*=PosixPath('/home/docs/clinada/data'), *max\_datasets*=10, *\*\*kwargs*)

Queries the data api for exposures datasets of the given type, downloads associated hdf5 files and turns them into a climada.entity.exposures.Exposures object.

#### Parameters

- **hazard\_type** (*str*) – Type of climada exposures.
- **dump\_dir** (*str*, *optional*) – Directory where the files should be downoladed. Default: SYSTEM\_DIR If the directory is the SYSTEM\_DIR, the eventual target directory is organized into dump\_dir > hazard\_type > dataset name > version
- **max\_datasets** (*int*, *optional*) – Download limit for datasets. If a query matches is matched by more datasets than this number, a ValueError is raised. Setting it to 0 or a negative number inactivates the limit. Default is 10.
- **\*\*kwargs** – additional parameters passed on to *Client.get\_datasets*

**Returns** The combined exposures object

**Return type** climada.entity.exposures.Exposures

**to\_exposures**(*datasets*, *dump\_dir*=PosixPath('/home/docs/clinada/data'))

Downloads hdf5 files belonging to the given datasets reads them into Exposures and concatenates them into a single climada.Exposures object.

#### Parameters

- **datasets** (*list of DatasetInfo*) – Datasets to download and read into climada.Exposures objects.
- **dump\_dir** (*str*, *optional*) – Directory where the files should be downoladed. Default: SYSTEM\_DIR If the directory is the SYSTEM\_DIR, the eventual target directory is organized into dump\_dir > exposures\_type > dataset name > version

**Returns** The combined exposures object

**Return type** climada.entity.exposures.Exposures

**get\_litpop\_default**(*country*=None, *dump\_dir*=PosixPath('/home/docs/clinada/data'))

Get a LitPop instance on a 150arcsec grid with the default parameters: exponents = (1,1) and fin\_mode = 'pc'.

#### Parameters

- **country** (*str or list*, *optional*) – List of country name or iso3 codes for which to create the LitPop object. If None is given, a global LitPop instance is created. Default is None
- **dump\_dir** (*str*) – directory where the files should be downoladed. Default: SYSTEM\_DIR

**Returns** default litpop Exposures object

**Return type** climada.entity.exposures.Exposures

**static into\_datasets\_df**(*datasets*)

Convenience function providing a DataFrame of datasets with properties.

**Parameters** **datasets** (*list of DatasetInfo*) – e.g., return of get\_datasets

**Returns** of datasets with properties as found in query by arguments

**Return type** pandas.DataFrame

**static** `into_files_df(datasets)`

Convenience function providing a DataFrame of files aligned with the input datasets.

**Parameters** `datasets` (*list of DatasetInfo*) – e.g., return of `get_datasets`

**Returns** of the files' informations including dataset informations

**Return type** `pandas.DataFrame`

### `climada.util.checker` module

`climada.util.checker.size(exp_len, var, var_name)`

Check if the length of a variable is the expected one.

**Raises** `ValueError` –

`climada.util.checker.shape(exp_row, exp_col, var, var_name)`

Check if the length of a variable is the expected one.

**Raises** `ValueError` –

`climada.util.checker.array_optional(exp_len, var, var_name)`

Check if array has right size. Warn if array empty. Call `check_size`.

**Parameters**

- `exp_len` (*str*) – expected array size
- `var` (*np.array*) – numpy array to check
- `var_name` (*str*) – name of the variable. Used in error/warning msg

**Raises** `ValueError` –

`climada.util.checker.array_default(exp_len, var, var_name, def_val)`

Check array has right size. Set default value if empty. Call `check_size`.

**Parameters**

- `exp_len` (*str*) – expected array size
- `var` (*np.array*) – numpy array to check
- `var_name` (*str*) – name of the variable. Used in error/warning msg
- `def_val` (*np.array*) – numpy array used as default value

**Raises** `ValueError` –

**Returns**

**Return type** Filled array

### `climada.util.config` module

### `climada.util.constants` module

`climada.util.constants.SYSTEM_DIR = PosixPath('/home/docs/climada/data')`

Folder containing the data used internally

`climada.util.constants.DEMO_DIR = PosixPath('/home/docs/climada/demo/data')`

Folder containing the data used for tutorials

```

climada.util.constants.ENT_DEMO_TODAY =
PosixPath('/home/docs/climada/demo/data/demo_today.xlsx')
 Entity demo present in xlsx format.

climada.util.constants.ENT_DEMO_FUTURE =
PosixPath('/home/docs/climada/demo/data/demo_future_TEST.xlsx')
 Entity demo future in xlsx format.

climada.util.constants.HAZ_DEMO_MAT =
PosixPath('/home/docs/climada/demo/data/atl_prob_nonames.mat')
 hurricanes from 1851 to 2011 over Florida with 100 centroids.

 Type Hazard demo from climada in MATLAB

climada.util.constants.HAZ_DEMO_FL =
PosixPath('/home/docs/climada/demo/data/SC22000_VE_M1.grd.gz')
 Raster file of flood over Venezuela. Model from GAR2015

climada.util.constants.ENT_TEMPLATE_XLS =
PosixPath('/home/docs/climada/data/entity_template.xlsx')
 Entity template in xls format.

climada.util.constants.HAZ_TEMPLATE_XLS =
PosixPath('/home/docs/climada/data/hazard_template.xlsx')
 Hazard template in xls format.

climada.util.constants.ONE_LAT_KM = 111.12
 Mean one latitude (in degrees) to km

climada.util.constants.EARTH_RADIUS_KM = 6371
 Earth radius in km

climada.util.constants.GLB_CENTROIDS_MAT =
PosixPath('/home/docs/climada/data/GLB_NatID_grid_0360as_adv_2.mat')
 Global centroids

climada.util.constants.GLB_CENTROIDS_NC =
PosixPath('/home/docs/climada/data/NatID_grid_0150as.nc')
 For backwards compatibility, it remains available under its old name.

climada.util.constants.ISIMIP_GPWV3_NATID_150AS =
PosixPath('/home/docs/climada/data/NatID_grid_0150as.nc')
 Compressed version of National Identifier Grid in 150 arc-seconds from ISIMIP project, based on GPWv3.
 Location in ISIMIP repository:

 ISIMIP2a/InputData/landuse_humaninfluences/population/ID_GRID/Nat_id_grid_ISIMIP.nc

 More references:

 • https://www.isimip.org/gettingstarted/input-data-bias-correction/details/13/
 • https://sedac.ciesin.columbia.edu/data/set/gpw-v3-national-identifier-grid

climada.util.constants.NATEARTH_CENTROIDS = {150:
PosixPath('/home/docs/climada/data/NatEarth_Centroids_150as.hdf5'), 360:
PosixPath('/home/docs/climada/data/NatEarth_Centroids_360as.hdf5')}
 Global centroids at XXX arc-seconds resolution, including region ids from Natural Earth. The 360 AS file
 includes distance to coast from NASA.

climada.util.constants.RIVER_FLOOD_REGIONS_CSV =
PosixPath('/home/docs/climada/data/NatRegIDs.csv')
 Look-up table for river flood module

```

```
climada.util.constants.TC_ANDREW_FL =
PosixPath('/home/docs/climada/demo/data/ibtracs_global_intp-None_1992230N11325.csv')
Tropical cyclone Andrew in Florida
```

```
climada.util.constants.HAZ_DEMO_H5 =
PosixPath('/home/docs/climada/demo/data/tc_fl_1990_2004.h5')
IBTrACS from 1990 to 2004 over Florida with 2500 centroids.
```

**Type** Hazard demo in hdf5 format

```
climada.util.constants.EXP_DEMO_H5 =
PosixPath('/home/docs/climada/demo/data/exp_demo_today.h5')
Exposures over Florida
```

```
climada.util.constants.WS_DEMO_NC =
[PosixPath('/home/docs/climada/demo/data/fp_lothar_crop-test.nc'),
PosixPath('/home/docs/climada/demo/data/fp_xynthia_crop-test.nc')]
Winter storm in Europe files. These test files have been generated using the netCDF kitchen sink:
```

```
>>> ncks -d latitude,50.5,54.0 -d longitude,3.0,7.5 ./file_in.nc ./file_out.nc
```

```
climada.util.constants.TEST_UNC_OUTPUT_IMPACT = 'test_unc_output_impact'
Demo uncertainty impact output
```

```
climada.util.constants.TEST_UNC_OUTPUT_COSTBEN = 'test_unc_output_costben'
Demo uncertainty costben output
```

### climada.util.coordinates module

```
climada.util.coordinates.NE_EPSG = 4326
Natural Earth CRS EPSG
```

```
climada.util.coordinates.NE_CRS = {'init': 'epsg:4326', 'no_defs': True}
Natural Earth CRS
```

```
climada.util.coordinates.TMP_ELEVATION_FILE =
PosixPath('/home/docs/climada/data/tmp_elevation.tif')
Path of elevation file written in set_elevation
```

```
climada.util.coordinates.DEM_NODATA = -9999
Value to use for no data values in DEM, i.e see points
```

```
climada.util.coordinates.MAX_DEM_TILES_DOWN = 300
Maximum DEM tiles to download
```

```
climada.util.coordinates.latlon_to_geosph_vector(lat, lon, rad=False, basis=False)
Convert lat/lon coordinates to radial vectors (on geosphere)
```

#### Parameters

- **lat, lon** (*ndarrays of floats, same shape*) – Latitudes and longitudes of points.
- **rad** (*bool, optional*) – If True, latitude and longitude are not given in degrees but in radians.
- **basis** (*bool, optional*) – If True, also return an orthonormal basis of the tangent space at the given points in lat-lon coordinate system. Default: False.

#### Returns

- **vn** (*ndarray of floats, shape (... , 3)*) – Same shape as lat/lon input with additional axis for components.

- **vbasis** (*ndarray of floats, shape (... , 2, 3)*) – Only present, if *basis* is True. Same shape as lat/lon input with additional axes for components of the two basis vectors.

`climada.util.coordinates.lon_normalize(lon, center=0.0)`

Normalizes degrees such that always  $-180 < \text{lon} - \text{center} \leq 180$

The input data is modified in place!

#### Parameters

- **lon** (*np.array*) – Longitudinal coordinates
- **center** (*float, optional*) – Central longitude value to use instead of 0. If None, the central longitude is determined automatically.

**Returns** **lon** – Normalized longitudinal coordinates. Since the input *lon* is modified in place (!), the returned array is the same Python object (instead of a copy).

**Return type** `np.array`

`climada.util.coordinates.lon_bounds(lon, buffer=0.0)`

Bounds of a set of degree values, respecting the periodicity in longitude

The longitudinal upper bound may be 180 or larger to make sure that the upper bound is always larger than the lower bound. The lower longitudinal bound will never lie below -180 and it will only assume the value -180 if the specified buffering enforces it.

Note that, as a consequence of this, the returned bounds do not satisfy the inequality  $\text{lon\_min} \leq \text{lon} \leq \text{lon\_max}$  in general!

Usually, an application of this function is followed by a renormalization of longitudinal values around the longitudinal middle value:

```
>>> bounds = lon_bounds(lon)
>>> lon_mid = 0.5 * (bounds[0] + bounds[2])
>>> lon = lon_normalize(lon, center=lon_mid)
>>> np.all((bounds[0] <= lon) & (lon <= bounds[2]))
```

#### Example

```
>>> lon_bounds(np.array([-179, 175, 178]))
(175, 181)
>>> lon_bounds(np.array([-179, 175, 178]), buffer=1)
(174, 182)
```

#### Parameters

- **lon** (*np.array*) – Longitudinal coordinates
- **buffer** (*float, optional*) – Buffer to add to both sides of the bounding box. Default: 0.0.

**Returns** **bounds** – Bounding box of the given points.

**Return type** `tuple (lon_min, lon_max)`

`climada.util.coordinates.latlon_bounds(lat, lon, buffer=0.0)`

Bounds of a set of degree values, respecting the periodicity in longitude

See *lon\_bounds* for more information about the handling of longitudinal values crossing the antimeridian.

### Example

```
>>> latlon_bounds(np.array([0, -2, 5]), np.array([-179, 175, 178]))
(175, -2, 181, 5)
>>> latlon_bounds(np.array([0, -2, 5]), np.array([-179, 175, 178]), buffer=1)
(174, -3, 182, 6)
```

#### Parameters

- **lat** (*np.array*) – Latitudinal coordinates
- **lon** (*np.array*) – Longitudinal coordinates
- **buffer** (*float, optional*) – Buffer to add to all sides of the bounding box. Default: 0.0.

**Returns** **bounds** – Bounding box of the given points.

**Return type** tuple (lon\_min, lat\_min, lon\_max, lat\_max)

`climada.util.coordinates.dist_approx(lat1, lon1, lat2, lon2, log=False, normalize=True, method='equirect', units='km')`

Compute approximation of geodistance in specified units

#### Parameters

- **lat1, lon1** (*ndarrays of floats, shape (nbatch, nx)*) – Latitudes and longitudes of first points.
- **lat2, lon2** (*ndarrays of floats, shape (nbatch, ny)*) – Latitudes and longitudes of second points.
- **log** (*bool, optional*) – If True, return the tangential vectors at the first points pointing to the second points (Riemannian logarithm). Default: False.
- **normalize** (*bool, optional*) – If False, assume that lon values are already between -180 and 180. Default: True
- **method** (*str, optional*) – Specify an approximation method to use: \* “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.

Note that ellipsoidal distances would be even more accurate, but are currently not implemented. Default: “equirect”.

- **units** (*str, optional*) – Specify a unit for the distance. One of: \* “km”: distance in km. \* “degree”: angular distance in decimal degrees. \* “radian”: angular distance in radians. Default: “km”.

#### Returns

- **dist**s (*ndarray of floats, shape (nbatch, nx, ny)*) – Approximate distances in specified units.
- **vtan** (*ndarray of floats, shape (nbatch, nx, ny, 2)*) – If *log* is True, tangential vectors at first points in local lat-lon coordinate system.

`climada.util.coordinates.get_gridcellarea(lat, resolution=0.5, unit='km2')`



The area covered by a grid cell is calculated depending on the latitude  $1 \text{ degree} = \text{ONE\_LAT\_KM}$  (111.12km at the equator) longitudinal distance in km =  $\text{ONE\_LAT\_KM} * \text{resolution} * \cos(\text{lat})$  latitudal distance in km =  $\text{ONE\_LAT\_KM} * \text{resolution}$  area = longitudinal distance \* latitudal distance

#### Parameters

- **lat** (*np.array*) – Latitude of the respective grid cell
- **resolution** (*int, optional*) – raster resolution in degree (default: 0.5 degree)
- **unit** (*string, optional*) – unit of the output area (default: km2, alternative: m2)

`climada.util.coordinates.grid_is_regular(coord)`

Return True if grid is regular. If True, returns height and width.

**Parameters** `coord` (*np.array*) – Each row is a lat-lon-pair.

#### Returns

- **regular** (*bool*) – Whether the grid is regular. Only in this case, the following width and height are reliable.
- **height** (*int*) – Height of the supposed grid.
- **width** (*int*) – Width of the supposed grid.

`climada.util.coordinates.get_coastlines(bounds=None, resolution=110)`

Get Polygons of coast intersecting given bounds

#### Parameters

- **bounds** (*tuple*) – min\_lon, min\_lat, max\_lon, max\_lat in EPSG:4326
- **resolution** (*float, optional*) – 10, 50 or 110. Resolution in m. Default: 110m, i.e. 1:110.000.000

**Returns** `coastlines` – Polygons of coast intersecting given bounds.

**Return type** `GeoDataFrame`

`climada.util.coordinates.convert_wgs_to_utm(lon, lat)`

Get EPSG code of UTM projection for input point in EPSG 4326

#### Parameters

- **lon** (*float*) – longitude point in EPSG 4326
- **lat** (*float*) – latitude of point (lat, lon) in EPSG 4326

**Returns** `epsg_code` – EPSG code of UTM projection.

**Return type** `int`

`climada.util.coordinates.utm_zones(wgs_bounds)`

Get EPSG code and bounds of UTM zones covering specified region

**Parameters** `wgs_bounds` (*tuple*) – lon\_min, lat\_min, lon\_max, lat\_max

**Returns** `zones` – EPSG code and bounding box in WGS coordinates.

**Return type** `list of pairs (zone_epsg, zone_wgs_bounds)`

`climada.util.coordinates.dist_to_coast(coord_lat, lon=None, signed=False)`

Compute (signed) distance to coast from input points in meters.

#### Parameters

- **coord\_lat** (*GeoDataFrame or np.array or float*) – One of the following: \* GeoDataFrame with geometry column in epsg:4326 \* np.array with two columns, first for latitude of each point and second with longitude in epsg:4326
  - np.array with one dimension containing latitudes in epsg:4326
  - float with a latitude value in epsg:4326
- **lon** (*np.array or float, optional*) – One of the following: \* np.array with one dimension containing longitudes in epsg:4326 \* float with a longitude value in epsg:4326
- **signed** (*bool*) – If True, distance is signed with positive values off shore and negative values on land. Default: False

**Returns** **dist** – (Signed) distance to coast in meters.

**Return type** np.array

`climada.util.coordinates.dist_to_coast_nasa(lat, lon, highres=False, signed=False)`

Read interpolated (signed) distance to coast (in m) from NASA data

Note: The NASA raster file is 300 MB and will be downloaded on first run!

#### Parameters

- **lat** (*np.array*) – latitudes in epsg:4326
- **lon** (*np.array*) – longitudes in epsg:4326
- **highres** (*bool, optional*) – Use full resolution of NASA data (much slower). Default: False.
- **signed** (*bool*) – If True, distance is signed with positive values off shore and negative values on land. Default: False

**Returns** **dist** – (Signed) distance to coast in meters.

**Return type** np.array

`climada.util.coordinates.get_land_geometry(country_names=None, extent=None, resolution=10)`

Get union of the specified (or all) countries or the points inside the extent.

#### Parameters

- **country\_names** (*list, optional*) – list with ISO3 names of countries, e.g ['ZWE', 'GBR', 'VNM', 'UZB']
- **extent** (*tuple, optional*) – (min\_lon, max\_lon, min\_lat, max\_lat)
- **resolution** (*float, optional*) – 10, 50 or 110. Resolution in m. Default: 10m, i.e. 1:10.000.000

**Returns** **geom** – Polygonal shape of union.

**Return type** shapely.geometry.multipolygon.MultiPolygon

`climada.util.coordinates.coord_on_land(lat, lon, land_geom=None)`

Check if points are on land.

#### Parameters

- **lat** (*np.array*) – latitude of points in epsg:4326
- **lon** (*np.array*) – longitude of points in epsg:4326

- **land\_geom** (*shapely.geometry.multipolygon.MultiPolygon, optional*) – If given, use these as profiles of land. Otherwise, the global landmass is used.

**Returns** **on\_land** – Entries are True if corresponding coordinate is on land and False otherwise.

**Return type** np.array(bool)

`climada.util.coordinates.nat_earth_resolution(resolution)`

Check if resolution is available in Natural Earth. Build string.

**Parameters** **resolution** (*int*) – resolution in millions, 110 == 1:110.000.000.

**Returns** **res\_name** – Natural Earth name of resolution (e.g. '110m')

**Return type** str

**Raises** **ValueError** –

`climada.util.coordinates.get_country_geometries(country_names=None, extent=None, resolution=10)`

Natural Earth country boundaries within given extent

If no arguments are given, simply returns the whole natural earth dataset.

Take heed: we assume WGS84 as the CRS unless the Natural Earth download utility from cartopy starts including the projection information. (They are saving a whopping 147 bytes by omitting it.) Same goes for UTF.

**Parameters**

- **country\_names** (*list, optional*) – list with ISO 3166 alpha-3 codes of countries, e.g ['ZWE', 'GBR', 'VNM', 'UZB']
- **extent** (*tuple (min\_lon, max\_lon, min\_lat, max\_lat), optional*) – Extent, assumed to be in the same CRS as the natural earth data.
- **resolution** (*float, optional*) – 10, 50 or 110. Resolution in m. Default: 10m

**Returns** **geom** – Natural Earth multipolygons of the specified countries, resp. the countries that lie within the specified extent.

**Return type** GeoDataFrame

`climada.util.coordinates.get_region_gridpoints(countries=None, regions=None, resolution=150, iso=True, rect=False, basemap='natearth')`

Get coordinates of gridpoints in specified countries or regions

**Parameters**

- **countries** (*list, optional*) – ISO 3166-1 alpha-3 codes of countries, or internal numeric NatID if *iso* is set to False.
- **regions** (*list, optional*) – Region IDs.
- **resolution** (*float, optional*) – Resolution in arc-seconds, either 150 (default) or 360.
- **iso** (*bool, optional*) – If True, assume that countries are given by their ISO 3166-1 alpha-3 codes (instead of the internal NatID). Default: True.
- **rect** (*bool, optional*) – If True, a rectangular box around the specified countries/regions is selected. Default: False.
- **basemap** (*str, optional*) – Choose between different data sources. Currently available: "isimip" and "natearth". Default: "natearth".

**Returns**

- **lat** (*np.array*) – Latitude of points in epsg:4326.

- **lon** (*np.array*) – Longitude of points in epsg:4326.

`climada.util.coordinates.assign_grid_points(x, y, grid_width, grid_height, grid_transform)`

To each coordinate in *x* and *y*, assign the closest centroid in the given raster grid

Make sure that your grid specification is relative to the same coordinate reference system as the *x* and *y* coordinates. In case of lon/lat coordinates, make sure that the longitudinal values are within the same longitudinal range (such as [-180, 180]).

If your grid is given by bounds instead of a transform, the functions `rasterio.transform.from_bounds` and `pts_to_raster_meta` might be helpful.

#### Parameters

- **x, y** (*np.array*) – *x*- and *y*-coordinates of points to assign coordinates to.
- **grid\_width** (*int*) – Width (number of columns) of the grid.
- **grid\_height** (*int*) – Height (number of rows) of the grid.
- **grid\_transform** (*affine.Affine*) – Affine transformation defining the grid raster.

**Returns** **assigned\_idx** – Index into the flattened *grid*. Note that the value *-1* is used to indicate that no matching coordinate has been found, even though *-1* is a valid index in NumPy!

**Return type** *np.array* of size equal to the size of *x* and *y*

`climada.util.coordinates.assign_coordinates(coords, coords_to_assign, method='NN', distance='haversine', threshold=100)`

To each coordinate in *coords*, assign a matching coordinate in *coords\_to\_assign*

If there is no exact match for some entry, an attempt is made to assign the geographically nearest neighbor. If the distance to the nearest neighbor exceeds *threshold*, the index *-1* is assigned.

Currently, the nearest neighbor matching works with lat/lon coordinates only. However, you can disable nearest neighbor matching by setting *threshold* to 0, in which case only exactly matching coordinates are assigned to each other.

Make sure that all coordinates are according to the same coordinate reference system. In case of lat/lon coordinates, the “haversine” distance is able to correctly compute the distance across the antimeridian. However, when exact matches are enforced with *threshold=0*, lat/lon coordinates need to be given in the same longitudinal range (such as (-180, 180)).

#### Parameters

- **coords** (*np.array with two columns*) – Each row is a geographical coordinate pair. The result's size will match this array's number of rows.
- **coords\_to\_assign** (*np.array with two columns*) – Each row is a geographical coordinate pair. The result will be an index into the rows of this array. Make sure that these coordinates use the same coordinate reference system as *coords*.
- **method** (*str, optional*) – Interpolation method to use for non-exact matching. Currently, “NN” (nearest neighbor) is the only supported value, see `climada.util.interpolation.interpol_index`.
- **distance** (*str, optional*) – Distance to use for non-exact matching. Possible values are “haversine” and “approx”, see `climada.util.interpolation.interpol_index`. Default: “haversine”
- **threshold** (*float, optional*) – If the distance to the nearest neighbor exceeds *threshold*, the index *-1* is assigned. Set *threshold* to 0 to disable nearest neighbor matching. Default: 100 (km)

**Returns** `assigned_idx` – Index into `coords_to_assign`. Note that the value `-1` is used to indicate that no matching coordinate has been found, even though `-1` is a valid index in NumPy!

**Return type** `np.array` of size equal to the number of rows in `coords`

`climada.util.coordinates.region2isos(regions)`

Convert region names to ISO 3166 alpha-3 codes of countries

**Parameters** `regions` (*str or list of str*) – Region name(s).

**Returns** `isos` – Sorted list of iso codes of all countries in specified region(s).

**Return type** `list of str`

`climada.util.coordinates.country_to_iso(countries, representation='alpha3', fillvalue=None)`

Determine ISO 3166 representation of countries

### Example

```
>>> country_to_iso(840)
'USA'
>>> country_to_iso("United States", representation="alpha2")
'US'
>>> country_to_iso(["United States of America", "SU"], "numeric")
[840, 810]
```

Some geopolitical areas that are not covered by ISO 3166 are added in the “user-assigned” range of ISO 3166-compliant values:

```
>>> country_to_iso(["XK", "Dhekelia"], "numeric") # XK for Kosovo
[983, 907]
```

### Parameters

- **countries** (*one of str, int, list of str, list of int*) – Country identifiers: name, official name, alpha-2, alpha-3 or numeric ISO codes. Numeric representations may be specified as str or int.
- **representation** (*str (one of “alpha3”, “alpha2”, “numeric”, “name”), optional*) – All countries are converted to this representation according to ISO 3166. Default: “alpha3”.
- **fillvalue** (*str or int or None, optional*) – The value to assign if a country is not recognized by the given identifier. By default, a `LookupError` is raised. Default: `None`

**Returns** `iso_list` – ISO 3166 representation of countries. Will only return a list if the input is a list. Numeric representations are returned as integers.

**Return type** `one of str, int, list of str, list of int`

`climada.util.coordinates.country_iso_alpha2numeric(iso_alpha)`

Deprecated: Use `country_to_iso` with `representation="numeric"` instead

`climada.util.coordinates.country_natid2iso(natids, representation='alpha3')`

Convert internal NatIDs to ISO 3166-1 alpha-3 codes

### Parameters

- **natids** (*int or list of int*) – NatIDs of countries (or single ID) as used in ISIMIP’s version of the GPWv3 national identifier grid.

- **representation** (*str*, one of “alpha3”, “alpha2” or “numeric”) – All countries are converted to this representation according to ISO 3166. Default: “alpha3”.

**Returns** **iso\_list** – ISO 3166 representation of countries. Will only return a list if the input is a list. Numeric representations are returned as integers.

**Return type** one of str, int, list of str, list of int

`climada.util.coordinates.country_iso2natid(isos)`

Convert ISO 3166-1 alpha-3 codes to internal NatIDs

**Parameters** **isos** (*str or list of str*) – ISO codes of countries (or single code).

**Returns** **natids** – Will only return a list if the input is a list.

**Return type** int or list of int

`climada.util.coordinates.natearth_country_to_int(country)`

Integer representation (ISO 3166, if possible) of Natural Earth GeoPandas country row

**Parameters** **country** (*GeoSeries*) – Row from Natural Earth GeoDataFrame.

**Returns** **iso\_numeric** – Integer representation of given country.

**Return type** int

`climada.util.coordinates.get_country_code(lat, lon, gridded=False)`

Provide numeric (ISO 3166) code for every point.

Oceans get the value zero. Areas that are not in ISO 3166 are given values in the range above 900 according to NATEARTH\_AREA\_NONISO\_NUMERIC.

**Parameters**

- **lat** (*np.array*) – latitude of points in epsg:4326
- **lon** (*np.array*) – longitude of points in epsg:4326
- **gridded** (*bool*) – If True, interpolate precomputed gridded data which is usually much faster. Default: False.

**Returns** **country\_codes** – Numeric code for each point.

**Return type** np.array(int)

`climada.util.coordinates.get_admin1_info(country_names)`

Provide Natural Earth registry info and shape files for admin1 regions

**Parameters** **country\_names** (*list*) – list with ISO3 names of countries, e.g. ['ZWE', 'GBR', 'VNM', 'UZB']

**Returns**

- **admin1\_info** (*dict*) – Data according to records in Natural Earth database.
- **admin1\_shapes** (*dict*) – Shape according to Natural Earth.

`climada.util.coordinates.get_resolution_1d(coords, min_resol=1e-08)`

Compute resolution of scalar grid

**Parameters**

- **coords** (*np.array*) – scalar coordinates
- **min\_resol** (*float, optional*) – minimum resolution to consider. Default: 1.0e-8.

**Returns** **res** – Resolution of given grid.

**Return type** float

`climada.util.coordinates.get_resolution(*coords, min_resol=1e-08)`

Compute resolution of n-d grid points

**Parameters**

- **X, Y, ...** (*np.array*) – Scalar coordinates in each axis
- **min\_resol** (*float, optional*) – minimum resolution to consider. Default: 1.0e-8.

**Returns** **resolution** – Resolution in each coordinate direction.

**Return type** pair of floats

`climada.util.coordinates.pts_to_raster_meta(points_bounds, res)`

Transform vector data coordinates to raster.

If a raster of the given resolution doesn't exactly fit the given bounds, the raster might have slightly larger (but never smaller) bounds.

**Parameters**

- **points\_bounds** (*tuple*) – points total bounds (xmin, ymin, xmax, ymax)
- **res** (*tuple*) – resolution of output raster (xres, yres)

**Returns**

- **nrows** (*int*) – Number of rows.
- **ncols** (*int*) – Number of columns.
- **ras\_trans** (*affine.Affine*) – Affine transformation defining the raster.

`climada.util.coordinates.raster_to_meshgrid(transform, width, height)`

Get coordinates of grid points in raster

**Parameters**

- **transform** (*affine.Affine*) – Affine transform defining the raster.
- **width** (*int*) – Number of points in first coordinate axis.
- **height** (*int*) – Number of points in second coordinate axis.

**Returns**

- **x** (*np.array*) – x-coordinates of grid points.
- **y** (*np.array*) – y-coordinates of grid points.

`climada.util.coordinates.to_crs_user_input(crs_obj)`

Returns a crs string or dictionary from a hdf5 file object.

bytes are decoded to str if the string starts with a '{' it is assumed to be a dumped string from a dictionary and ast is used to parse it.

**Parameters** **crs\_obj** (*int, dict or str or bytes*) – the crs object to be converted user input

**Returns** to eventually be used as argument of `rasterio.crs.CRS.from_user_input` and `pyproj.crs.CRS.from_user_input`

**Return type** str or dict

**Raises** **ValueError** – if `type(crs_obj)` has the wrong type

`climada.util.coordinates.equal_crs(crs_one, crs_two)`

Compare two crs

**Parameters**

- **crs\_one** (*dict, str or int*) – user crs
- **crs\_two** (*dict, str or int*) – user crs

**Returns** **equal** – Whether the two specified CRS are equal according to rasterio.crs.CRS.from\_user\_input

**Return type** bool

`climada.util.coordinates.read_raster(file_name, band=None, src_crs=None, window=None, geometry=None, dst_crs=None, transform=None, width=None, height=None, resampling=Resampling.nearest)`

Read raster of bands and set 0-values to the masked ones.

**Parameters**

- **file\_name** (*str*) – name of the file
- **band** (*list(int), optional*) – band number to read. Default: 1
- **window** (*rasterio.windows.Window, optional*) – window to read
- **geometry** (*shapely.geometry, optional*) – consider pixels only in shape
- **dst\_crs** (*crs, optional*) – reproject to given crs
- **transform** (*rasterio.Affine*) – affine transformation to apply
- **width** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp.Resampling optional*) – resampling function used for re-projection to dst\_crs

**Returns**

- **meta** (*dict*) – Raster meta (height, width, transform, crs).
- **data** (*np.array*) – Each row corresponds to one band (raster points are flattened, can be reshaped to height x width).

`climada.util.coordinates.read_raster_bounds(path, bounds, res=None, bands=None)`

Read raster file within given bounds and refine to given resolution

Makes sure that the extent of pixel centers covers the specified regions

**Parameters**

- **path** (*str*) – Path to raster file to open with rasterio.
- **bounds** (*tuple*) – (xmin, ymin, xmax, ymax)
- **res** (*float, optional*) – Resolution of output. Default: Resolution of input raster file.
- **bands** (*list of int, optional*) – Bands to read from the input raster file. Default: [1]

**Returns**

- **data** (*3d np.array*) – First dimension is for the selected raster bands. Second dimension is y (lat) and third dimension is x (lon).
- **transform** (*rasterio.Affine*) – Affine transformation defining the output raster data.



`climada.util.coordinates.read_raster_sample(path, lat, lon, intermediate_res=None, method='linear', fill_value=None)`

Read point samples from raster file.

#### Parameters

- **path** (*str*) – path of the raster file
- **lat** (*np.array*) – latitudes in file's CRS
- **lon** (*np.array*) – longitudes in file's CRS
- **intermediate\_res** (*float, optional*) – If given, the raster is not read in its original resolution but in the given one. This can increase performance for files of very high resolution.
- **method** (*str, optional*) – The interpolation method, passed to `scipy.interp.interpn`. Default: 'linear'.
- **fill\_value** (*numeric, optional*) – The value used outside of the raster bounds. Default: The raster's nodata value or 0.

**Returns values** – Interpolated raster values for each given coordinate point.

**Return type** `np.array` of same length as `lat`

`climada.util.coordinates.interp_raster_data(data, interp_y, interp_x, transform, method='linear', fill_value=0)`

Interpolate raster data, given as array and affine transform

#### Parameters

- **data** (*np.array*) – 2d numpy array containing the values
- **interp\_y** (*np.array*) – y-coordinates of points (corresp. to first axis of data)
- **interp\_x** (*np.array*) – x-coordinates of points (corresp. to second axis of data)
- **transform** (*affine.Affine*) – affine transform defining the raster
- **method** (*str, optional*) –  
The interpolation method, passed to `scipy.interp.interpn`. Default: 'linear'.
- **fill\_value** (*numeric, optional*) –  
The value used outside of the raster bounds. Default: 0.

**Returns values** – Interpolated raster values for each given coordinate point.

**Return type** `np.array`

`climada.util.coordinates.refine_raster_data(data, transform, res, method='linear', fill_value=0)`

Refine raster data, given as array and affine transform

#### Parameters

- **data** (*np.array*) – 2d array containing the values
- **transform** (*affine.Affine*) – affine transform defining the raster
- **res** (*float or pair of floats*) – new resolution
- **method** (*str, optional*) –  
The interpolation method, passed to `scipy.interp.interpn`. Default: 'linear'.

#### Returns

- **new\_data** (*np.array*) – 2d array containing the interpolated values.

- **new\_transform** (*affine.Affine*) – Affine transform defining the refined raster.

`climada.util.coordinates.read_vector(file_name, field_name, dst_crs=None)`

Read vector file format supported by fiona.

#### Parameters

- **file\_name** (*str*) – vector file with format supported by fiona and ‘geometry’ field.
- **field\_name** (*list(str)*) – list of names of the columns with values.
- **dst\_crs** (*crs, optional*) – reproject to given crs

#### Returns

- **lat** (*np.array*) – Latitudinal coordinates.
- **lon** (*np.array*) – Longitudinal coordinates.
- **geometry** (*GeoSeries*) – Shape geometries.
- **value** (*np.array*) – Values associated to each shape.

`climada.util.coordinates.write_raster(file_name, data_matrix, meta, dtype=<class 'numpy.float32'>)`

Write raster in GeoTiff format.

#### Parameters

- **file\_name** (*str*) – File name to write.
- **data\_matrix** (*np.array*) – 2d raster data. Either containing one band, or every row is a band and the column represents the grid in 1d.
- **meta** (*dict*) – rasterio meta dictionary containing raster properties: width, height, crs and transform must be present at least. Include *compress="deflate"* for compressed output.
- **dtype** (*numpy dtype, optional*) – A numpy dtype. Default: `np.float32`

`climada.util.coordinates.points_to_raster(points_df, val_names=None, res=0.0, raster_res=0.0, crs='EPSG:4326', scheduler=None)`

Compute raster (as data and transform) from GeoDataFrame.

#### Parameters

- **points\_df** (*GeoDataFrame*) – contains columns latitude, longitude and those listed in the parameter *val\_names*.
- **val\_names** (*list of str, optional*) – The names of columns in *points\_df* containing values. The raster will contain one band per column. Default: `['value']`
- **res** (*float, optional*) – resolution of current data in units of latitude and longitude, approximated if not provided.
- **raster\_res** (*float, optional*) – desired resolution of the raster
- **crs** (*object (anything accepted by pyproj.CRS.from\_user\_input), optional*) – If given, overwrites the CRS information given in *points\_df*. If no CRS is explicitly given and there is no CRS information in *points\_df*, the CRS is assumed to be EPSG:4326 (lat/lon). Default: `None`
- **scheduler** (*str*) – used for `dask map_partitions`. “threads”, “synchronous” or “processes”

#### Returns

- **data** (*np.array*) – 2d array containing the raster values.
- **transform** (*affine.Affine*) – Affine transform defining the raster coordinates.

`climada.util.coordinates.subraster_from_bounds(transform, bounds)`

Compute a subraster definition from a given reference transform and bounds.

#### Parameters

- **transform** (*rasterio.Affine*) – Affine transformation defining the reference grid.
- **bounds** (*tuple of floats (xmin, ymin, xmax, ymax)*) – Bounds of the subraster in units and CRS of the reference grid.

#### Returns

- **dst\_transform** (*rasterio.Affine*) – Subraster affine transformation.
- **dst\_shape** (*tuple of ints (height, width)*) – Number of pixels of subraster in vertical and horizontal direction.

`climada.util.coordinates.align_raster_data(source, src_crs, src_transform, dst_crs=None, dst_resolution=None, dst_bounds=None, global_origin=(-180, 90), resampling=None, conserve=None, **kwargs)`

Reproject 2D np.ndarray to be aligned to a reference grid.

This function ensures that reprojected data with the same `dst_resolution` and `global_origins` are aligned to the same global grid, i.e., no offset between destination grid points for different source grids that are projected to the same target resolution.

Note that the origin is required to be in the upper left corner. The result is always oriented left to right (west to east) and top to bottom (north to south).

#### Parameters

- **source** (*np.ndarray*) – The source is a 2D ndarray containing the values to be reprojected.
- **src\_crs** (*CRS or dict*) – Source coordinate reference system, in rasterio dict format.
- **src\_transform** (*rasterio.Affine*) – Source affine transformation.
- **dst\_crs** (*CRS, optional*) – Target coordinate reference system, in rasterio dict format. Default: `src_crs`
- **dst\_resolution** (*tuple (x\_resolution, y\_resolution) or float, optional*) – Target resolution (positive pixel sizes) in units of the target CRS. Default: `(abs(src_transform[0]), abs(src_transform[4]))`
- **dst\_bounds** (*tuple of floats (xmin, ymin, xmax, ymax), optional*) – Bounds of the target raster in units of the target CRS. By default, the source's bounds are reprojected to the target CRS.
- **global\_origin** (*tuple (west, north) of floats, optional*) – Coordinates of the reference grid's upper left corner. Default: `(-180, 90)`. Make sure to change `global_origin` for non-geographical CRS!
- **resampling** (*int, rasterio.enums.Resampling or str, optional*) – Resampling method to use. String values like “nearest” or “bilinear” are resolved to attributes of `rasterio.enums.Resampling`. Default: `rasterio.enums.Resampling.nearest`
- **conserve** (*str, optional*) – If provided, conserve the source array's ‘mean’ or ‘sum’ in the transformed data or normalize the values of the transformed data ndarray (‘norm’). Default: `None` (no conservation)
- **kwargs** (*dict, optional*) – Additional arguments passed to `rasterio.warp.reproject`.

**Raises ValueError** –

#### Returns

- **destination** (np.ndarray with same dtype as *source*) – The transformed 2D ndarray.
- **dst\_transform** (*rasterio.Affine*) – Destination affine transformation.

`climada.util.coordinates.mask_raster_with_geometry(raster, transform, shapes, nodata=None, **kwargs)`

Change values in *raster* that are outside of given *shapes* to *nodata*.

This function is a wrapper for `rasterio.mask.mask` to allow for in-memory processing. This is done by first writing data to memfile and then reading from it before the function call to `rasterio.mask.mask()`. The `MemoryFile` will be discarded after exiting the with statement.

#### Parameters

- **raster** (*numpy.ndarray*) – raster to be masked with dim: [H, W].
- **transform** (*affine.Affine*) – the transform of the raster.
- **shapes** (*GeoJSON-like dict or an object that implements the Python geo*) – interface protocol (such as a `Shapely Polygon`) Passed to `rasterio.mask.mask`
- **nodata** (*int or float, optional*) – Passed to `rasterio.mask.mask`: Data points outside *shapes* are set to *nodata*.
- **kwargs** (*optional*) – Passed to `rasterio.mask.mask`.

**Returns** **masked** – raster with dim: [H, W] and points outside shapes set to *nodata*

**Return type** `numpy.ndarray` or `numpy.ma.MaskedArray`

`climada.util.coordinates.set_df_geometry_points(df_val, scheduler=None, crs=None)`

Set given geometry to given dataframe using dask if scheduler.

#### Parameters

- **df\_val** (*GeoDataFrame*) – contains latitude and longitude columns
- **scheduler** (*str, optional*) – used for dask map\_partitions. “threads”, “synchronous” or “processes”
- **crs** (*object (anything readable by pyproj4.CRS.from\_user\_input), optional*) – Coordinate Reference System, if omitted or `None`: `df_val.geometry.crs`

`climada.util.coordinates.fao_code_def()`

Generates list of FAO country codes and corresponding ISO numeric-3 codes.

#### Returns

- **iso\_list** (*list*) – list of ISO numeric-3 codes
- **faocode\_list** (*list*) – list of FAO country codes

`climada.util.coordinates.country_faocode2iso(input_fao)`

Convert FAO country code to ISO numeric-3 codes.

**Parameters** **input\_fao** (*int or array*) – FAO country codes of countries (or single code)

**Returns** **output\_iso** – ISO numeric-3 codes of countries (or single code)

**Return type** `int` or `array`

`climada.util.coordinates.country_iso2faocode(input_iso)`

Convert ISO numeric-3 codes to FAO country code.

**Parameters** **input\_iso** (*int or array*) – ISO numeric-3 codes of countries (or single code)

**Returns** **output\_faocode** – FAO country codes of countries (or single code)

**Return type** int or array

### climada.util.dates\_times module

`climada.util.dates_times.date_to_str(date)`

Compute date string in ISO format from input datetime ordinal int. :Parameters: **date** (*int or list or np.array*) – input datetime ordinal

**Returns**

**Return type** str or list(str)

`climada.util.dates_times.str_to_date(date)`

Compute datetime ordinal int from input date string in ISO format. :Parameters: **date** (*str or list*) – idate string in ISO format, e.g. '2018-04-06'

**Returns**

**Return type** int

`climada.util.dates_times.datetime64_to_ordinal(datetime)`

Converts from a numpy datetime64 object to an ordinal date. See <https://stackoverflow.com/a/21916253> for the horrible details. :Parameters: **datetime** (*np.datetime64, or list or np.array*) – date and time

**Returns**

**Return type** int

`climada.util.dates_times.last_year(ordinal_vector)`

Extract first year from ordinal date

**Parameters** **ordinal\_vector** (*list or np.array*) – input datetime ordinal

**Returns**

**Return type** int

`climada.util.dates_times.first_year(ordinal_vector)`

Extract first year from ordinal date

**Parameters** **ordinal\_vector** (*list or np.array*) – input datetime ordinal

**Returns**

**Return type** int

### climada.util.dwd\_icon\_loader module

`climada.util.dwd_icon_loader.download_icon_grib(run_datetime, model_name='icon-eu-eps',  
parameter_name='vmax_10m',  
max_lead_time=None, download_dir=None)`

download the gribfiles of a weather forecast run for a certain weather parameter from open-data.dwd.de/weather/nwp/.

**Parameters**

- **run\_datetime** (*datetime*) – The starting timepoint of the forecast run
- **model\_name** (*str*) – the name of the forecast model written as it appears in the folder structure in opendata.dwd.de/weather/nwp/ or 'test'
- **parameter\_name** (*str*) – the name of the meteorological parameter written as it appears in the folder structure in opendata.dwd.de/weather/nwp/

- **max\_lead\_time** (*int*) – number of hours for which files should be downloaded, will default to maximum available data
- **download\_dir** (: *str or Path*) – directory where the downloaded files should be saved in

**Returns** **file\_names** – a list of filenames that link to all just downloaded or available files from the forecast run, defined by the input parameters

**Return type** list

```
climada.util.dwd_icon_loader.delete_icon_grib(run_datetime, model_name='icon-eu-eps',
 parameter_name='vmax_10m', max_lead_time=None,
 download_dir=None)
```

delete the downloaded gribfiles of a weather forecast run for a certain weather parameter from open-data.dwd.de/weather/nwp/.

#### Parameters

- **run\_datetime** (*datetime*) – The starting timepoint of the forecast run
- **model\_name** (*str*) – the name of the forecast model written as it appears in the folder structure in opendata.dwd.de/weather/nwp/
- **parameter\_name** (*str*) – the name of the meteorological parameter written as it appears in the folder structure in opendata.dwd.de/weather/nwp/
- **max\_lead\_time** (*int*) – number of hours for which files should be deleted, will default to maximum available data
- **download\_dir** (*str or Path*) – directory where the downloaded files are stored at the moment

```
climada.util.dwd_icon_loader.download_icon_centroids_file(model_name='icon-eu-eps',
 download_dir=None)
```

create centroids based on netcdf files provided by dwd, links found here: [https://www.dwd.de/DE/leistungen/opendata/neuigkeiten/opendata\\_dez2018\\_02.html](https://www.dwd.de/DE/leistungen/opendata/neuigkeiten/opendata_dez2018_02.html) [https://www.dwd.de/DE/leistungen/opendata/neuigkeiten/opendata\\_aug2020\\_01.html](https://www.dwd.de/DE/leistungen/opendata/neuigkeiten/opendata_aug2020_01.html)

#### Parameters

- **model\_name** (*str*) – the name of the forecast model written as it appears in the folder structure in opendata.dwd.de/weather/nwp/
- **download\_dir** (*str or Path*) – directory where the downloaded files should be saved in

**Returns** **file\_name** – absolute path and filename of the downloaded and decompressed netcdf file

**Return type** str

### climada.util.earth\_engine module

### climada.util.files\_handler module

```
climada.util.files_handler.to_list(num_exp, values, val_name)
```

Check size and transform to list if necessary. If size is one, build a list with num\_exp repeated values.

#### Parameters

- **num\_exp** (*int*) – expected number of list elements
- **values** (*object or list(object)*) – values to check and transform
- **val\_name** (*str*) – name of the variable values

**Returns****Return type** list`climada.util.files_handler.get_file_names(file_name)`

Return list of files contained. Supports globbing.

**Parameters** `file_name` (*str or list(str)*) – Either a single string or a list of strings that are either - a file path - or the path of the folder containing the files - or a globbing pattern.**Returns****Return type** list(str)**climada.util.finance module**`climada.util.finance.net_present_value(years, disc_rates, val_years)`

Compute net present value.

**Parameters**

- `years` (*np.array*) – array with the sequence of years to consider.
- `disc_rates` (*np.array*) – discount rate for every year in years.
- `val_years` (*np.array*) – chash flow at each year.

**Returns****Return type** float`climada.util.finance.income_group(cntry_iso, ref_year, shp_file=None)`

Get country's income group from World Bank's data at a given year, or closest year value. If no data, get the natural earth's approximation.

**Parameters**

- `cntry_iso` (*str*) – key = ISO alpha\_3 country
- `ref_year` (*int*) – reference year
- `shp_file` (*cartopy.io.shapereader.Reader, optional*) – shape file with INCOME\_GRP attribute for every country. Load Natural Earth admin0 if not provided.

`climada.util.finance.gdp(cntry_iso, ref_year, shp_file=None, per_capita=False)`

Get country's (current value) GDP from World Bank's data at a given year, or closest year value. If no data, get the natural earth's approximation.

**Parameters**

- `cntry_iso` (*str*) – key = ISO alpha\_3 country
- `ref_year` (*int*) – reference year
- `shp_file` (*cartopy.io.shapereader.Reader, optional*) – shape file with INCOME\_GRP attribute for every country. Load Natural Earth admin0 if not provided.
- `per_capita` (*boolean, optional*) – If True, GDP is returned per capita

**Returns****Return type** float

## climada.util.hdf5\_handler module

`climada.util.hdf5_handler.read(file_name, with_refs=False)`

Load a hdf5 data structure from a file.

### Parameters

- **file\_name** – file to load
- **with\_refs** – enable loading of the references. Default is unset, since it increments the execution time considerably.

**Returns** dictionary structure containing all the variables.

**Return type** contents

## Examples

```
>>> # Contents contains the Matlab data in a dictionary.
>>> contents = read("/path/to/dummy.mat")
>>> # Contents contains the Matlab data and its reference in a dictionary.
>>> contents = read("/path/to/dummy.mat", True)
```

**Raises Exception while reading –**

`climada.util.hdf5_handler.get_string(array)`

Form string from input array of unsigned integers.

**Parameters** array – array of integers

**Returns**

**Return type** string

`climada.util.hdf5_handler.get_str_from_ref(file_name, var)`

Form string from a reference HDF5 variable of the given file.

**Parameters**

- **file\_name** – matlab file name
- **var** – HDF5 reference variable

**Returns**

**Return type** string

`climada.util.hdf5_handler.get_list_str_from_ref(file_name, var)`

Form list of strings from a reference HDF5 variable of the given file.

**Parameters**

- **file\_name** – matlab file name
- **var** – array of HDF5 reference variable

**Returns**

**Return type** string

`climada.util.hdf5_handler.get_sparse_csr_mat(mat_dict, shape)`

Form sparse matrix from input hdf5 sparse matrix data type.



**Parameters**

- **mat\_dict** – dictionary containing the sparse matrix information.
- **shape** – tuple describing output matrix shape.

**Returns**

**Return type** sparse csr matrix

**climada.util.interpolation module**

`climada.util.interpolation.interpol_index(centroids, coordinates, method='NN', distance='haversine', threshold=100)`

Returns for each coordinate the centroids indexes used for interpolation.

**Parameters**

- **centroids** (*2d array*) – First column contains latitude, second column contains longitude. Each row is a geographic point
- **coordinates** (*2d array*) – First column contains latitude, second column contains longitude. Each row is a geographic point
- **method** (*str, optional*) – interpolation method to use. NN default.
- **distance** (*str, optional*) – distance to use. Haversine default
- **threshold** (*float*) – distance threshold in km over which no neighbor will be found. Those are assigned with a -1 index

**Returns**

**Return type** numpy array with so many rows as coordinates containing the centroids indexes

`climada.util.interpolation.dist_sqr_approx(lats1, lons1, cos_lats1, lats2, lons2)`

Compute squared equirectangular approximation distance. Values need to be sqrt and multiplied by ONE\_LAT\_KM to obtain distance in km.

`climada.util.interpolation.DIST_DEF = ['approx', 'haversine']`

Distances

`climada.util.interpolation.METHOD = ['NN']`

Interpolation methods

**climada.util.plot module**

`climada.util.plot.geo_bin_from_array(array_sub, geo_coord, var_name, title, pop_name=True, buffer=1.0, extend='neither', proj=<cartopy.crs.PlateCarree object>, shapes=True, axes=None, figsize=(9, 13), adapt_fontsize=True, **kwargs)`

Plot array values binned over input coordinates.

**Parameters**

- **array\_sub** (*np.array(1d or 2d) or list(np.array)*) – Each array (in a row or in the list) are values at each point in corresponding geo\_coord that are binned in one subplot.
- **geo\_coord** (*2d np.array or list(2d np.array)*) – (lat, lon) for each point in a row. If one provided, the same grid is used for all subplots. Otherwise provide as many as subplots in array\_sub.

- **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.
- **pop\_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates, by default BUFFER
- **extend** (*str, optional*) – extend border colorbar with arrows. [ ‘neither’ | ‘both’ | ‘min’ | ‘max’ ], by default ‘neither’
- **proj** (*ccrs, optional*) – coordinate reference system of the given data, by default ccrs.PlateCarree()
- **shapes** (*bool, optional*) – Overlay Earth’s countries coastlines to matplotlib.pyplot axis. The default is True
- **axes** (*Axes or ndarray(Axes), optional*) – by default None
- **figsize** (*tuple, optional*) – figure size for plt.subplots, by default (9, 13)
- **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

#### Returns

**Return type** cartopy.mpl.geoaxes.GeoAxesSubplot

**Raises ValueError:** – Input array size mismatch

```
climada.util.plot.geo_im_from_array(array_sub, coord, var_name, title, proj=None, smooth=True,
 axes=None, figsize=(9, 13), adapt_fontsize=True, **kwargs)
```

Image(s) plot defined in array(s) over input coordinates.

#### Parameters

- **array\_sub** (*np.array(1d or 2d) or list(np.array)*) – Each array (in a row or in the list) are values at each point in corresponding geo\_coord that are plotted in one subplot.
- **coord** (*2d np.array*) – (lat, lon) for each point in a row. The same grid is used for all subplots.
- **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.
- **proj** (*ccrs, optional*) – coordinate reference system used in coordinates, by default None
- **smooth** (*bool, optional*) – smooth plot to RESOLUTIONxRESOLUTION, by default True
- **axes** (*Axes or ndarray(Axes), optional*) – by default None
- **figsize** (*tuple, optional*) – figure size for plt.subplots, by default (9, 13)
- **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 pcolormesh matplotlib function

#### Returns

**Return type** cartopy.mpl.geoaxes.GeoAxesSubplot

**Raises** `ValueError` –

`climada.util.plot.make_map(num_sub=1, figsize=(9, 13), proj=<cartopy.crs.PlateCarree object>, adapt_fontsize=True)`

Create map figure with cartopy.

#### Parameters

- **num\_sub** (*int or tuple*) – number of total subplots in figure OR number of subfigures in row and column: (num\_row, num\_col).
- **figsize** (*tuple*) – figure size for plt.subplots
- **proj** (*cartopy.crs projection, optional*) – geographical projection, The default is PlateCarree default.
- **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.

**Returns** `fig, axis_sub`

**Return type** matplotlib.figure.Figure, cartopy.mpl.geoaxes.GeoAxesSubplot

`climada.util.plot.add_shapes(axis)`

Overlay Earth's countries coastlines to matplotlib.pyplot axis.

#### Parameters

- **axis** (*cartopy.mpl.geoaxes.GeoAxesSubplot*) – Cartopy axis
- **projection** (*cartopy.crs projection, optional*) – Geographical projection. The default is PlateCarree.

`climada.util.plot.add_populated_places(axis, extent, proj=<cartopy.crs.PlateCarree object>, fontsize=None)`

Add city names.

#### Parameters

- **axis** (*cartopy.mpl.geoaxes.GeoAxesSubplot*) – cartopy axis.
- **extent** (*list*) – geographical limits [min\_lon, max\_lon, min\_lat, max\_lat]
- **proj** (*cartopy.crs projection, optional*) – geographical projection, The default is PlateCarree.
- **fontsize** (*int, optional*) – Size of the fonts. If set to None, the default matplotlib settings are used.

`climada.util.plot.add_cntry_names(axis, extent, proj=<cartopy.crs.PlateCarree object>, fontsize=None)`

Add country names.

#### Parameters

- **axis** (*cartopy.mpl.geoaxes.GeoAxesSubplot*) – Cartopy axis.
- **extent** (*list*) – geographical limits [min\_lon, max\_lon, min\_lat, max\_lat]
- **proj** (*cartopy.crs projection, optional*) – **Geographical projection.** The default is PlateCarree.
- **fontsize** [*int, optional*] Size of the fonts. If set to None, the default matplotlib settings are used.

### climada.util.save module

`climada.util.save.save(out_file_name, var)`

Save variable with provided file name. Uses configuration `save_dir` folder if no absolute path provided.

#### Parameters

- **out\_file\_name** (*str*) – file name (absolute path or relative to configured `save_dir`)
- **var** (*object*) – variable to save in pickle format

`climada.util.save.load(in_file_name)`

Load variable contained in file. Uses configuration `save_dir` folder if no absolute path provided.

**Parameters** `in_file_name` (*str*) – file name

#### Returns

**Return type** object

### climada.util.scalebar\_plot module

`climada.util.scalebar_plot.scale_bar(ax, location, length, metres_per_unit=1000, unit_name='km',  
tol=0.01, angle=0, color='black', linewidth=3, text_offset=0.005,  
ha='center', va='bottom', plot_kwargs=None, text_kwargs=None,  
**kwargs)`

Add a scale bar to CartoPy axes.

For angles between 0 and 90 the text and line may be plotted at slightly different angles for unknown reasons. To work around this, override the ‘rotation’ keyword argument with `text_kwargs`.

#### Parameters

- **ax** – CartoPy axes.
- **location** – Position of left-side of bar in axes coordinates.
- **length** – Geodesic length of the scale bar.
- **metres\_per\_unit** – Number of metres in the given unit. Default: 1000
- **unit\_name** – Name of the given unit. Default: ‘km’
- **tol** – Allowed relative error in length of bar. Default: 0.01
- **angle** – Anti-clockwise rotation of the bar.
- **color** – Color of the bar and text. Default: ‘black’
- **linewidth** – Same argument as for plot.
- **text\_offset** – Perpendicular offset for text in axes coordinates. Default: 0.005
- **ha** – Horizontal alignment. Default: ‘center’
- **va** – Vertical alignment. Default: ‘bottom’
- **plot\_kwargs** – Keyword arguments for plot, overridden by **\*\*kwargs**.
- **text\_kwargs** – Keyword arguments for text, overridden by **\*\*kwargs**.
- **kwargs** – Keyword arguments for both plot and text.

**climada.util.select module**

`climada.util.select.get_attributes_with_matching_dimension(obj, dims)`

Get the attributes of an object that have len(dims) number of dimensions or more, and all dims are individual parts of the attribute's shape.

**Parameters**

- **obj** (*object of any class*) – The object from which matching attributes are returned
- **dims** (*list[int]*) – List of dimensions size to match

**Returns** `list_of_attrs` – List of names of the attributes with matching dimensions

**Return type** `list[str]`

**climada.util.value\_representation module**

`climada.util.value_representation.sig_dig(x, n_sig_dig=16)`

Rounds x to n\_sig\_dig number of significant digits. 0, inf, Nan are returned unchanged. Examples: n\_sig\_dig = 5

1.234567 -> 1.2346, 123456.89 -> 123460.0

**Parameters**

- **x** (*float*) – number to be rounded
- **n\_sig\_dig** (*int, optional*) – Number of significant digits. The default is 16.

**Returns** Rounded number

**Return type** `float`

`climada.util.value_representation.sig_dig_list(iterable, n_sig_dig=16)`

Vectorized form of sig\_dig. Rounds a list of float to a number of significant digits

**Parameters**

- **iterable** (*iter(float)*) – iterable of numbers to be rounded
- **n\_sig\_dig** (*int, optional*) – Number of significant digits. The default is 16.

**Returns** list of rounded floats

**Return type** `list`

`climada.util.value_representation.value_to_monetary_unit(values, n_sig_dig=None, abbreviations=None)`

Converts list of values to closest common monetary unit 0, Nan and inf have not unit.

**Parameters**

- **values** (*int or float, list(int or float) or np.ndarray(int or float)*) – Values to be converted
- **n\_sig\_dig** (*int, optional*) – Number of significant digits to return. Examples n\_sig\_di=5: 1.234567 -> 1.2346, 123456.89 -> 123460.0 Default: all digits are returned.
- **abbreviations** (*dict, optional*) – Name of the abbreviations for the money 1000s counts Default:

{ 1: '1:', 1000: 'K', 1000000: 'M', 1000000000: 'Bn', 1000000000000: 'Tn' }

**Returns**

- **mon\_val** (*np.ndarray*) – Array of values in monetary unit
- **name** (*string*) – Monetary unit

### Examples

```
values = [1e6, 2*1e6, 4.5*1e7, 0, Nan, inf] -> [1, 2, 4.5, 0, Nan, inf] ['M']
```

### climada.util.yearsets module

```
climada.util.yearsets.impact_yearset(imp, sampled_years, lam=None, correction_fac=True)
```

Create a yearset of impacts (**yimp**) containing a probabilistic impact for each year in the `sampled_years` list by sampling events from the impact received as input with a Poisson distribution centered around `lam` per year (`lam = sum(imp.frequency)`). In contrast to the expected annual impact (`eai`) **yimp** contains impact values that differ among years. When correction factor is true, the **yimp** are scaled such that the average over all years is equal to the `eai`.

#### Parameters

- **imp** (*climada.engine.Impact()*) – impact object containing impacts per event
- **sampled\_years** (*list*) – A list of years that shall be covered by the resulting **yimp**.

#### Optional parameters

**lam**: **int** The applied Poisson distribution is centered around `lam` events per year. If no lambda value is given, the default `lam = sum(imp.frequency)` is used.

**correction\_fac** [boolean] If True a correction factor is applied to the resulting **yimp**. It is scaled in such a way that the expected annual impact (`eai`) of the **yimp** equals the `eai` of the input impact

#### Returns

- **yimp** (*climada.engine.Impact()*) – yearset of impacts containing annual impacts for all `sampled_years`
- **sampling\_vect** (*2D array*) – The sampling vector specifies how to sample the **yimp**, it consists of one sub-array per `sampled_year`, which contains the `event_ids` of the events used to calculate the annual impacts. Can be used to re-create the exact same **yimp**.

```
climada.util.yearsets.impact_yearset_from_sampling_vect(imp, sampled_years, sampling_vect,
 correction_fac=True)
```

Create a yearset of impacts (**yimp**) containing a probabilistic impact for each year in the `sampled_years` list by sampling events from the impact received as input following the sampling vector provided. In contrast to the expected annual impact (`eai`) **yimp** contains impact values that differ among years. When correction factor is true, the **yimp** are scaled such that the average over all years is equal to the `eai`.

#### Parameters

- **imp** (*climada.engine.Impact()*) – impact object containing impacts per event
- **sampled\_years** (*list*) – A list of years that shall be covered by the resulting **yimp**.

- **sampling\_vect** (*2D array*) – The sampling vector specifies how to sample the yimp, it consists of one sub-array per sampled\_year, which contains the event\_ids of the events used to calculate the annual impacts. It needs to be obtained in a first call, i.e. [yimp, sampling\_vect] = climada\_yearsets.impact\_yearset(...) and can then be provided in this function to obtain the exact same sampling (also for a different imp object)

#### Optional parameter

**correction\_fac** [boolean] If True a correction factor is applied to the resulting yimp. It is scaled in such a way that the expected annual impact (eai) of the yimp equals the eai of the input impact

**Returns yimp** – yearset of impacts containing annual impacts for all sampled\_years

**Return type** climada.engine.Impact()

climada.util.yearsets.**sample\_from\_poisson**(*n\_sampled\_years, lam*)

Sample the number of events for n\_sampled\_years

#### Parameters

- **n\_sampled\_years** (*int*) – The target number of years the impact yearset shall contain.
- **lam** (*int*) – the applied Poisson distribution is centered around lambda events per year

**Returns events\_per\_year** – Number of events per sampled year

**Return type** array

climada.util.yearsets.**sample\_events**(*events\_per\_year, freqs\_orig*)

Sample events uniformly from an array (indices\_orig) without replacement (if sum(events\_per\_year) > n\_input\_events the input events are repeated

(tot\_n\_events/n\_input\_events) times, by ensuring that the same events doesn't occur more than once per sampled year).

#### Parameters

- **events\_per\_year** (*array*) – Number of events per sampled year
- **freqs\_orig** (*array*) – Frequency of each input event

**Returns sampling\_vect** – The sampling vector specifies how to sample the yimp, it consists of one sub-array per sampled\_year, which contains the event\_ids of the events used to calculate the annual impacts.

**Return type** 2D array

climada.util.yearsets.**compute\_imp\_per\_year**(*imp, sampling\_vect*)

Sample annual impacts from the given event\_impacts according to the sampling dictionary

#### Parameters

- **imp** (*climada.engine.Impact()*) – impact object containing impacts per event
- **sampling\_vect** (*2D array*) – The sampling vector specifies how to sample the yimp, it consists of one sub-array per sampled\_year, which contains the event\_ids of the events used to calculate the annual impacts.

**Returns imp\_per\_year** – Sampled impact per year (length = sampled\_years)

**Return type** array

`climada.util.yearsets.calculate_correction_fac(imp_per_year, imp)`

Calculate a correction factor that can be used to scale the yimp in such a way that the expected annual impact (eai) of the yimp amounts to the eai of the input *imp*

**Parameters**

- ***imp\_per\_year*** (*array*) – sampled yimp
- ***imp*** (*climada.engine.Impact()*) – impact object containing impacts per event

**Returns** **correction\_factor** – The correction factor is calculated as  $\text{imp\_eai}/\text{yimp\_eai}$

**Return type** `int`

- `genindex`
- `modindex`



**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 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/>>.



## PYTHON MODULE INDEX

### C

`climada.engine.calibration_opt`, 289  
`climada.engine.cost_benefit`, 292  
`climada.engine.forecast`, 296  
`climada.engine.impact`, 301  
`climada.engine.impact_data`, 309  
`climada.engine.uncertainty_quantification.calc_base`, 271  
`climada.engine.uncertainty_quantification.calc_cost_benefit`, 273  
`climada.engine.uncertainty_quantification.calc_impact`, 275  
`climada.engine.uncertainty_quantification.input_var`, 276  
`climada.engine.uncertainty_quantification.unc_output`, 282  
`climada.entity.disc_rates.base`, 313  
`climada.entity.entity_def`, 346  
`climada.entity.exposures.base`, 327  
`climada.entity.exposures.litpop.gpw_population`, 316  
`climada.entity.exposures.litpop.litpop`, 317  
`climada.entity.exposures.litpop.nightlight`, 323  
`climada.entity.impact_funcs.base`, 336  
`climada.entity.impact_funcs.impact_func_set`, 337  
`climada.entity.impact_funcs.storm_europe`, 340  
`climada.entity.impact_funcs.trop_cyclone`, 340  
`climada.entity.measures.base`, 341  
`climada.entity.measures.measure_set`, 343  
`climada.entity.tag`, 347  
`climada.hazard.base`, 354  
`climada.hazard.centroids.cent`, 347  
`climada.hazard.isimip_data`, 362  
`climada.hazard.storm_europe`, 362  
`climada.hazard.tag`, 365  
`climada.hazard.tc_clim_change`, 366  
`climada.hazard.tc_tracks`, 366  
`climada.hazard.tc_tracks_synth`, 373  
`climada.hazard.trop_cyclone`, 375  
`climada.util.api_client`, 378  
`climada.util.checker`, 384  
`climada.util.config`, 384  
`climada.util.constants`, 384  
`climada.util.coordinates`, 386  
`climada.util.dates_times`, 401  
`climada.util.dwd_icon_loader`, 401  
`climada.util.files_handler`, 402  
`climada.util.finance`, 403  
`climada.util.hdf5_handler`, 404  
`climada.util.interpolation`, 405  
`climada.util.plot`, 405  
`climada.util.save`, 408  
`climada.util.scalebar_plot`, 408  
`climada.util.select`, 409  
`climada.util.value_representation`, 409  
`climada.util.yearsets`, 410



## Symbols

- `__init__()` (*climada.engine.cost\_benefit.CostBenefit* method), 293
  - `__init__()` (*climada.engine.forecast.Forecast* method), 297
  - `__init__()` (*climada.engine.impact.Impact* method), 302
  - `__init__()` (*climada.engine.impact.ImpactFreqCurve* method), 301
  - `__init__()` (*climada.engine.uncertainty\_quantification.calc\_base.CalcBase* method), 271
  - `__init__()` (*climada.engine.uncertainty\_quantification.calc\_cost\_benefit.CalcCostBenefit* method), 274
  - `__init__()` (*climada.engine.uncertainty\_quantification.calc\_impact.CalcImpact* method), 275
  - `__init__()` (*climada.engine.uncertainty\_quantification.input\_var.InputVar* method), 277
  - `__init__()` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* method), 288
  - `__init__()` (*climada.engine.uncertainty\_quantification.unc\_output.UncImpactOutput* method), 288
  - `__init__()` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* method), 282
  - `__init__()` (*climada.entity.disc\_rates.base.DiscRates* method), 314
  - `__init__()` (*climada.entity.entity\_def.Entity* method), 346
  - `__init__()` (*climada.entity.exposures.base.Exposures* method), 328
  - `__init__()` (*climada.entity.impact\_funcs.base.ImpactFunc* method), 336
  - `__init__()` (*climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet* method), 338
  - `__init__()` (*climada.entity.impact\_funcs.storm\_europe.ImpfStormEurope* method), 340
  - `__init__()` (*climada.entity.impact\_funcs.trop\_cyclone.ImpfTropCyclone* method), 340
  - `__init__()` (*climada.entity.measures.base.Measure* method), 342
  - `__init__()` (*climada.entity.measures.measure\_set.MeasureSet* method), 343
  - `__init__()` (*climada.entity.tag.Tag* method), 347
  - `__init__()` (*climada.hazard.base.Hazard* method), 355
  - `__init__()` (*climada.hazard.centroids.centroids.Centroids* method), 348
  - `__init__()` (*climada.hazard.storm\_europe.StormEurope* method), 362
  - `__init__()` (*climada.hazard.tag.Tag* method), 365
  - `__init__()` (*climada.hazard.tc\_tracks.TCTracks* method), 367
  - `__init__()` (*climada.hazard.trop\_cyclone.TropCyclone* method), 375
  - `__init__()` (*climada.util.api\_client.Client* method), 380
  - `__init__()` (*climada.util.api\_client.DataTypeInfo* method), 378
  - `__init__()` (*climada.util.api\_client.DatasetInfo* method), 379
  - `__init__()` (*climada.util.api\_client.FileInfo* method), 378
  - `data()` (*climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet* attribute), 337
  - `data()` (*climada.entity.measures.measure\_set.MeasureSet* attribute), 343
- ## A
- `aai_agg()` (*climada.engine.impact.Impact* attribute), 302
  - `activation_date` (*climada.util.api\_client.DatasetInfo* attribute), 379
  - `add_cntry_names()` (in module *climada.util.plot*), 407
  - `add_populated_places()` (in module *climada.util.plot*), 407
  - `add_sea()` (in module *climada.entity.exposures.base*), 335
  - `add_shapes()` (in module *climada.util.plot*), 407
  - `ai_agg()` (*climada.engine.forecast.Forecast* method), 297
  - `align_raster_data()` (in module *climada.util.coordinates*), 399
  - `append()` (*climada.entity.disc\_rates.base.DiscRates* method), 314
  - `append()` (*climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet* method), 338
  - `append()` (*climada.entity.measures.measure\_set.MeasureSet* method), 344

- [append\(\)](#) (*climada.entity.tag.Tag* method), 347  
[append\(\)](#) (*climada.hazard.base.Hazard* method), 360  
[append\(\)](#) (*climada.hazard.centroids.centri.Centroids* method), 351  
[append\(\)](#) (*climada.hazard.tag.Tag* method), 365  
[append\(\)](#) (*climada.hazard.tc\_tracks.TCTracks* method), 367  
[apply\(\)](#) (*climada.entity.measures.base.Measure* method), 343  
[apply\\_risk\\_transfer\(\)](#) (*climada.engine.cost\_benefit.CostBenefit* method), 294  
[area\\_pixel](#) (*climada.hazard.centroids.centri.Centroids* attribute), 348  
[array\\_default\(\)](#) (in module *climada.util.checker*), 384  
[array\\_optional\(\)](#) (in module *climada.util.checker*), 384  
[assign\\_centroids\(\)](#) (*climada.entity.exposures.base.Exposures* method), 329  
[assign\\_coordinates\(\)](#) (in module *climada.util.coordinates*), 392  
[assign\\_grid\\_points\(\)](#) (in module *climada.util.coordinates*), 392  
[assign\\_hazard\\_to\\_emdat\(\)](#) (in module *climada.engine.impact\_data*), 309  
[assign\\_track\\_to\\_em\(\)](#) (in module *climada.engine.impact\_data*), 310  
[at\\_event](#) (*climada.engine.impact.Impact* attribute), 302
- ## B
- [basin](#) (*climada.hazard.trop\_cyclone.TropCyclone* attribute), 375  
[benefit](#) (*climada.engine.cost\_benefit.CostBenefit* attribute), 292  
[BM\\_FILENAMES](#) (in module *climada.entity.exposures.litpop.nightlight*), 323  
[bounds](#) (*climada.hazard.tc\_tracks.TCTracks* property), 371
- ## C
- [Calc](#) (class in *climada.engine.uncertainty\_quantification.calc\_base*), 271  
[calc\(\)](#) (*climada.engine.cost\_benefit.CostBenefit* method), 293  
[calc\(\)](#) (*climada.engine.forecast.Forecast* method), 298  
[calc\(\)](#) (*climada.engine.impact.Impact* method), 303  
[calc\\_at\\_event](#) (*climada.engine.uncertainty\_quantification.calc\_impact.CalcImpact* attribute), 275  
[calc\\_eai\\_exp](#) (*climada.engine.uncertainty\_quantification.calc\_impact.CalcImpact* attribute), 275  
[calc\\_freq\\_curve\(\)](#) (*climada.engine.impact.Impact* method), 302  
[calc\\_impact\(\)](#) (*climada.entity.measures.base.Measure* method), 342  
[calc\\_impact\\_year\\_set\(\)](#) (*climada.engine.impact.Impact* method), 306  
[calc\\_mdr\(\)](#) (*climada.entity.impact\_funcs.base.ImpactFunc* method), 336  
[calc\\_perturbed\\_trajectories\(\)](#) (*climada.hazard.tc\_tracks.TCTracks* method), 371  
[calc\\_perturbed\\_trajectories\(\)](#) (in module *climada.hazard.tc\_tracks\_synth*), 373  
[calc\\_pixels\\_polygons\(\)](#) (*climada.hazard.centroids.centri.Centroids* method), 353  
[calc\\_random\\_walk\(\)](#) (*climada.hazard.tc\_tracks.TCTracks* method), 371  
[calc\\_risk\\_transfer\(\)](#) (*climada.engine.impact.Impact* method), 303  
[calc\\_scale\\_knutson\(\)](#) (in module *climada.hazard.tc\_clim\_change*), 366  
[calc\\_ssi\(\)](#) (*climada.hazard.storm\_europe.StormEurope* method), 363  
[calc\\_year\\_set\(\)](#) (*climada.hazard.base.Hazard* method), 360  
[CalcCostBenefit](#) (class in *climada.engine.uncertainty\_quantification.calc\_cost\_benefit*), 273  
[CalcImpact](#) (class in *climada.engine.uncertainty\_quantification.calc\_impact*), 275  
[calculate\\_correction\\_fac\(\)](#) (in module *climada.util.yearsets*), 411  
[calib\\_all\(\)](#) (in module *climada.engine.calibration\_opt*), 290  
[calib\\_cost\\_calc\(\)](#) (in module *climada.engine.calibration\_opt*), 290  
[calib\\_instance\(\)](#) (in module *climada.engine.calibration\_opt*), 289  
[calib\\_optimize\(\)](#) (in module *climada.engine.calibration\_opt*), 291  
[CAT\\_NAMES](#) (in module *climada.hazard.tc\_tracks*), 366  
[category](#) (*climada.hazard.trop\_cyclone.TropCyclone* attribute), 375  
[category\\_id](#) (*climada.entity.exposures.base.Exposures* attribute), 328  
[centr\\_](#) (*climada.entity.exposures.base.Exposures* attribute), 328  
[Centroids](#) (class in *climada.hazard.centroids.centri*), 347  
[centroids](#) (*climada.hazard.base.Hazard* attribute), 354  
[change\\_centroids\(\)](#) (*climada.hazard.base.Hazard* method), 361  
[change\\_impf\(\)](#) (in module *climada*), 361

`mada.engine.calibration_opt`), 290  
`check()` (`climada.entity.disc_rates.base.DiscRates` method), 314  
`check()` (`climada.entity.entity_def.Entity` method), 346  
`check()` (`climada.entity.exposures.base.Exposures` method), 328  
`check()` (`climada.entity.impact_funcs.base.ImpactFunc` method), 337  
`check()` (`climada.entity.impact_funcs.impact_func_set.ImpactFuncSet` method), 339  
`check()` (`climada.entity.measures.base.Measure` method), 342  
`check()` (`climada.entity.measures.measure_set.MeasureSet` method), 345  
`check()` (`climada.hazard.base.Hazard` method), 356  
`check()` (`climada.hazard.centroids.centri.Centroids` method), 348  
`check_assigned_track()` (in module `climada.engine.impact_data`), 310  
`check_distr()` (`climada.engine.uncertainty_quantification.climada.engine.uncertainty_quantification` method), 271  
`check_nl_local_file_exists()` (in module `climada.entity.exposures.litpop.nightlight`), 324  
`check_salib()` (`climada.engine.uncertainty_quantification.climada.engine.uncertainty_quantification` method), 283  
`check_sum` (`climada.util.api_client.FileInfo` attribute), 378  
`checkhash()` (in module `climada.util.api_client`), 379  
`checksize()` (in module `climada.util.api_client`), 379  
`clean_emdat_df()` (in module `climada.engine.impact_data`), 310  
`clear()` (`climada.entity.disc_rates.base.DiscRates` method), 314  
`clear()` (`climada.entity.impact_funcs.impact_func_set.ImpactFuncSet` method), 338  
`clear()` (`climada.entity.measures.measure_set.MeasureSet` method), 343  
`clear()` (`climada.hazard.base.Hazard` method), 355  
`clear()` (`climada.hazard.centroids.centri.Centroids` method), 351  
`Client` (class in `climada.util.api_client`), 379  
`Client.AmbiguousResult`, 380  
`Client.NoResult`, 380  
`climada.engine.calibration_opt` module, 289  
`climada.engine.cost_benefit` module, 292  
`climada.engine.forecast` module, 296  
`climada.engine.impact` module, 301  
`climada.engine.impact_data` module, 309  
`climada.engine.uncertainty_quantification.calc_cost_benefit` module, 271  
`climada.engine.uncertainty_quantification.calc_cost_benefit` module, 273  
`climada.engine.uncertainty_quantification.calc_impact` module, 275  
`climada.engine.uncertainty_quantification.input_var` module, 276  
`climada.engine.uncertainty_quantification.unc_output` module, 282  
`climada.entity.disc_rates.base` module, 313  
`climada.entity.entity_def` module, 346  
`climada.entity.exposures.base` module, 327  
`climada.entity.exposures.litpop.gpw_population` module, 316  
`climada.entity.exposures.litpop.litpop` module, 317  
`climada.entity.exposures.litpop.nightlight` module, 323  
`climada.entity.impact_funcs.base` module, 336  
`climada.entity.impact_funcs.impact_func_set` module, 337  
`climada.entity.impact_funcs.storm_europe` module, 340  
`climada.entity.impact_funcs.trop_cyclone` module, 340  
`climada.entity.measures.base` module, 341  
`climada.entity.measures.measure_set` module, 343  
`climada.entity.tag` module, 347  
`climada.hazard.base` module, 354  
`climada.hazard.centroids.centri` module, 347  
`climada.hazard.isimip_data` module, 362  
`climada.hazard.storm_europe` module, 362  
`climada.hazard.tag` module, 365  
`climada.hazard.tc_clim_change` module, 366  
`climada.hazard.tc_tracks` module, 366  
`climada.hazard.tc_tracks_synth` module, 373  
`climada.hazard.trop_cyclone` module, 375  
`climada.util.api_client`

module, 378  
 climada.util.checker  
   module, 384  
 climada.util.config  
   module, 384  
 climada.util.constants  
   module, 384  
 climada.util.coordinates  
   module, 386  
 climada.util.dates\_times  
   module, 401  
 climada.util.dwd\_icon\_loader  
   module, 401  
 climada.util.files\_handler  
   module, 402  
 climada.util.finance  
   module, 403  
 climada.util.hdf5\_handler  
   module, 404  
 climada.util.interpolation  
   module, 405  
 climada.util.plot  
   module, 405  
 climada.util.save  
   module, 408  
 climada.util.scalebar\_plot  
   module, 408  
 climada.util.select  
   module, 409  
 climada.util.value\_representation  
   module, 409  
 climada.util.yearsets  
   module, 410  
 color\_rgb (climada.engine.cost\_benefit.CostBenefit attribute), 292  
 color\_rgb (climada.entity.measures.base.Measure attribute), 341  
 combine\_measures() (climada.engine.cost\_benefit.CostBenefit method), 293  
 compute\_imp\_per\_year() (in module climada.util.yearsets), 411  
 concat() (climada.entity.exposures.base.Exposures static method), 335  
 concat() (climada.hazard.base.Hazard class method), 361  
 convert\_wgs\_to\_utm() (in module climada.util.coordinates), 389  
 coord (climada.hazard.centroids.centri.Centroids property), 354  
 coord\_exp (climada.engine.impact.Impact attribute), 302  
 coord\_on\_land() (in module climada.util.coordinates), 390

copy() (climada.entity.exposures.base.Exposures method), 335  
 cost (climada.entity.measures.base.Measure attribute), 341  
 cost\_ben\_ratio (climada.engine.cost\_benefit.CostBenefit attribute), 292  
 CostBenefit (class in climada.engine.cost\_benefit), 292  
 country\_faocode2iso() (in module climada.util.coordinates), 400  
 country\_iso2faocode() (in module climada.util.coordinates), 400  
 country\_iso2natid() (in module climada.util.coordinates), 394  
 country\_iso\_alpha2numeric() (in module climada.util.coordinates), 393  
 country\_natid2iso() (in module climada.util.coordinates), 393  
 country\_to\_iso() (in module climada.util.coordinates), 393  
 cover (climada.entity.exposures.base.Exposures attribute), 327  
 create\_lookup() (in module climada.engine.impact\_data), 309  
 crs (climada.entity.exposures.base.Exposures attribute), 327  
 crs (climada.entity.exposures.base.Exposures property), 328  
 crs (climada.hazard.centroids.centri.Centroids property), 354

## D

data (climada.hazard.tc\_tracks.TCTracks attribute), 366  
 data\_type (climada.util.api\_client.DatasetInfo attribute), 379  
 data\_type (climada.util.api\_client.DataTypeInfo attribute), 378  
 data\_type\_group (climada.util.api\_client.DataTypeInfo attribute), 378  
 DatasetInfo (class in climada.util.api\_client), 378  
 DataTypeInfo (class in climada.util.api\_client), 378  
 date (climada.engine.impact.Impact attribute), 302  
 date (climada.hazard.base.Hazard attribute), 354  
 date\_to\_str() (in module climada.util.dates\_times), 401  
 datetime64\_to\_ordinal() (in module climada.util.dates\_times), 401  
 deductible (climada.entity.exposures.base.Exposures attribute), 327  
 def\_file (climada.entity.entity\_def.Entity attribute), 346  
 delete\_icon\_grib() (in module climada.util.dwd\_icon\_loader), 402  
 DEM\_NODATA (in module climada.util.coordinates), 386



- DEMO\_DIR (in module *climada.util.constants*), 384  
 description (climada.entity.tag.Tag attribute), 347  
 description (climada.hazard.tag.Tag attribute), 365  
 description (climada.util.api\_client.DatasetInfo attribute), 379  
 description (climada.util.api\_client.DataTypeInfo attribute), 378  
 disc\_rates (climada.entity.entity\_def.Entity attribute), 346  
 DiscRates (class in *climada.entity.disc\_rates.base*), 313  
 dist\_approx() (in module *climada.util.coordinates*), 388  
 dist\_coast (climada.hazard.centroids.centri.Centroids attribute), 348  
 DIST\_DEF (in module *climada.util.interpolation*), 405  
 dist\_sqr\_approx() (in module *climada.util.interpolation*), 405  
 dist\_to\_coast() (in module *climada.util.coordinates*), 389  
 dist\_to\_coast\_nasa() (in module *climada.util.coordinates*), 390  
 distr\_dict (climada.engine.uncertainty\_quantification.calc\_base.CalcBase attribute), 273  
 distr\_dict (climada.engine.uncertainty\_quantification.input\_var.InputVar attribute), 273  
 distr\_dict (climada.engine.uncertainty\_quantification.unc\_output.UncOutput attribute), 282  
 DoesNotExist (climada.util.api\_client.Download attribute), 378  
 doi (climada.util.api\_client.DatasetInfo attribute), 379  
 Download (class in *climada.util.api\_client*), 378  
 Download.Failed, 378  
 download\_dataset() (climada.util.api\_client.Client method), 381  
 download\_file() (climada.util.api\_client.Client method), 381  
 download\_icon\_centroids\_file() (in module *climada.util.dwd\_icon\_loader*), 402  
 download\_icon\_grib() (in module *climada.util.dwd\_icon\_loader*), 401  
 download\_nl\_files() (in module *climada.entity.exposures.litpop.nightlight*), 324
- ## E
- eai\_exp (climada.engine.impact.Impact attribute), 302  
 EARTH\_RADIUS\_KM (in module *climada.util.constants*), 385  
 ei\_exp() (climada.engine.forecast.Forecast method), 297  
 elevation (climada.hazard.centroids.centri.Centroids attribute), 348  
 emdat\_countries\_by\_hazard() (in module *climada.engine.impact\_data*), 311  
 emdat\_impact\_event() (in module *climada.engine.impact\_data*), 312  
 emdat\_impact\_yearlysum() (in module *climada.engine.impact\_data*), 312  
 emdat\_possible\_hit() (in module *climada.engine.impact\_data*), 310  
 emdat\_to\_impact() (in module *climada.engine.impact\_data*), 312  
 empty\_geometry\_points() (climada.hazard.centroids.centri.Centroids method), 353  
 enddownload (climada.util.api\_client.Download attribute), 378  
 ent() (climada.engine.uncertainty\_quantification.input\_var.InputVar static method), 279  
 ENT\_DEMO\_FUTURE (in module *climada.util.constants*), 385  
 ENT\_DEMO\_TODAY (in module *climada.util.constants*), 384  
 ent\_fut\_input\_var (climada.engine.uncertainty\_quantification.calc\_cost\_benefit.CalcCostBenefit attribute), 273  
 ent\_input\_var (climada.engine.uncertainty\_quantification.calc\_cost\_benefit.CalcCostBenefit attribute), 273  
 ENT\_TEMPLATE\_XLS (in module *climada.util.constants*), 385  
 entfut() (climada.engine.uncertainty\_quantification.input\_var.InputVar static method), 281  
 Entity (class in *climada.entity.entity\_def*), 346  
 equal() (climada.hazard.centroids.centri.Centroids method), 348  
 equal\_crs() (in module *climada.util.coordinates*), 395  
 equal\_timestep() (climada.hazard.tc\_tracks.TCTracks method), 371  
 est\_comp\_time() (climada.engine.uncertainty\_quantification.calc\_base.CalcBase method), 271  
 evaluate() (climada.engine.uncertainty\_quantification.input\_var.InputVar method), 277  
 event\_date (climada.engine.forecast.Forecast attribute), 296  
 event\_id (climada.engine.impact.Impact attribute), 302  
 event\_id (climada.hazard.base.Hazard attribute), 354  
 event\_name (climada.engine.impact.Impact attribute), 302  
 event\_name (climada.hazard.base.Hazard attribute), 354  
 exp() (climada.engine.uncertainty\_quantification.input\_var.InputVar static method), 278  
 EXP\_DEMO\_H5 (in module *climada.util.constants*), 386  
 exp\_input\_var (climada.engine.uncertainty\_quantification.calc\_impact.CalcImpact attribute), 275  
 exp\_region\_id (climada.entity.measures.base.Measure attribute), 348

- attribute), 342
- expiration\_date (climada.util.api\_client.DatasetInfo attribute), 379
- exponents (climada.entity.exposures.litpop.litpop.LitPop attribute), 317
- exposure (climada.engine.forecast.Forecast attribute), 297
- exposure\_name (climada.engine.forecast.Forecast attribute), 297
- Exposures (class in climada.entity.exposures.base), 327
- exposures (climada.entity.entity\_def.Entity attribute), 346
- exposures\_set (climada.entity.measures.base.Measure attribute), 341
- extend() (climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet method), 339
- extend() (climada.entity.measures.measure\_set.MeasureSet method), 345
- extent (climada.hazard.tc\_tracks.TCTracks property), 371
- ## F
- fao\_code\_def() (in module climada.util.coordinates), 400
- file\_format (climada.util.api\_client.FileInfo attribute), 378
- file\_name (climada.entity.tag.Tag attribute), 347
- file\_name (climada.hazard.tag.Tag attribute), 365
- file\_name (climada.util.api\_client.FileInfo attribute), 378
- file\_size (climada.util.api\_client.FileInfo attribute), 378
- FileInfo (class in climada.util.api\_client), 378
- files (climada.util.api\_client.DatasetInfo attribute), 379
- fin\_mode (climada.entity.exposures.litpop.litpop.LitPop attribute), 317
- first\_year() (in module climada.util.dates\_times), 401
- Forecast (class in climada.engine.forecast), 296
- fraction (climada.hazard.base.Hazard attribute), 355
- frequency (climada.engine.impact.Impact attribute), 302
- frequency (climada.hazard.base.Hazard attribute), 355
- frequency\_from\_tracks() (climada.hazard.trop\_cyclone.TropCyclone method), 377
- from\_base\_grid() (climada.hazard.centroids.centri.Centroids static method), 348
- from\_geodataframe() (climada.hazard.centroids.centri.Centroids static method), 349
- from\_hdf5() (climada.engine.uncertainty\_quantification.uncertainty\_quantification.Input static method), 287
- from\_json() (climada.util.api\_client.DatasetInfo static method), 379
- from\_single\_track() (climada.hazard.trop\_cyclone.TropCyclone static method), 377
- from\_tracks() (climada.hazard.trop\_cyclone.TropCyclone static method), 375
- func (climada.engine.uncertainty\_quantification.input\_var.InputVar attribute), 277
- future\_year (climada.engine.cost\_benefit.CostBenefit attribute), 292
- ## G
- gdp() (in module climada.util.finance), 403
- generate\_centroids() (climada.hazard.tc\_tracks.TCTracks method), 371
- generate\_prob\_storms() (climada.hazard.storm\_europe.StormEurope method), 364
- geo\_bin\_from\_array() (in module climada.util.plot), 405
- geo\_im\_from\_array() (in module climada.util.plot), 406
- geometry (climada.entity.exposures.base.Exposures attribute), 327
- geometry (climada.hazard.centroids.centri.Centroids attribute), 348
- get\_admin1\_info() (in module climada.util.coordinates), 394
- get\_attributes\_with\_matching\_dimension() (in module climada.util.select), 409
- get\_bounds() (climada.hazard.tc\_tracks.TCTracks method), 371
- get\_closest\_point() (climada.hazard.centroids.centri.Centroids method), 351
- get\_coastlines() (in module climada.util.coordinates), 389
- get\_country\_code() (in module climada.util.coordinates), 394
- get\_country\_geometries() (in module climada.util.coordinates), 391
- get\_data\_type() (climada.util.api\_client.Client method), 381
- get\_data\_types() (climada.util.api\_client.Client method), 381
- get\_dataset() (climada.util.api\_client.Client method), 380
- get\_dataset\_by\_uuid() (climada.util.api\_client.Client method), 381
- get\_data\_type() (climada.util.api\_client.Client method), 380

<code>get_event_date()</code>	( <i>climada.hazard.base.Hazard</i> method), 359	<code>get_resolution_1d()</code>	(in module <i>climada.util.coordinates</i> ), 394
<code>get_event_id()</code>	( <i>climada.hazard.base.Hazard</i> method), 359	<code>get_samples_df()</code>	( <i>climada.engine.uncertainty_quantification.unc_output.UncOutput</i> method), 282
<code>get_event_name()</code>	( <i>climada.hazard.base.Hazard</i> method), 359	<code>get_sens_df()</code>	( <i>climada.engine.uncertainty_quantification.unc_output.UncOutput</i> method), 283
<code>get_exposures()</code>	( <i>climada.util.api_client.Client</i> method), 382	<code>get_sensitivity()</code>	( <i>climada.engine.uncertainty_quantification.unc_output.UncOutput</i> method), 284
<code>get_extent()</code>	( <i>climada.hazard.tc_tracks.TCTracks</i> method), 371	<code>get_sparse_csr_mat()</code>	(in module <i>climada.util.hdf5_handler</i> ), 404
<code>get_file_names()</code>	(in module <i>climada.util.files_handler</i> ), 403	<code>get_struct_ref()</code>	(in module <i>climada.util.hdf5_handler</i> ), 404
<code>get_func()</code>	( <i>climada.entity.impact_funcs.impact_func_set.ImpactFuncSet</i> method), 338	<code>get_string()</code>	(in module <i>climada.util.hdf5_handler</i> ), 404
<code>get_gpw_file_path()</code>	(in module <i>climada.entity.exposures.litpop.gpw_population</i> ), 316	<code>get_track()</code>	( <i>climada.hazard.tc_tracks.TCTracks</i> method), 367
<code>get_gridcellarea()</code>	(in module <i>climada.util.coordinates</i> ), 388	<code>get_unc_df()</code>	( <i>climada.engine.uncertainty_quantification.unc_output.UncOutput</i> method), 282
<code>get_hazard()</code>	( <i>climada.util.api_client.Client</i> method), 382	<code>get_uncertainty()</code>	( <i>climada.engine.uncertainty_quantification.unc_output.UncOutput</i> method), 284
<code>get_hazard_types()</code>	( <i>climada.entity.impact_funcs.impact_func_set.ImpactFuncSet</i> method), 339	<code>get_value_unit()</code>	(in module <i>climada.entity.exposures.litpop.litpop</i> ), 321
<code>get_hazard_types()</code>	( <i>climada.entity.measures.measure_set.MeasureSet</i> method), 344	<code>GLB_CENTROIDS_MAT</code>	(in module <i>climada.util.constants</i> ), 385
<code>get_ids()</code>	( <i>climada.entity.impact_funcs.impact_func_set.ImpactFuncSet</i> method), 339	<code>GLB_CENTROIDS_NC</code>	(in module <i>climada.util.constants</i> ), 385
<code>get_impf_column()</code>	( <i>climada.entity.exposures.base.Exposures</i> method), 329	<code>gpw_version</code>	( <i>climada.entity.exposures.litpop.litpop.LitPop</i> attribute), 317
<code>get_knutson_criterion()</code>	(in module <i>climada.hazard.tc_clim_change</i> ), 366	<code>GPW_VERSION</code>	(in module <i>climada.entity.exposures.litpop.litpop</i> ), 317
<code>get_land_geometry()</code>	(in module <i>climada.util.coordinates</i> ), 390	<code>grid_is_regular()</code>	(in module <i>climada.util.coordinates</i> ), 389
<code>get_largest_si()</code>	( <i>climada.engine.uncertainty_quantification.unc_output.UncOutput</i> method), 284	<code>gridpoints_core_calc()</code>	(in module <i>climada.entity.exposures.litpop.litpop</i> ), 322
<code>get_list_str_from_ref()</code>	(in module <i>climada.util.hdf5_handler</i> ), 404	<b>H</b>	
<code>get_litpop_default()</code>	( <i>climada.util.api_client.Client</i> method), 383	<code>haz()</code>	( <i>climada.engine.uncertainty_quantification.input_var.InputVar</i> static method), 278
<code>get_measure()</code>	( <i>climada.entity.measures.measure_set.MeasureSet</i> method), 344	<code>HAZ_DEMO_FL</code>	(in module <i>climada.util.constants</i> ), 385
<code>get_names()</code>	( <i>climada.entity.measures.measure_set.MeasureSet</i> method), 344	<code>HAZ_DEMO_H5</code>	(in module <i>climada.util.constants</i> ), 386
<code>get_region_gridpoints()</code>	(in module <i>climada.util.coordinates</i> ), 391	<code>HAZ_DEMO_MAT</code>	(in module <i>climada.util.constants</i> ), 385
<code>get_required_nl_files()</code>	(in module <i>climada.entity.exposures.litpop.nightlight</i> ), 324	<code>haz_input_var</code>	( <i>climada.engine.uncertainty_quantification.calc_cost_ben</i> attribute), 273
<code>get_resolution()</code>	(in module <i>climada.util.coordinates</i> ), 395	<code>haz_input_var</code>	( <i>climada.engine.uncertainty_quantification.calc_impact.C</i> attribute), 275
		<code>haz_model</code>	( <i>climada.engine.forecast.Forecast</i> attribute), 297
		<code>haz_summary_str()</code>	( <i>climada.engine.forecast.Forecast</i> method), 298
		<code>HAZ_TEMPLATE_XLS</code>	(in module <i>climada.util.constants</i> ), 385

**haz\_type** (*climada.entity.impact\_funcs.base.ImpactFunc* attribute), 336  
**haz\_type** (*climada.entity.measures.base.Measure* attribute), 341  
**haz\_type** (*climada.hazard.tag.Tag* attribute), 365  
**haz\_unc\_fut\_Var** (*climada.engine.uncertainty\_quantification.calc\_cost\_benefit.CalcCostBenefit* attribute), 273  
**Hazard** (class in *climada.hazard.base*), 354  
**hazard** (*climada.engine.forecast.Forecast* attribute), 297  
**hazard\_freq\_cutoff** (*climada.entity.measures.base.Measure* attribute), 341  
**hazard\_inten\_imp** (*climada.entity.measures.base.Measure* attribute), 342  
**hazard\_set** (*climada.entity.measures.base.Measure* attribute), 341  
**hit\_country\_per\_hazard()** (in module *climada.engine.impact\_data*), 309  
**I**  
**id** (*climada.entity.impact\_funcs.base.ImpactFunc* attribute), 336  
**id** (*climada.util.api\_client.Download* attribute), 378  
**imp\_fun\_map** (*climada.entity.measures.base.Measure* attribute), 342  
**imp\_mat** (*climada.engine.impact.Impact* attribute), 302  
**imp\_meas\_future** (*climada.engine.cost\_benefit.CostBenefit* attribute), 292  
**imp\_meas\_present** (*climada.engine.cost\_benefit.CostBenefit* attribute), 292  
**Impact** (class in *climada.engine.impact*), 301  
**impact** (*climada.engine.impact.ImpactFreqCurve* attribute), 301  
**impact\_funcs** (*climada.entity.entity\_def.Entity* attribute), 346  
**impact\_yearset()** (in module *climada.util.yearsets*), 410  
**impact\_yearset\_from\_sampling\_vect()** (in module *climada.util.yearsets*), 410  
**ImpactFreqCurve** (class in *climada.engine.impact*), 301  
**ImpactFunc** (class in *climada.entity.impact\_funcs.base*), 336  
**ImpactFuncSet** (class in *climada.entity.impact\_funcs.impact\_func\_set*), 337  
**impf\_** (*climada.entity.exposures.base.Exposures* attribute), 327  
**impf\_input\_var** (*climada.engine.uncertainty\_quantification.calc\_impact.CalcImpact* attribute), 275  
**impfset()** (*climada.engine.uncertainty\_quantification.input\_var.InputVar* static method), 279  
**ImpfStormEurope** (class in *climada.entity.impact\_funcs.storm\_europe*), 340  
**ImpfTropCyclone** (class in *climada.entity.impact\_funcs.trop\_cyclone*), 340  
**income\_group()** (in module *climada.util.finance*), 403  
**INDICATOR\_CENTR** (in module *climada.entity.exposures.base*), 335  
**INDICATOR\_IMP** (in module *climada.entity.exposures.base*), 335  
**init\_impact\_data()** (in module *climada.engine.calibration\_opt*), 290  
**init\_impf()** (in module *climada.engine.calibration\_opt*), 289  
**input\_var\_names** (*climada.engine.uncertainty\_quantification.calc\_cost\_benefit.CalcCostBenefit* attribute), 273  
**input\_var\_names** (*climada.engine.uncertainty\_quantification.calc\_impact.CalcImpact* attribute), 275  
**input\_vars** (*climada.engine.uncertainty\_quantification.calc\_base.CalcBase* property), 271  
**InputVar** (class in *climada.engine.uncertainty\_quantification.input\_var*), 276  
**intensity** (*climada.entity.impact\_funcs.base.ImpactFunc* attribute), 336  
**intensity** (*climada.hazard.base.Hazard* attribute), 355  
**intensity\_thres** (*climada.hazard.base.Hazard* attribute), 355  
**intensity\_thres** (*climada.hazard.storm\_europe.StormEurope* attribute), 362  
**intensity\_thres** (*climada.hazard.trop\_cyclone.TropCyclone* attribute), 375  
**intensity\_unit** (*climada.entity.impact\_funcs.base.ImpactFunc* attribute), 336  
**interp\_raster\_data()** (in module *climada.util.coordinates*), 397  
**interp\_index()** (in module *climada.util.interpolation*), 405  
**into\_datasets\_df()** (*climada.util.api\_client.Client* static method), 383  
**into\_files\_df()** (*climada.util.api\_client.Client* static method), 383  
**ISIMIP\_GPWV3\_NATID\_150AS** (in module *climada.util.constants*), 385  
**J**  
**join\_descriptions()** (*climada.hazard.tag.Tag* attribute), 365



*method*), 365  
 join\_file\_names() (*climada.hazard.tag.Tag* *method*), 365

## K

Key (*climada.engine.cost\_benefit.CostBenefit* *attribute*), 292

## L

label (*climada.engine.impact.ImpactFreqCurve* *attribute*), 301  
 labels (*climada.engine.uncertainty\_quantification.input\_var* *attribute*), 277  
 LANDFALL\_DECAY\_P (*in module climada.hazard.tc\_tracks\_synth*), 373  
 LANDFALL\_DECAY\_V (*in module climada.hazard.tc\_tracks\_synth*), 373  
 last\_year() (*in module climada.util.dates\_times*), 401  
 lat (*climada.hazard.centroids.centri* *Centroids* *attribute*), 347  
 latitude (*climada.entity.exposures.base.Exposures* *attribute*), 327  
 latlon\_bounds() (*in module climada.util.coordinates*), 387  
 latlon\_to\_geosph\_vector() (*in module climada.util.coordinates*), 386  
 lead\_time() (*climada.engine.forecast.Forecast* *method*), 298  
 license (*climada.util.api\_client.DatasetInfo* *attribute*), 379  
 LitPop (*class in climada.entity.exposures.litpop.litpop*), 317  
 load() (*in module climada.util.save*), 408  
 load\_gpw\_pop\_shape() (*in module climada.entity.exposures.litpop.gpw\_population*), 316  
 load\_nasa\_nl\_shape() (*in module climada.entity.exposures.litpop.nightlight*), 323  
 load\_nasa\_nl\_shape\_single\_tile() (*in module climada.entity.exposures.litpop.nightlight*), 325  
 load\_nightlight\_nasa() (*in module climada.entity.exposures.litpop.nightlight*), 325  
 load\_nightlight\_noaa() (*in module climada.entity.exposures.litpop.nightlight*), 326  
 local\_exceedance\_imp() (*climada.engine.impact.Impact* *method*), 307  
 local\_exceedance\_inten() (*climada.hazard.base.Hazard* *method*), 358  
 lon (*climada.hazard.centroids.centri* *Centroids* *attribute*), 347  
 lon\_bounds() (*in module climada.util.coordinates*), 387

lon\_normalize() (*in module climada.util.coordinates*), 387  
 longitude (*climada.entity.exposures.base.Exposures* *attribute*), 327

## M

make\_map() (*in module climada.util.plot*), 407  
 make\_sample() (*climada.engine.uncertainty\_quantification.calc\_base.CalcBase* *method*), 272  
 mask\_raster\_with\_geometry() (*in module climada.util.coordinates*), 400  
 match\_id() (*in module climada.engine.impact\_data*), 310  
 MAX\_DEM\_TILES\_DOWN (*in module climada.util.coordinates*), 386  
 MAX\_WAITING\_PERIOD (*climada.util.api\_client.Client* *attribute*), 380  
 mdd (*climada.entity.impact\_funcs.base.ImpactFunc* *attribute*), 336  
 mdd\_impact (*climada.entity.measures.base.Measure* *attribute*), 342  
 Measure (*class in climada.entity.measures.base*), 341  
 measures (*climada.entity.entity\_def.Entity* *attribute*), 346  
 MeasureSet (*class in climada.entity.measures.measure\_set*), 343  
 meta (*climada.entity.exposures.base.Exposures* *attribute*), 327  
 meta (*climada.hazard.centroids.centri* *Centroids* *attribute*), 347  
 METHOD (*in module climada.util.interpolation*), 405  
 metric\_names (*climada.engine.uncertainty\_quantification.calc\_cost\_benefit* *attribute*), 273  
 metric\_names (*climada.engine.uncertainty\_quantification.calc\_impact* *attribute*), 275  
 module  
   *climada.engine.calibration\_opt*, 289  
   *climada.engine.cost\_benefit*, 292  
   *climada.engine.forecast*, 296  
   *climada.engine.impact*, 301  
   *climada.engine.impact\_data*, 309  
   *climada.engine.uncertainty\_quantification.calc\_base*, 271  
   *climada.engine.uncertainty\_quantification.calc\_cost\_benefit*, 273  
   *climada.engine.uncertainty\_quantification.calc\_impact*, 275  
   *climada.engine.uncertainty\_quantification.input\_var*, 276  
   *climada.engine.uncertainty\_quantification.unc\_output*, 282  
   *climada.entity.disc\_rates.base*, 313  
   *climada.entity.entity\_def*, 346  
   *climada.entity.exposures.base*, 327

<code>climada.entity.exposures.litpop.gpw_population</code> , 316	<code>NASA_RESOLUTION_DEG</code> (in module <code>climada.entity.exposures.litpop.nightlight</code> ), 323
<code>climada.entity.exposures.litpop.litpop</code> , 317	<code>NASA_TILE_SIZE</code> (in module <code>climada.entity.exposures.litpop.nightlight</code> ), 323
<code>climada.entity.exposures.litpop.nightlight</code> , 323	<code>nat_earth_resolution()</code> (in module <code>climada.util.coordinates</code> ), 391
<code>climada.entity.impact_funcs.base</code> , 336	<code>NATEARTH_CENTROIDS</code> (in module <code>climada.util.constants</code> ), 385
<code>climada.entity.impact_funcs.impact_func_set</code> , 337	<code>natearth_country_to_int()</code> (in module <code>climada.util.coordinates</code> ), 394
<code>climada.entity.impact_funcs.storm_europe</code> , 340	<code>NE_CRS</code> (in module <code>climada.util.coordinates</code> ), 386
<code>climada.entity.impact_funcs.trop_cyclone</code> , 340	<code>NE_EPSG</code> (in module <code>climada.util.coordinates</code> ), 386
<code>climada.entity.measures.base</code> , 341	<code>net_present_value()</code> (in module <code>climada.entity.disc_rates.base.DiscRates</code> method), 314
<code>climada.entity.measures.measure_set</code> , 343	<code>net_present_value()</code> (in module <code>climada.util.finance</code> ), 403
<code>climada.entity.tag</code> , 347	<code>NOAA_BORDER</code> (in module <code>climada.entity.exposures.litpop.nightlight</code> ), 323
<code>climada.hazard.base</code> , 354	<code>NOAA_RESOLUTION_DEG</code> (in module <code>climada.entity.exposures.litpop.nightlight</code> ), 323
<code>climada.hazard.centroids.centri</code> , 347	
<code>climada.hazard.isimip_data</code> , 362	<b>O</b>
<code>climada.hazard.storm_europe</code> , 362	<code>on_land</code> ( <code>climada.hazard.centroids.centri.Centroids</code> attribute), 348
<code>climada.hazard.tag</code> , 365	<code>ONE_LAT_KM</code> (in module <code>climada.util.constants</code> ), 385
<code>climada.hazard.tc_clim_change</code> , 366	<code>orig</code> ( <code>climada.hazard.base.Hazard</code> attribute), 355
<code>climada.hazard.tc_tracks</code> , 366	
<code>climada.hazard.tc_tracks_synth</code> , 373	<b>P</b>
<code>climada.hazard.trop_cyclone</code> , 375	<code>paa</code> ( <code>climada.entity.impact_funcs.base.ImpactFunc</code> attribute), 336
<code>climada.util.api_client</code> , 378	<code>paa_impact</code> ( <code>climada.entity.measures.base.Measure</code> attribute), 342
<code>climada.util.checker</code> , 384	<code>param_labels</code> ( <code>climada.engine.uncertainty_quantification.unc_output.unc_output</code> attribute), 282
<code>climada.util.config</code> , 384	<code>param_labels</code> ( <code>climada.engine.uncertainty_quantification.unc_output.unc_output</code> property), 283
<code>climada.util.constants</code> , 384	<code>path</code> ( <code>climada.util.api_client.Download</code> attribute), 378
<code>climada.util.coordinates</code> , 386	<code>plot()</code> ( <code>climada.engine.impact.ImpactFreqCurve</code> method), 301
<code>climada.util.dates_times</code> , 401	<code>plot()</code> ( <code>climada.engine.uncertainty_quantification.input_var.InputVar</code> method), 278
<code>climada.util.dwd_icon_loader</code> , 401	<code>plot()</code> ( <code>climada.entity.disc_rates.base.DiscRates</code> method), 314
<code>climada.util.files_handler</code> , 402	<code>plot()</code> ( <code>climada.entity.exposures.base.Exposures</code> method), 333
<code>climada.util.finance</code> , 403	<code>plot()</code> ( <code>climada.entity.impact_funcs.base.ImpactFunc</code> method), 336
<code>climada.util.hdf5_handler</code> , 404	<code>plot()</code> ( <code>climada.entity.impact_funcs.impact_func_set.ImpactFuncSet</code> method), 339
<code>climada.util.interpolation</code> , 405	
<code>climada.util.plot</code> , 405	
<code>climada.util.save</code> , 408	
<code>climada.util.scalebar_plot</code> , 408	
<code>climada.util.select</code> , 409	
<code>climada.util.value_representation</code> , 409	
<code>climada.util.yearsets</code> , 410	
<b>N</b>	
<code>n_samples</code> ( <code>climada.engine.uncertainty_quantification.unc_output.unc_output</code> attribute), 282	
<code>n_samples</code> ( <code>climada.engine.uncertainty_quantification.unc_output.unc_output</code> property), 283	
<code>name</code> ( <code>climada.entity.impact_funcs.base.ImpactFunc</code> attribute), 336	
<code>name</code> ( <code>climada.entity.measures.base.Measure</code> attribute), 341	
<code>name</code> ( <code>climada.util.api_client.DatasetInfo</code> attribute), 379	

`plot()` (*climada.hazard.centroids.centri.Centroids* method), 353  
`plot()` (*climada.hazard.tc\_tracks.TCTracks* method), 371  
`plot_arrow_averted()` (*climada.engine.cost\_benefit.CostBenefit* method), 295  
`plot_basemap()` (*climada.entity.exposures.base.Exposures* method), 331  
`plot_basemap_eai_exposure()` (*climada.engine.impact.Impact* method), 305  
`plot_basemap_impact_exposure()` (*climada.engine.impact.Impact* method), 306  
`plot_cost_benefit()` (*climada.engine.cost\_benefit.CostBenefit* method), 294  
`plot_event_view()` (*climada.engine.cost\_benefit.CostBenefit* method), 294  
`plot_exceedence_prob()` (*climada.engine.forecast.Forecast* method), 299  
`plot_fraction()` (*climada.hazard.base.Hazard* method), 359  
`plot_hexbin()` (*climada.entity.exposures.base.Exposures* method), 330  
`plot_hexbin_eai_exposure()` (*climada.engine.impact.Impact* method), 303  
`plot_hexbin_ei_exposure()` (*climada.engine.forecast.Forecast* method), 300  
`plot_hexbin_impact_exposure()` (*climada.engine.impact.Impact* method), 305  
`plot_hist()` (*climada.engine.forecast.Forecast* method), 299  
`plot_imp_map()` (*climada.engine.forecast.Forecast* method), 298  
`plot_intensity()` (*climada.hazard.base.Hazard* method), 358  
`plot_raster()` (*climada.entity.exposures.base.Exposures* method), 331  
`plot_raster_eai_exposure()` (*climada.engine.impact.Impact* method), 304  
`plot_rp_imp()` (*climada.engine.impact.Impact* method), 307  
`plot_rp_intensity()` (*climada.hazard.base.Hazard* method), 358  
`plot_rp_uncertainty()` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* method), 285  
`plot_sample()` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* method), 284  
`plot_scatter()` (*climada.entity.exposures.base.Exposures* method), 330  
`plot_scatter_eai_exposure()` (*climada.engine.impact.Impact* method), 304  
`plot_sensitivity()` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* method), 285  
`plot_sensitivity_map()` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* method), 287  
`plot_sensitivity_second_order()` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* method), 286  
`plot_ssi()` (*climada.hazard.storm\_europe.StormEurope* method), 364  
`plot_uncertainty()` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* method), 285  
`plot_warn_map()` (*climada.engine.forecast.Forecast* method), 300  
`plot_waterfall()` (*climada.engine.cost\_benefit.CostBenefit* static method), 294  
`plot_waterfall_accumulated()` (*climada.engine.cost\_benefit.CostBenefit* method), 295  
`points_to_raster()` (in module *climada.util.coordinates*), 398  
`present_year` (*climada.engine.cost\_benefit.CostBenefit* attribute), 292  
`problem_sa` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* attribute), 282  
`problem_sa` (*climada.engine.uncertainty\_quantification.unc\_output.UncOutput* property), 283  
`properties` (*climada.util.api\_client.DatasetInfo* attribute), 379  
`properties` (*climada.util.api\_client.DataTypeInfo* attribute), 378  
`pts_to_raster_meta()` (in module *climada.util.coordinates*), 395  
`purge_cache()` (*climada.util.api\_client.Client* static method), 382

## R

`raster_to_meshgrid()` (in module *climada.util.coordinates*), 395  
`raster_to_vector()` (*climada.hazard.base.Hazard* method), 357  
`rates` (*climada.entity.disc\_rates.base.DiscRates* attribute), 314  
`read()` (in module *climada.util.hdf5\_handler*), 404  
`read_bm_file()` (in module *climada.entity.exposures.litpop.nightlight*), 326  
`read_cosmoe_file()` (*climada.hazard.storm\_europe.StormEurope*

method), 363

read\_csv() (climada.engine.impact.Impact method), 307

read\_excel() (climada.engine.impact.Impact method), 307

read\_excel() (climada.entity.disc\_rates.base.DiscRates method), 315

read\_excel() (climada.entity.entity\_def.Entity method), 346

read\_excel() (climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet method), 339

read\_excel() (climada.entity.measures.measure\_set.MeasureSet method), 345

read\_excel() (climada.hazard.base.Hazard method), 357

read\_excel() (climada.hazard.centroids.centri.Centroids method), 351

read\_footprints() (climada.hazard.storm\_europe.StormEurope method), 362

read\_hdf5() (climada.entity.exposures.base.Exposures method), 332

read\_hdf5() (climada.hazard.base.Hazard method), 360

read\_hdf5() (climada.hazard.centroids.centri.Centroids method), 353

read\_ibtracs\_netcdf() (climada.hazard.tc\_tracks.TCTracks method), 368

read\_icon\_grib() (climada.hazard.storm\_europe.StormEurope method), 363

read\_mat() (climada.entity.disc\_rates.base.DiscRates method), 315

read\_mat() (climada.entity.entity\_def.Entity method), 346

read\_mat() (climada.entity.exposures.base.Exposures method), 332

read\_mat() (climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet method), 340

read\_mat() (climada.entity.measures.measure\_set.MeasureSet method), 345

read\_mat() (climada.hazard.base.Hazard method), 357

read\_mat() (climada.hazard.centroids.centri.Centroids method), 350

read\_netcdf() (climada.hazard.tc\_tracks.TCTracks method), 372

read\_one\_gettelman() (climada.hazard.tc\_tracks.TCTracks method), 370

read\_processed\_ibtracs\_csv() (climada.hazard.tc\_tracks.TCTracks method), 370

read\_raster() (in module climada.util.coordinates), 396

read\_raster\_bounds() (in module climada.util.coordinates), 396

read\_raster\_sample() (in module climada.util.coordinates), 396

read\_simulations\_chaz() (climada.hazard.tc\_tracks.TCTracks method), 370

read\_simulations\_emanuel() (climada.hazard.tc\_tracks.TCTracks method), 370

read\_simulations\_storm() (climada.hazard.tc\_tracks.TCTracks method), 370

read\_sparse\_csr() (climada.engine.impact.Impact static method), 307

read\_vector() (in module climada.util.coordinates), 398

ref\_year (climada.entity.exposures.base.Exposures attribute), 327

refine\_raster\_data() (in module climada.util.coordinates), 397

region2isos() (in module climada.util.coordinates), 393

region\_id (climada.entity.exposures.base.Exposures attribute), 328

region\_id (climada.hazard.centroids.centri.Centroids attribute), 348

remove\_duplicate\_points() (climada.hazard.centroids.centri.Centroids method), 352

remove\_duplicates() (climada.hazard.base.Hazard method), 360

remove\_func() (climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet method), 338

remove\_measure() (climada.engine.cost\_benefit.CostBenefit method), 294

remove\_measure() (climada.entity.measures.measure\_set.MeasureSet method), 344

reproject\_input\_data() (in module climada.entity.exposures.litpop.litpop), 321

reproject\_raster() (climada.hazard.base.Hazard method), 356

reproject\_vector() (climada.hazard.base.Hazard method), 357

return\_per (climada.engine.impact.ImpactFreqCurve attribute), 301

risk\_aai\_agg() (in module climada.engine.cost\_benefit), 296

risk\_rp\_100() (in module climada.engine.cost\_benefit), 296

risk\_rp\_250() (in module climada.engine.cost\_benefit), 296



- mada.engine.cost\_benefit*), 296
- risk\_transf\_attach** (*climada.entity.measures.base.Measure* attribute), 342
- risk\_transf\_cost\_factor** (*climada.entity.measures.base.Measure* attribute), 342
- risk\_transf\_cover** (*climada.entity.measures.base.Measure* attribute), 342
- RIVER\_FLOOD\_REGIONS\_CSV** (in module *climada.util.constants*), 385
- rp** (*climada.engine.uncertainty\_quantification.calc\_impact* attribute), 275
- run\_datetime** (*climada.engine.forecast.Forecast* attribute), 296
- ## S
- SAFFIR\_SIM\_CAT** (in module *climada.hazard.tc\_tracks*), 366
- sample\_events()** (in module *climada.util.yearsets*), 411
- sample\_from\_poisson()** (in module *climada.util.yearsets*), 411
- samples\_df** (*climada.engine.uncertainty\_quantification.unc\_output* attribute), 282
- sampling\_kwargs** (*climada.engine.uncertainty\_quantification.unc\_output* property), 283
- sampling\_method** (*climada.engine.uncertainty\_quantification.unc\_output* attribute), 282
- sampling\_method** (*climada.engine.uncertainty\_quantification.unc\_output* property), 283
- sanitize\_event\_ids()** (*climada.hazard.base.Hazard* method), 359
- save()** (in module *climada.util.save*), 408
- scale\_bar()** (in module *climada.util.scalebar\_plot*), 408
- scale\_impact2refyear()** (in module *climada.engine.impact\_data*), 311
- select()** (*climada.engine.impact.Impact* method), 308
- select()** (*climada.entity.disc\_rates.base.DiscRates* method), 314
- select()** (*climada.hazard.base.Hazard* method), 358
- select()** (*climada.hazard.centroids.centri.Centroids* method), 353
- sensitivity()** (*climada.engine.uncertainty\_quantification.calc\_base.CalcBase* method), 272
- sensitivity\_metrics** (*climada.engine.uncertainty\_quantification.unc\_output* property), 283
- set\_area\_approx()** (*climada.hazard.centroids.centri.Centroids* method), 352
- set\_area\_pixel()** (*climada.hazard.centroids.centri.Centroids* method), 352
- set\_category()** (in module *climada.hazard.tc\_tracks*), 372
- set\_climate\_scenario\_knu()** (*climada.hazard.trop\_cyclone.TropCyclone* method), 376
- set\_countries()** (*climada.entity.exposures.litpop.litpop.LitPop* method), 317
- set\_country()** (*climada.entity.exposures.litpop.litpop.LitPop* method), 320
- set\_crs()** (*climada.entity.exposures.base.Exposures* method), 328
- set\_custom\_shape()** (*climada.entity.exposures.litpop.litpop.LitPop* method), 320
- set\_custom\_shape\_from\_countries()** (*climada.entity.exposures.litpop.litpop.LitPop* method), 319
- set\_df\_geometry\_points()** (in module *climada.util.coordinates*), 400
- set\_dist\_coast()** (*climada.hazard.centroids.centri.Centroids* method), 352
- set\_elevation()** (*climada.hazard.centroids.centri.Centroids* method), 352
- set\_emanuel\_usa()** (*climada.entity.impact\_funcs.trop\_cyclone.ImpfTropCyclone* method), 340
- set\_frequency()** (*climada.hazard.base.Hazard* method), 360
- set\_from\_raster()** (*climada.entity.exposures.base.Exposures* method), 329
- set\_from\_tracks()** (*climada.hazard.trop\_cyclone.TropCyclone* method), 375
- set\_gdf()** (*climada.entity.exposures.base.Exposures* method), 329
- set\_geometry\_points()** (*climada.entity.exposures.base.Exposures* method), 329
- set\_geometry\_points()** (*climada.hazard.centroids.centri.Centroids* method), 354
- set\_lat\_lon()** (*climada.entity.exposures.base.Exposures* method), 329
- set\_lat\_lon()** (*climada.hazard.centroids.centri.Centroids* method), 349
- set\_lat\_lon\_to\_meta()** (*climada.entity.exposures.base.Exposures* method), 329

- mada.hazard.centroids.centri.Centroids* method), 353
- `set_meta_to_lat_lon()` (*climada.hazard.centroids.centri.Centroids* method), 353
- `set_nightlight_intensity()` (*climada.entity.exposures.litpop.litpop.LitPop* method), 318
- `set_on_land()` (*climada.hazard.centroids.centri.Centroids* method), 352
- `set_population()` (*climada.entity.exposures.litpop.litpop.LitPop* method), 318
- `set_raster()` (*climada.hazard.base.Hazard* method), 356
- `set_raster_file()` (*climada.hazard.centroids.centri.Centroids* method), 350
- `set_raster_from_pix_bounds()` (*climada.hazard.centroids.centri.Centroids* method), 349
- `set_raster_from_pnt_bounds()` (*climada.hazard.centroids.centri.Centroids* method), 349
- `set_region_id()` (*climada.hazard.centroids.centri.Centroids* method), 352
- `set_schwierz()` (*climada.entity.impact\_funcs.storm\_europe.ImpfStormEurope* method), 340
- `set_sens_df()` (*climada.engine.uncertainty\_quantification.unc\_output\_unc\_output* method), 283
- `set_sigmoid_impf()` (*climada.entity.impact\_funcs.base.ImpactFunc* method), 337
- `set_ssi()` (*climada.hazard.storm\_europe.StormEurope* method), 364
- `set_step_impf()` (*climada.entity.impact\_funcs.base.ImpactFunc* method), 337
- `set_unc_df()` (*climada.engine.uncertainty\_quantification.unc\_output\_unc\_output* method), 283
- `set_vector()` (*climada.hazard.base.Hazard* method), 356
- `set_vector_file()` (*climada.hazard.centroids.centri.Centroids* method), 350
- `set_welker()` (*climada.entity.impact\_funcs.storm\_europe.ImpfStormEurope* method), 340
- `shape` (*climada.hazard.centroids.centri.Centroids* property), 354
- `shape()` (in module *climada.util.checker*), 384
- `sig_dig()` (in module *climada.util.value\_representation*), 409
- `sig_dig_list()` (in module *climada.util.value\_representation*), 409
- `size` (*climada.hazard.base.Hazard* property), 360
- `size` (*climada.hazard.centroids.centri.Centroids* property), 354
- `size` (*climada.hazard.tc\_tracks.TCTracks* property), 371
- `size()` (*climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet* method), 339
- `size()` (*climada.entity.measures.measure\_set.MeasureSet* method), 344
- `size()` (in module *climada.util.checker*), 384
- `ssi` (*climada.hazard.storm\_europe.StormEurope* attribute), 362, 364
- `ssi_dawkins` (*climada.hazard.storm\_europe.StormEurope* self attribute), 364
- `ssi_wisc` (*climada.hazard.storm\_europe.StormEurope* attribute), 362
- `startdownload` (*climada.util.api\_client.Download* attribute), 378
- `status` (*climada.util.api\_client.DatasetInfo* attribute), 379
- `StormEurope` (class in *climada.hazard.storm\_europe*), 362
- `str_to_date()` (in module *climada.util.dates\_times*), 401
- `subraster_from_bounds()` (in module *climada.util.coordinates*), 398
- `subset()` (*climada.hazard.tc\_tracks.TCTracks* method), 367
- `summary_str()` (*climada.engine.forecast.Forecast* method), 388
- `SYSTEM_DIR` (in module *climada.util.constants*), 384
- ## T
- `Tag` (class in *climada.entity.tag*), 347
- `Tag` (class in *climada.hazard.tag*), 365
- `tag` (*climada.engine.impact.Impact* attribute), 301
- `tag` (*climada.engine.impact.ImpactFreqCurve* attribute), 301
- `tag` (*climada.entity.disc\_rates.base.DiscRates* attribute), 327
- `tag` (*climada.entity.exposures.base.Exposures* attribute), 327
- `tag` (*climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet* attribute), 337
- `tag` (*climada.entity.measures.measure\_set.MeasureSet* attribute), 343
- `tag` (*climada.hazard.base.Hazard* attribute), 354
- `TC_ANDREW_FL` (in module *climada.util.constants*), 385
- `TCTracks` (class in *climada.hazard.tc\_tracks*), 366
- `TEST_UNC_OUTPUT_COSTBEN` (in module *climada.util.constants*), 386
- `TEST_UNC_OUTPUT_IMPACT` (in module *climada.util.constants*), 386

**TMP\_ELEVATION\_FILE** (in module *climada.util.coordinates*), 386  
**to\_crs()** (*climada.entity.exposures.base.Exposures* method), 332  
**to\_crs\_user\_input()** (in module *climada.util.coordinates*), 395  
**to\_exposures()** (*climada.util.api\_client.Client* method), 383  
**to\_geodataframe()** (*climada.hazard.tc\_tracks.TCTracks* method), 372  
**to\_hazard()** (*climada.util.api\_client.Client* method), 382  
**to\_hdf5()** (*climada.engine.uncertainty\_quantification.unc\_output* method), 287  
**to\_list()** (in module *climada.util.files\_handler*), 402  
**tot\_climate\_risk** (*climada.engine.cost\_benefit.CostBenefit* attribute), 292  
**TOT\_RADIATIVE\_FORCE** (in module *climada.hazard.tc\_clim\_change*), 366  
**tot\_value** (*climada.engine.impact.Impact* attribute), 302  
**total\_bounds** (*climada.hazard.centroids.centr.Centroids* property), 354  
**tracks\_in\_exp()** (*climada.hazard.tc\_tracks.TCTracks* method), 367  
**TropCyclone** (class in *climada.hazard.trop\_cyclone*), 375

## U

**UncCostBenefitOutput** (class in *climada.engine.uncertainty\_quantification.unc\_output*), 287  
**uncertainty()** (*climada.engine.uncertainty\_quantification.calc\_cost\_benefit.CalcCostBenefit* method), 274  
**uncertainty()** (*climada.engine.uncertainty\_quantification.calc\_impact.CalcImpact* method), 276  
**uncertainty\_metrics** (*climada.engine.uncertainty\_quantification.unc\_output* property), 283  
**UncImpactOutput** (class in *climada.engine.uncertainty\_quantification.unc\_output*), 288  
**UncOutput** (class in *climada.engine.uncertainty\_quantification.unc\_output*), 282  
**union()** (*climada.hazard.centroids.centr.Centroids* method), 351  
**unit** (*climada.engine.cost\_benefit.CostBenefit* attribute), 292  
**unit** (*climada.engine.impact.Impact* attribute), 302  
**unit** (*climada.engine.impact.ImpactFreqCurve* attribute), 301  
**units** (*climada.hazard.base.Hazard* attribute), 354  
**untar\_noaa\_stable\_nightlight()** (in module *climada.entity.exposures.litpop.nightlight*), 326  
**unzip\_tif\_to\_py()** (in module *climada.entity.exposures.litpop.nightlight*), 326  
**url** (*climada.util.api\_client.Download* attribute), 378  
**url** (*climada.util.api\_client.FileInfo* attribute), 378  
**utm\_zones()** (in module *climada.util.coordinates*), 389  
**uuid** (*climada.util.api\_client.DatasetInfo* attribute), 379  
**uuid** (*climada.util.api\_client.FileInfo* attribute), 378

## V

**value** (*climada.engine.cost\_benefit.CostBenefit* attribute), 292  
**value** (*climada.entity.exposures.base.Exposures* attribute), 327  
**value\_to\_monetary\_unit()** (in module *climada.util.value\_representation*), 409  
**value\_unit** (*climada.engine.uncertainty\_quantification.calc\_cost\_benefit.CalcCostBenefit* attribute), 273  
**value\_unit** (*climada.engine.uncertainty\_quantification.calc\_impact.CalcImpact* attribute), 275  
**value\_unit** (*climada.entity.exposures.base.Exposures* attribute), 327  
**var\_to\_inputvar()** (*climada.engine.uncertainty\_quantification.input\_var.InputVar* static method), 278  
**vars\_check** (*climada.hazard.centroids.centr.Centroids* attribute), 348  
**vars\_def** (*climada.entity.exposures.base.Exposures* attribute), 328  
**vars\_def** (*climada.hazard.base.Hazard* attribute), 355  
**vars\_oblig** (*climada.entity.exposures.base.Exposures* attribute), 328  
**vars\_oblig** (*climada.hazard.base.Hazard* attribute), 355  
**vars\_opt** (*climada.entity.exposures.base.Exposures* attribute), 328  
**vars\_opt** (*climada.hazard.base.Hazard* attribute), 355  
**vars\_opt** (*climada.hazard.storm\_europe.StormEurope* attribute), 362  
**vars\_opt** (*climada.hazard.trop\_cyclone.TropCyclone* attribute), 375  
**vector\_to\_raster()** (*climada.hazard.base.Hazard* method), 357  
**version** (*climada.util.api\_client.DatasetInfo* attribute), 379  
**video\_direct\_impact()** (*climada.engine.impact.Impact* static method), 307  
**video\_intensity()** (*climada.hazard.trop\_cyclone.TropCyclone* static method), 376

`vulnerability` (*climada.engine.forecast.Forecast* attribute), 297

## W

`write_csv()` (*climada.engine.impact.Impact* method), 306

`write_excel()` (*climada.engine.impact.Impact* method), 306

`write_excel()` (*climada.entity.disc\_rates.base.DiscRates* method), 315

`write_excel()` (*climada.entity.entity\_def.Entity* method), 346

`write_excel()` (*climada.entity.impact\_funcs.impact\_func\_set.ImpactFuncSet* method), 340

`write_excel()` (*climada.entity.measures.measure\_set.MeasureSet* method), 345

`write_hdf5()` (*climada.entity.exposures.base.Exposures* method), 332

`write_hdf5()` (*climada.hazard.base.Hazard* method), 360

`write_hdf5()` (*climada.hazard.centroids.centri.Centroids* method), 353

`write_netcdf()` (*climada.hazard.tc\_tracks.TCTracks* method), 372

`write_raster()` (*climada.entity.exposures.base.Exposures* method), 335

`write_raster()` (*climada.hazard.base.Hazard* method), 360

`write_raster()` (in module *climada.util.coordinates*), 398

`write_sparse_csr()` (*climada.engine.impact.Impact* method), 306

`WS_DEMO_NC` (in module *climada.util.constants*), 386

## Y

`years` (*climada.entity.disc\_rates.base.DiscRates* attribute), 313