CLIMADA documentation

Release 1.5.0

CLIMADA contributors

Sep 24, 2020

CONTENTS

This is the documentation for version v1.5.0. In CLIMADA-project you will find CLIMADA's contributors, repository and scientific publications.

CHAPTER

ONE

INTRODUCTION

CLIMADA implements a fully probabilistic risk assessment model. According to the [IPCC2014], 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.

CHAPTER

TWO

INSTALLATION

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

2.1 Download

Download the last CLIMADA release available in climada releases as a zip or tar.gz file. Uncompress it to your local computer. Hereinafter `climada_python-x.y.z` refers to the downloaded folder of CLIMADA version x.y.z.

2.2 Unix, Mac and Windows Operating Systems

2.2.1 Install environment with Anaconda

1. **Anaconda**: Download or update to the latest version of Anaconda. Execute it.
2. **Install dependencies**: In the *Environments* section, use the *Import* box to create a new virtual environment from a yml file. A dialogue box will ask you for the location of the file. Provide first the path of climada's `climada_python-x.y.z/requirements/env_climada.yml`. The default name of the environment, *climada_env*, appears. Click the *Import* button to start the installation.

   The installation of the packages will take some minutes. No dialogue box should appear in the meantime. If an error happens, try to solve it looking into the details description.

   Finally, set the `climada_python-x.y.z` folder path into the environment using the following command:

   ```
   source activate climada_env
   conda develop /your/path/to/climada_python-x.y.z/
   conda deactivate
   ```

   You can also do so by clicking on the green right arrow in the Anaconda GUI in the *Environments* section right to the 'climada_env' environment, select 'Open Terminal' and execute the following line:

   ```
   conda develop /your/path/to/climada_python-x.y.z/
   ```

3. **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 2min).
4. **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 *Tutorial* for more information.

2.3 Unix and Mac Operating Systems

2.3.1 Install environment with Miniconda

1. **Miniconda**: Download or update to the latest version of Miniconda.
2. **Install dependencies**: Create the virtual environment *climada_env* with climada's dependencies:

```
cd climada_python-x.y.z
conda env create -f requirements/env_climada.yml --name climada_env
```

   Finally, set the `climada_python-x.y.z` folder path into the environment using the following command:

```
source activate climada_env
conda develop /your/path/to/climada_python-x.y.z/
conda deactivate
```

3. **Test installation**: Activate the environment, execute the installation tests and deactivate the environment when finished using climada:

```
source activate climada_env
python3 tests_install.py
source deactivate
```

   If the installation has been successful, an OK will appear at the end (the execution should last less than 2min).

4. **Run tutorials**: Install and launch *jupyter notebook* in the same environment:

```
source activate climada_env
conda install jupyter
jupyter notebook --notebook-dir /path/to/climada_python-x.y.z
```

A browser window will show up. Open `climada_python-x.y.z/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 *Tutorial* for more information.

2.4 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:

```
source 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 right problems in macOS Mojave: try the solutions suggested here https://github.com/conda/conda/issues/8440.

CHAPTER

THREE

TUTORIAL

The main tutorial walks you through all the functionalities of this version of CLIMADA. There, you will find the links to additional tutorials for specific features of CLIMADA, such as different hazard and exposure models. You can execute it by opening `climada_python-x.y.z/doc/tutorial/1_main_climada.ipynb` with Jupyter Notebook and the CLIMADA environment (*climada_env*) activated (i.e. CLIMADA needs to be installed as in *Installation*).

Navigate through the tutorial here:

3.1 CLIMADA features

The functionality of climada is gathered in the following classes:

- **Entity**: socio-economic models:
- Exposures: exposed values
  - BlackMarble: regional economic model from nightlight intensities and economic indicators (GDP, income group)
  - LitPop: regional economic model using nightlight and population maps together with several economic indicators
- ImpactFuncSet: collection of impact functions per hazard
  - ImpactFunc: one adjustable impact function
  - IFTropCyclone: definition of impact functions for tropical cyclones
- DiscRates: discount rates per year
- MeasureSet: collection of measures for adaptation
  - Measure: one configurable measure
- **Hazard**: meteorological models:
- TropCyclone: tropical cyclone events
- **Impact**: impacts of the Hazard and Entity interaction.
- **CostBenefit**: adaptation options appraisal.
- **Add-ons**: OpenStreetMap and Google Earth Engine routines.

3.2 Risk assessment

3.2.1 Entity

The entity class is just a container for the exposures, impact functions, discount rates and measures. It can be directly filled from an excel file following climada's template or from MATLAB files of the climada MATLAB version. The excel template can be found in `climada_python/data/system/entity_template.xlsx`.

[1]:
```
from climada.entity import Entity
from climada.util.constants import ENT_DEMO_TODAY

# absolute path of file following template.
ent_file = ENT_DEMO_TODAY
ent_fl = Entity()
ent_fl.read_excel(ent_file)
```

```
2020-09-16 14:51:40,441 - climada - DEBUG - Loading default config file: /home/tovogt/
↪code/climada_python/climada/conf/defaults.conf
```

Every class has a `check()` method. This verifies that the necessary data to compute the impact is correctly provided and logs the optional variables that are not present. Use it always after filling an instance.

[2]:
```
ent_fl.check() # checks exposures, impact functions, discount rates and measures
```

```
2020-09-16 14:51:41,572 - climada.entity.exposures.base - INFO - crs set to default␣
↪value: {'init': 'epsg:4326', 'no_defs': True}
2020-09-16 14:51:41,572 - climada.entity.exposures.base - INFO - ref_year metadata␣
↪set to default value: 2018
2020-09-16 14:51:41,573 - climada.entity.exposures.base - INFO - value_unit metadata␣
↪set to default value: USD
2020-09-16 14:51:41,573 - climada.entity.exposures.base - INFO - meta metadata set to␣
↪default value: None
2020-09-16 14:51:41,573 - climada.entity.exposures.base - INFO - centr_ not set.
2020-09-16 14:51:41,574 - climada.entity.exposures.base - INFO - category_id not set.
2020-09-16 14:51:41,574 - climada.entity.exposures.base - INFO - region_id not set.
2020-09-16 14:51:41,575 - climada.entity.exposures.base - INFO - geometry not set.
```

Exposures

The `Entity`'s `exposures` attribute contains geolocalized values of anything exposed to the hazard, let it be monetary value of assets or number of human lifes, for example. It is of type `Exposures`.

See Exposures tutorial to learn how to fill and use exposures.

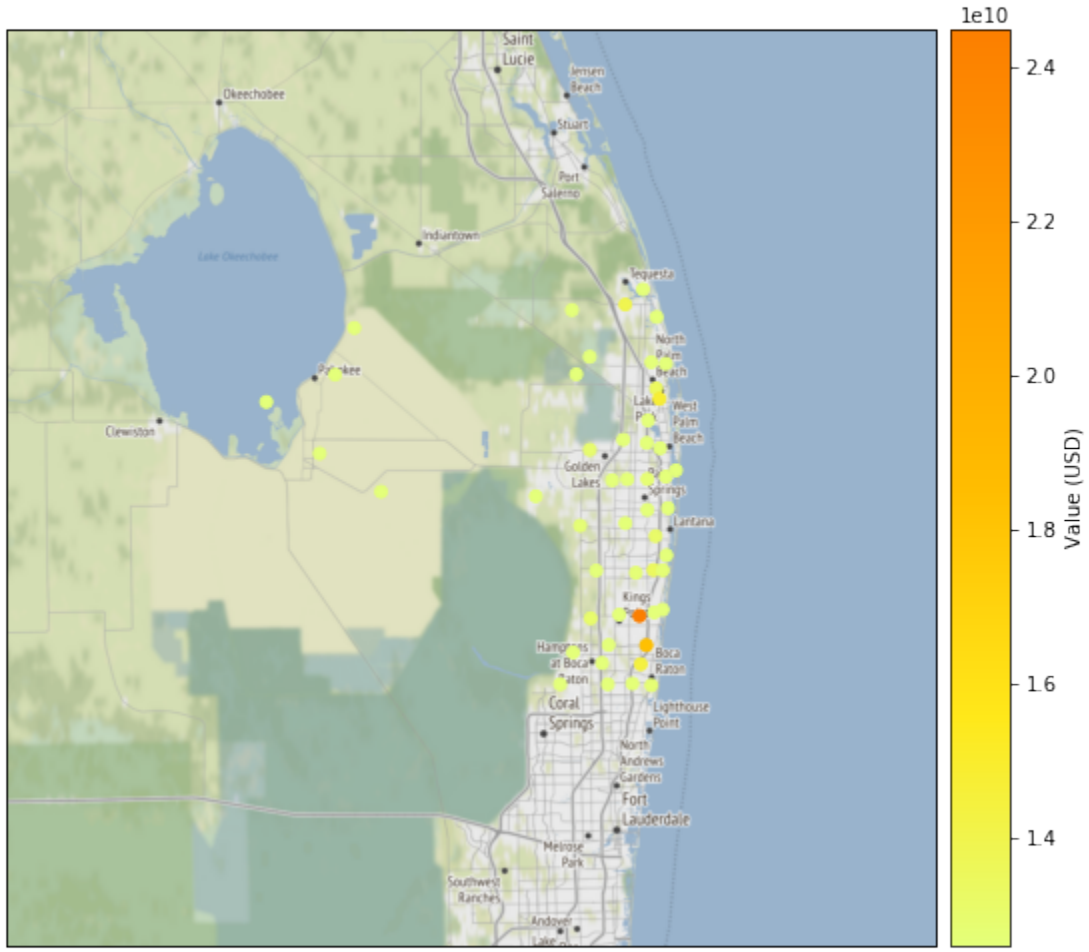
See LitPop to model economic exposures using night-time light and population densitites. See BlackMarble to model economic exposures based only on night-time light intensities. To combine your exposure with OpenStreetMap's data see OSM.

[3]:
```
%matplotlib inline
ent_fl.exposures.plot_basemap(buffer=50000.0); # exposures in Florida
```

```
2020-09-16 14:51:41,579 - climada.util.coordinates - INFO - Setting geometry points.
2020-09-16 14:51:41,588 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
/home/tovogt/code/climada_python/climada/util/plot.py:314: UserWarning: Tight layout␣
↪not applied. The left and right margins cannot be made large enough to accommodate␣
↪all axes decorations.
  fig.tight_layout()
/home/tovogt/.local/share/miniconda3/envs/tc_env/lib/python3.7/site-packages/
↪contextily/tile.py:199: FutureWarning: The url format using 'tileX', 'tileY',␣
↪'tileZ' as placeholders is deprecated. Please use '{x}', '{y}', '{z}' instead.
  FutureWarning,
```

```
2020-09-16 14:52:05,354 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

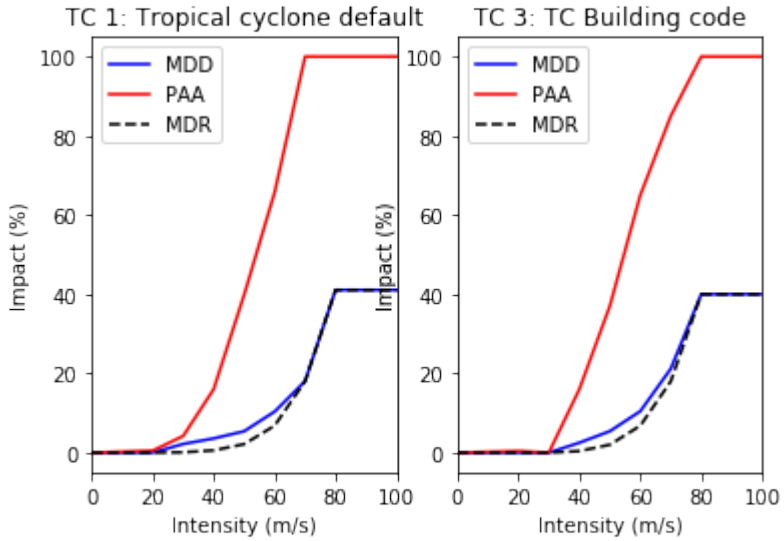


Impact Functions

The `impact_funcs` attribute is of type `ImpactFuncSet`. As such, it contains impact functions for different hazards.

See Impact Functions tutorial to learn how to handle this class.

[4]:
```
ent_fl.impact_funcs.plot('TC'); # tropical cyclone impact functions
```



Adaptation Measures

The `measures` attribute is of type `MeasureSet`. This class is a container of `Measure` instances, similarly to `ImpactFuncSet` containing several `ImpactFunc`. Adaptation measures aim to decrease hazards impacts and are subjected to a cost.

See Adaptation Measures to learn to handle measures.

[5]:
```
# print measures names
print(ent_fl.measures.get_names())
```

```
{'TC': ['Mangroves', 'Beach nourishment', 'Seawall', 'Building code']}
```

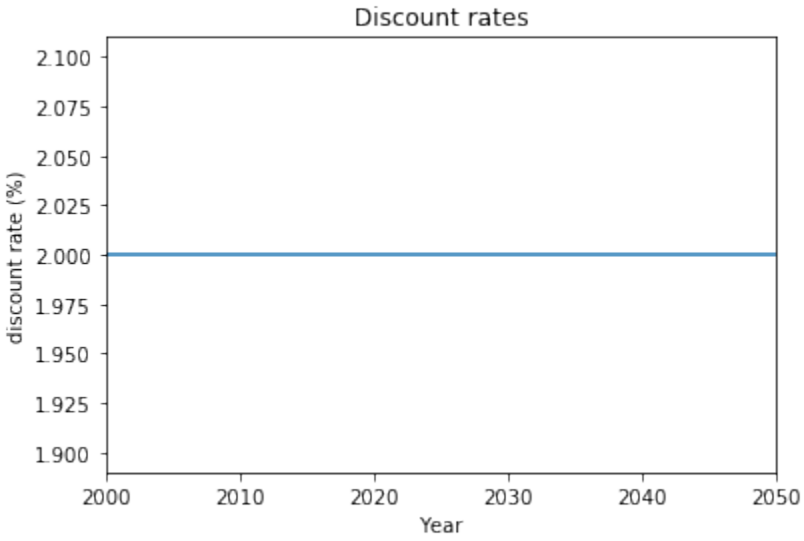
Discount Rates

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 Discount Rates.

[6]:
```
ent_fl.disc_rates.plot()
```

[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ad1a65048>
```



3.2.2 Hazard

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

See Hazard to learn how to handle hazards.

See TropCyclone to learn to model tropical cyclones. TCSurge implements an approximation on tropical cyclones surges. StromEurope creates a hazard event set for extratropical cylones or winter windstorms in Europe.

To use satellite images in your models follow the tutorial Google Earth Engine.

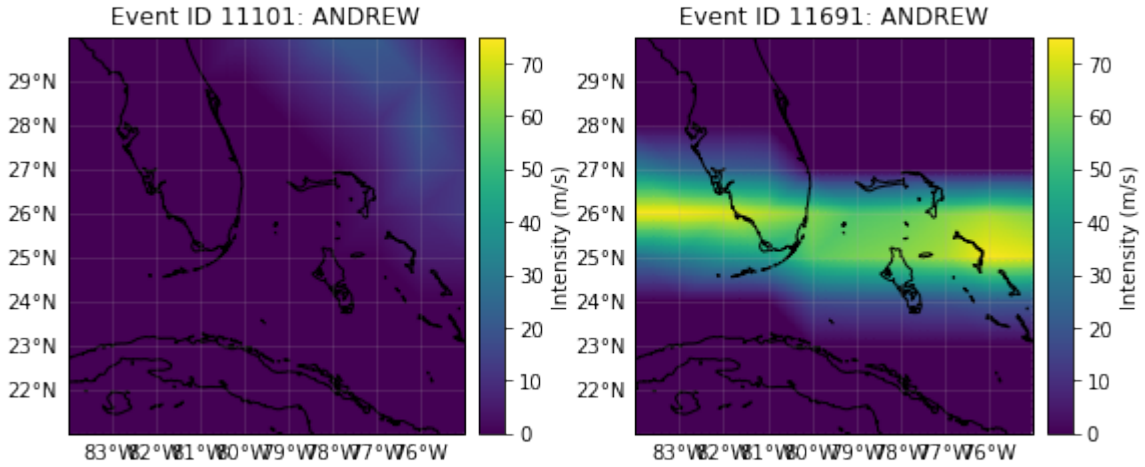
A complete set of tropical cyclones events in Florida can be found in file `HAZ_DEMO_MAT`. This contains 1445 historical events from year 1851 to 2011 and 9 synthetic events for each historical one.

[7]:
```
from climada.hazard import Hazard
from climada.util import HAZ_DEMO_MAT
tc_fl = Hazard('TC')
tc_fl.read_mat(HAZ_DEMO_MAT, 'Historic and synthetic tropical cyclones in Florida␣
↪from 1851 to 2011.')
tc_fl.event_name = [""] * tc_fl.event_id.size # storm names are missing from the demo␣
↪data set
tc_fl.event_name[11100] = "ANDREW"
tc_fl.event_name[11690] = "ANDREW"
tc_fl.plot_intensity('ANDREW') # plot intensity of hurricanes Andrew
print('Two hurricanes called Andrew happened in ', tc_fl.get_event_date('ANDREW'))
```

```
2020-09-16 14:52:06,149 - climada.hazard.base - INFO - Reading /home/tovogt/code/
↪climada_python/data/demo/atl_prob_nonames.mat
2020-09-16 14:52:06,172 - climada.hazard.centroids.centr - INFO - Reading /home/
↪tovogt/code/climada_python/data/demo/atl_prob_nonames.mat
```

```
/home/tovogt/code/climada_python/climada/util/plot.py:314: UserWarning: Tight layout␣
↪not applied. The left and right margins cannot be made large enough to accommodate␣
↪all axes decorations.
  fig.tight_layout()
```

```
Two hurricanes called Andrew happened in  ['1986-06-05', '1992-08-16']
```



3.2.3 Impact

The impact of hazard events over an entity can be computed easily from the previously explained classes. By computing the impact for each event (historical and synthetic), 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.

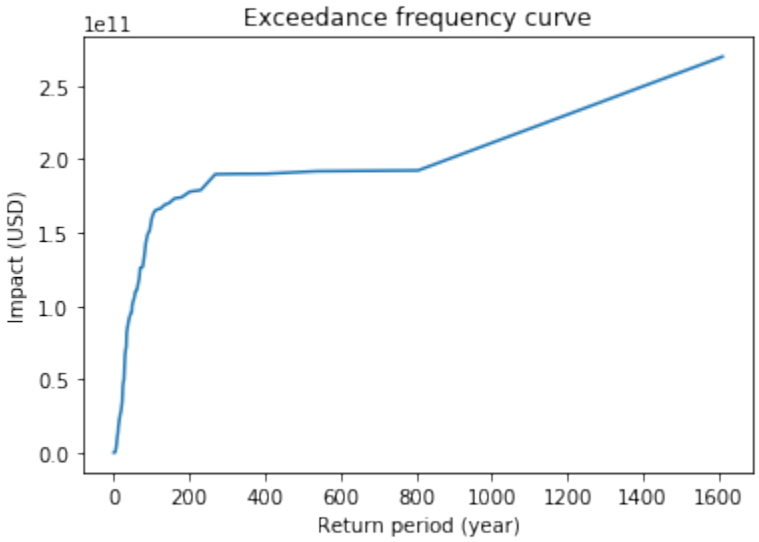
Let us compute the impact of tropical cyclones over the exposures selected in Florida.

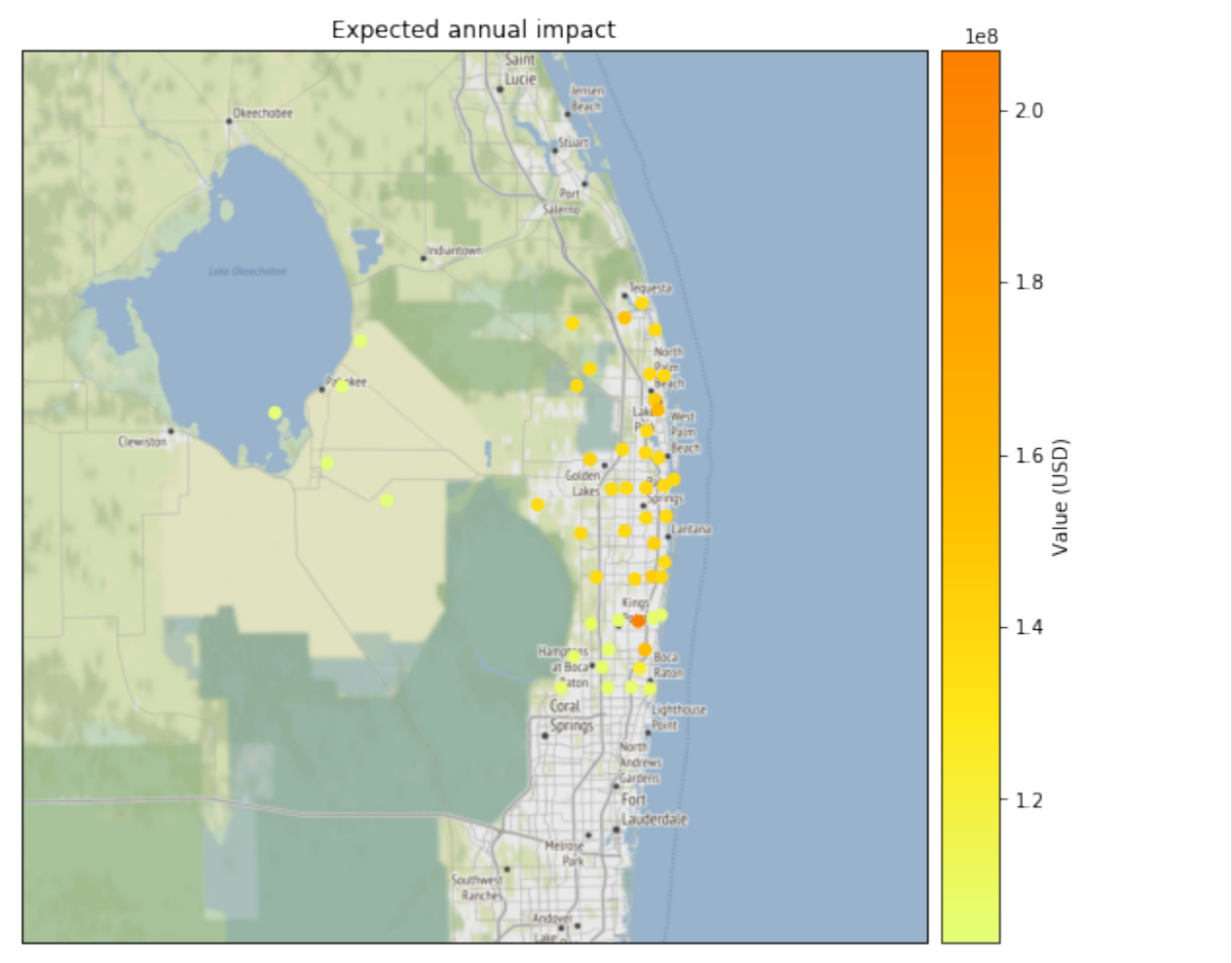
The configurable parameter `MAX_SIZE` controls the maximum matrix size contained in a chunk. You can decrease its value if you are having memory issues when using the `Impact`'s `calc` method. A high value will make the computation fast, but increase the memory use. The configuration file is located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

[8]:
```
from climada.engine import Impact

imp_fl = Impact()
imp_fl.calc(ent_fl.exposures, ent_fl.impact_funcs, tc_fl)

freq_curve_fl = imp_fl.calc_freq_curve() # impact exceedence frequency curve
freq_curve_fl.plot();

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

imp_fl.plot_basemap_eai_exposure(buffer=50000.0); # average annual impact at each␣
↪exposure
```

```
2020-09-16 14:52:09,044 - climada.entity.exposures.base - INFO - Matching 50␣
↪exposures with 100 centroids.
2020-09-16 14:52:09,047 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
Expected average annual impact: 6.512e+09 USD
2020-09-16 14:52:09,092 - climada.util.coordinates - INFO - Setting geometry points.
2020-09-16 14:52:09,098 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2020-09-16 14:52:10,856 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
/home/tovogt/code/climada_python/climada/util/plot.py:314: UserWarning: Tight layout␣
↪not applied. The left and right margins cannot be made large enough to accommodate␣
↪all axes decorations.
  fig.tight_layout()
/home/tovogt/.local/share/miniconda3/envs/tc_env/lib/python3.7/site-packages/
↪contextily/tile.py:199: FutureWarning: The url format using 'tileX', 'tileY',␣
↪'tileZ' as placeholders is deprecated. Please use '{x}', '{y}', '{z}' instead.
  FutureWarning,
```





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.

[9]:
```
import os
from climada.util import save, load
save('impact_florida.p', imp_fl)

# Later, the data can be read as follows:
abs_path = os.path.join(os.getcwd(), 'results/impact_florida.p') # absolute path
data = load(abs_path)
print('Data read:', type(data))
```

```
2020-09-16 14:52:11,205 - climada.util.save - INFO - Written file /home/tovogt/code/
↪climada_python/doc/tutorial/results/impact_florida.p
Data read: <class 'climada.engine.impact.Impact'>
```

`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 class doc to get more information about these functions.

See Impact to learn more about impact calculations.

3.3 Adaptation options appraisal

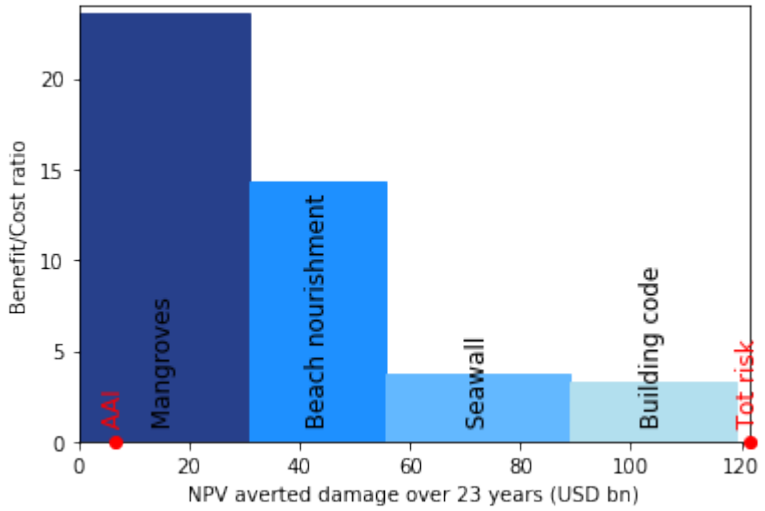
The adaptation measures defined before can be valued by estimating its cost-benefit ratio. This is done in the class `CostBenefit`.

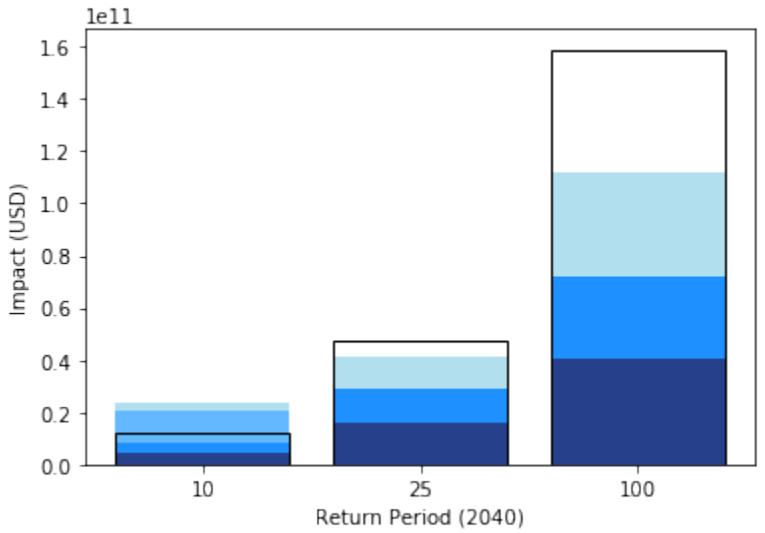
Let us suppose that the socioeconomic and climatoligical conditions remain the same in 2040. We then compute the cost and benefit of every adaptation measure as follows:

[10]:
```
from climada.engine import CostBenefit

cost_ben = CostBenefit()
cost_ben.calc(tc_fl, ent_fl, 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() # plot averted damage of each measure for every return␣
↪period
```

```
2020-09-16 14:52:11,217 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,219 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,244 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,245 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,278 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,279 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,312 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,313 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,365 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,366 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,394 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,395 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,426 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2040.

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------  ----------  ------------  ---------
Mangroves                  1.31177             31.0058        23.6367
Beach nourishment          1.728               24.6898        14.2881
Seawall                    8.87878             33.133          3.7317
Building code              9.2                 30.3762         3.30177

-------------  -----  -----
Total climate risk:   121.505   (USD bn)
Average annual risk:    6.5122  (USD bn)
Residual risk:          2.3001  (USD bn)
-------------  -----  -----
Net Present Values
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1acd508cf8>
```





Let us now assume that the exposure evolves according to `ENT_DEMO_FUTURE` in 2040 and that the intensity of the hazards increase uniformly due to climate change.

[11]:
```
import copy
from climada.util.constants import ENT_DEMO_FUTURE

# future conditions
ent_future = Entity()
ent_future.read_excel(ENT_DEMO_FUTURE)
ent_future.check()
ent_future.exposures.ref_year = 2040

haz_future = copy.deepcopy(tc_fl)
haz_future.intensity.data += 15  # increase uniformly the intensity

cost_ben = CostBenefit()
cost_ben.calc(tc_fl, ent_fl, haz_future, ent_future, save_imp=True)
```

(continues on next page)

(continued from previous page)

```
cost_ben.plot_cost_benefit() # plot cost benefit ratio and averted damage of every␣
↪exposure
cost_ben.plot_event_view() # plot averted damage of each measure for every return␣
↪period
ax = cost_ben.plot_waterfall(tc_fl, ent_fl, haz_future, ent_future) # plot expected␣
↪annual impact
ax.set_title('Expected Annual Impact in 2015 and 2040')
ax = cost_ben.plot_waterfall_accumulated(tc_fl, ent_fl, ent_future) # plot␣
↪accumulated impact from present to future
cost_ben.plot_arrow_averted(ax, accumulate=True, combine=True, disc_rates=ent_fl.disc_
↪rates) # plot total averted damages
```

```
2020-09-16 14:52:11,713 - climada.entity.exposures.base - INFO - crs set to default␣
↪value: {'init': 'epsg:4326', 'no_defs': True}
2020-09-16 14:52:11,714 - climada.entity.exposures.base - INFO - ref_year metadata␣
↪set to default value: 2018
2020-09-16 14:52:11,714 - climada.entity.exposures.base - INFO - value_unit metadata␣
↪set to default value: USD
2020-09-16 14:52:11,714 - climada.entity.exposures.base - INFO - meta metadata set to␣
↪default value: None
2020-09-16 14:52:11,715 - climada.entity.exposures.base - INFO - centr_ not set.
2020-09-16 14:52:11,715 - climada.entity.exposures.base - INFO - category_id not set.
2020-09-16 14:52:11,715 - climada.entity.exposures.base - INFO - region_id not set.
2020-09-16 14:52:11,715 - climada.entity.exposures.base - INFO - geometry not set.
2020-09-16 14:52:11,724 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,726 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,752 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,753 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,783 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,785 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,811 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,812 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,860 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,861 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,890 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,892 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,925 - climada.entity.exposures.base - INFO - Matching 50␣
↪exposures with 100 centroids.
2020-09-16 14:52:11,928 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,957 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:11,958 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:11,999 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
```

(continues on next page)

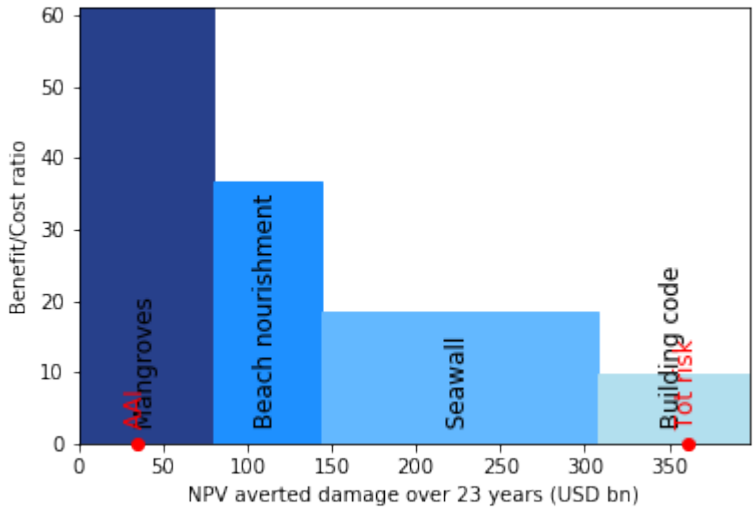
(continued from previous page)

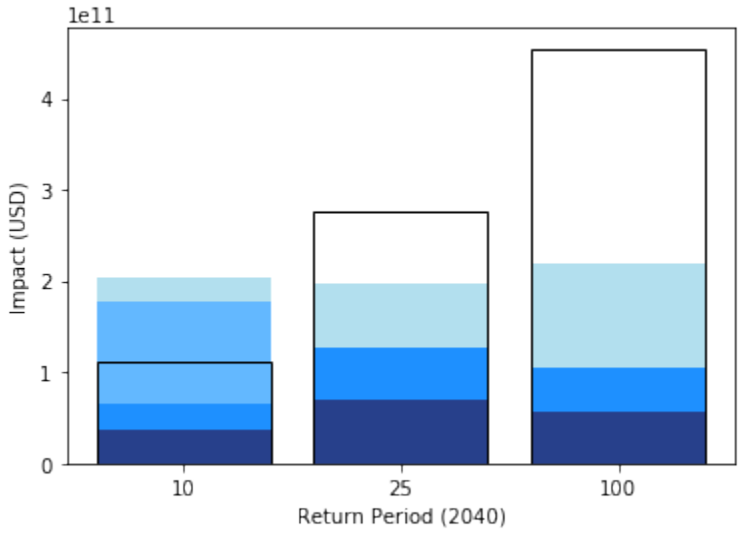
```
2020-09-16 14:52:12,001 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:12,036 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:12,037 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:12,094 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:12,096 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:12,127 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:12,128 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:12,163 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2040.

Measure               Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------  ----------  ------------  ---------
Mangroves                  1.31177            80.0097        60.9938
Beach nourishment          1.728              63.3336        36.6514
Seawall                    8.87878           164.132         18.4858
Building code              9.2                90.2786         9.81289

-------------  -----  -----
Total climate risk:   361.115   (USD bn)
Average annual risk:   34.3977  (USD bn)
Residual risk:        -36.6389  (USD bn)
-------------  -----  -----
Net Present Values
2020-09-16 14:52:12,201 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:12,202 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:12,233 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:12,234 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:12,271 - climada.engine.cost_benefit - INFO - Risk at 2018: 6.512e+09
2020-09-16 14:52:12,272 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:12,273 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:12,304 - climada.engine.cost_benefit - INFO - Risk with development␣
↪at 2040: 1.302e+10
2020-09-16 14:52:12,304 - climada.engine.cost_benefit - INFO - Risk with development␣
↪and climate change at 2040: 3.440e+10
2020-09-16 14:52:12,315 - climada.engine.cost_benefit - INFO - Current total risk at␣
↪2040: 1.215e+11
2020-09-16 14:52:12,315 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2020-09-16 14:52:12,316 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 14450 events.
2020-09-16 14:52:12,348 - climada.engine.cost_benefit - INFO - Total risk with␣
↪development at 2040: 1.775e+11
2020-09-16 14:52:12,349 - climada.engine.cost_benefit - INFO - Total risk with␣
↪development and climate change at 2040: 3.611e+11
```

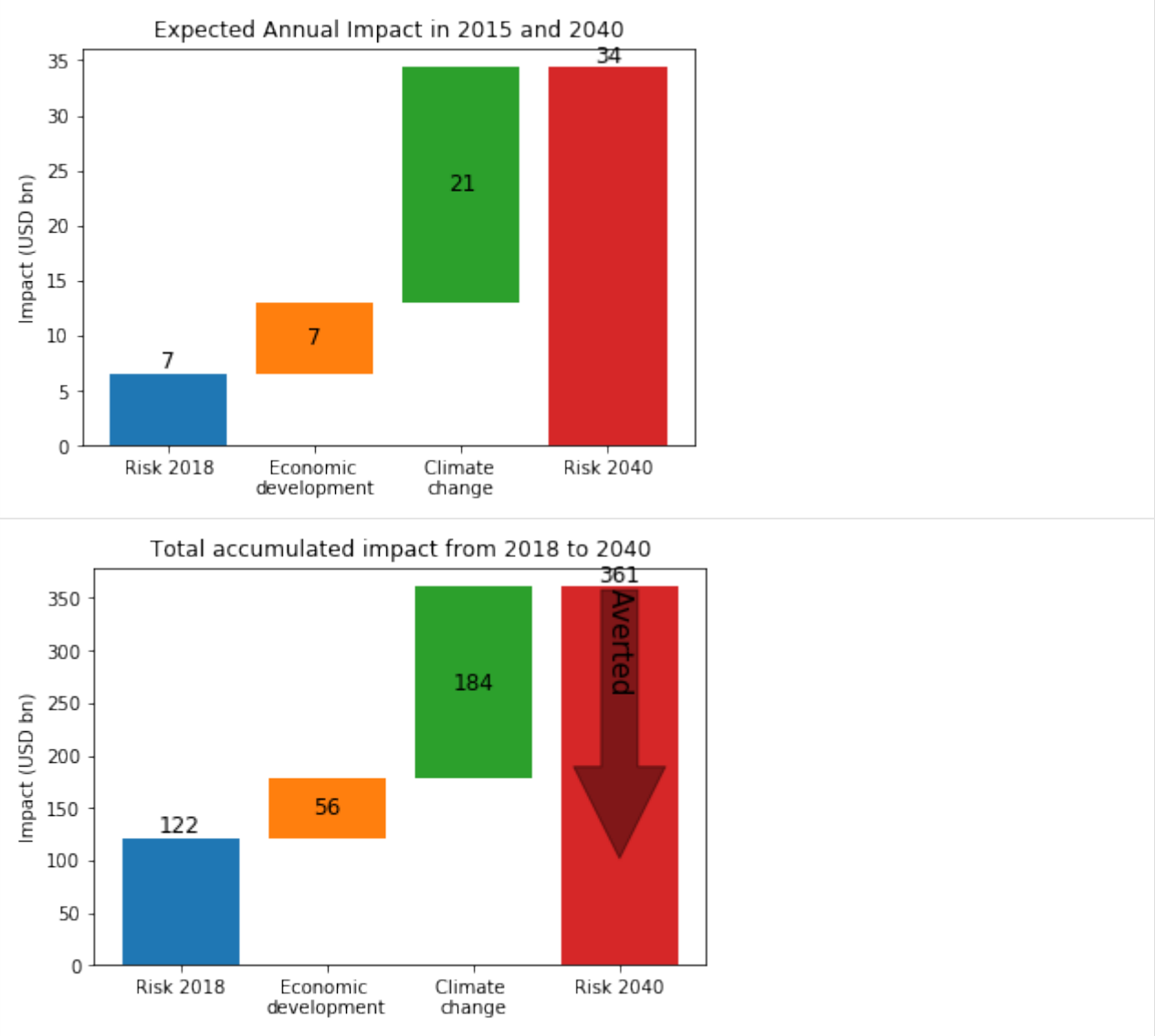
(continues on next page)

(continued from previous page)

```
2020-09-16 14:52:12,364 - climada.engine.cost_benefit - INFO - Combining measures␣
↪['Mangroves', 'Beach nourishment', 'Seawall', 'Building code']

Measure      Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
------  ----------  ------------  ---------
combine            21.1185             262.731         12.4408

-------------  -----  -----
Total climate risk:   361.115   (USD bn)
Average annual risk:   34.3977  (USD bn)
Residual risk:         98.3835  (USD bn)
-------------  -----  -----
Net Present Values
```







Check what happens when different parameters are changed, such as the `imp_time_depen` and `risk_func` in CostBenefit.calc() (and plot_waterfall(), plot_waterfall_accumulated())

3.4 Your case

1. Build an entity. It might be one from your previous runs in MATLAB. Make sure i'is saved in version > v7.3 if it's a MATLAB file. If it's not, you'll get an error message. Then, you can save it again in MATLAB like that: `save('file_name.mat','variable_name', '-v7.3')`
2. Build a hazard. It might also come from a previous run in MATLAB. This file might already contain the centroids. If not, define the centroids as well and use them in your calculations.
3. Compute the impact.
4. Visualization. Plot:

- the damage functions for the hazard

- the entity values map
- the strongest event intensity
- the maximum hazard intensity of all the events in Zürich (47.38, 8.54)
- the impact exceedence frequency curve

[12]:
```
# Put your code here
```

[13]:
```
# SOLUTION: example: winter storms in europe
from climada.util import DATA_DIR
import pandas as pd
from climada.hazard import Hazard
from climada.entity import Exposures, ImpactFuncSet
from climada.engine import Impact

# Put any absoulte path for your files or set up the configuration variable
↪"repository"
FILE_HAZARD = DATA_DIR + '/demo/WS_ERA40_sample.mat'
FILE_ENTITY = DATA_DIR + '/demo/WS_Europe.xls'

# Define hazard type
HAZ_TYPE = 'WS'

# 1. Entity: we only need impact functions and exposures to compute the impact
# Exposures
exp_ws_eu = pd.read_excel(FILE_ENTITY)
exp_ws_eu = Exposures(exp_ws_eu)
exp_ws_eu.check()

# Impact functions
impf_ws_eu = ImpactFuncSet()
impf_ws_eu.read_excel(FILE_ENTITY, 'Impact functions for winter storms in EU.')

# 2. Hazard
haz_ws_eu = Hazard(HAZ_TYPE)
haz_ws_eu.read_mat(FILE_HAZARD, 'WS EU ERA 40')

# 3. Impact
imp_ws_eu = Impact()
imp_ws_eu.calc(exp_ws_eu, impf_ws_eu, haz_ws_eu)

# 4.
# the damage functions for the hazard
impf_ws_eu.plot()

# the exposures values map
exp_ws_eu.plot_hexbin(pop_name=False)

# the strongest event
haz_ws_eu.plot_intensity(-1) # might be better to use an other earth projection?

# the impact exceedence frequency curve
```

(continues on next page)

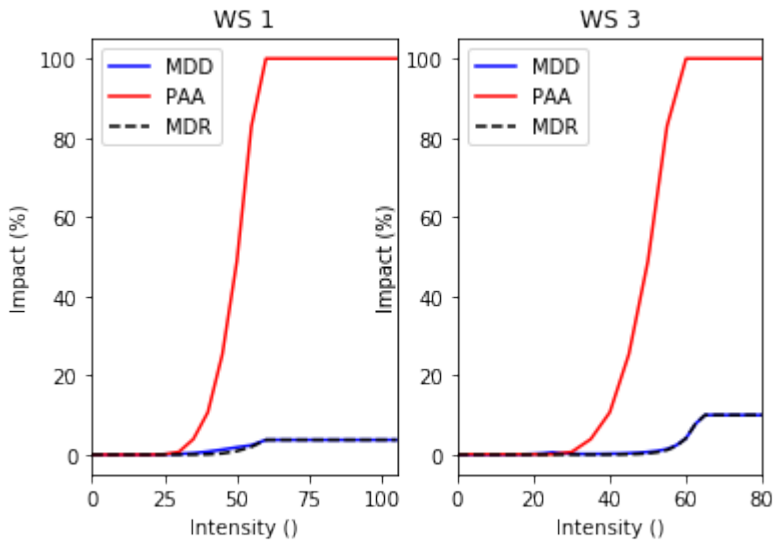
(continued from previous page)

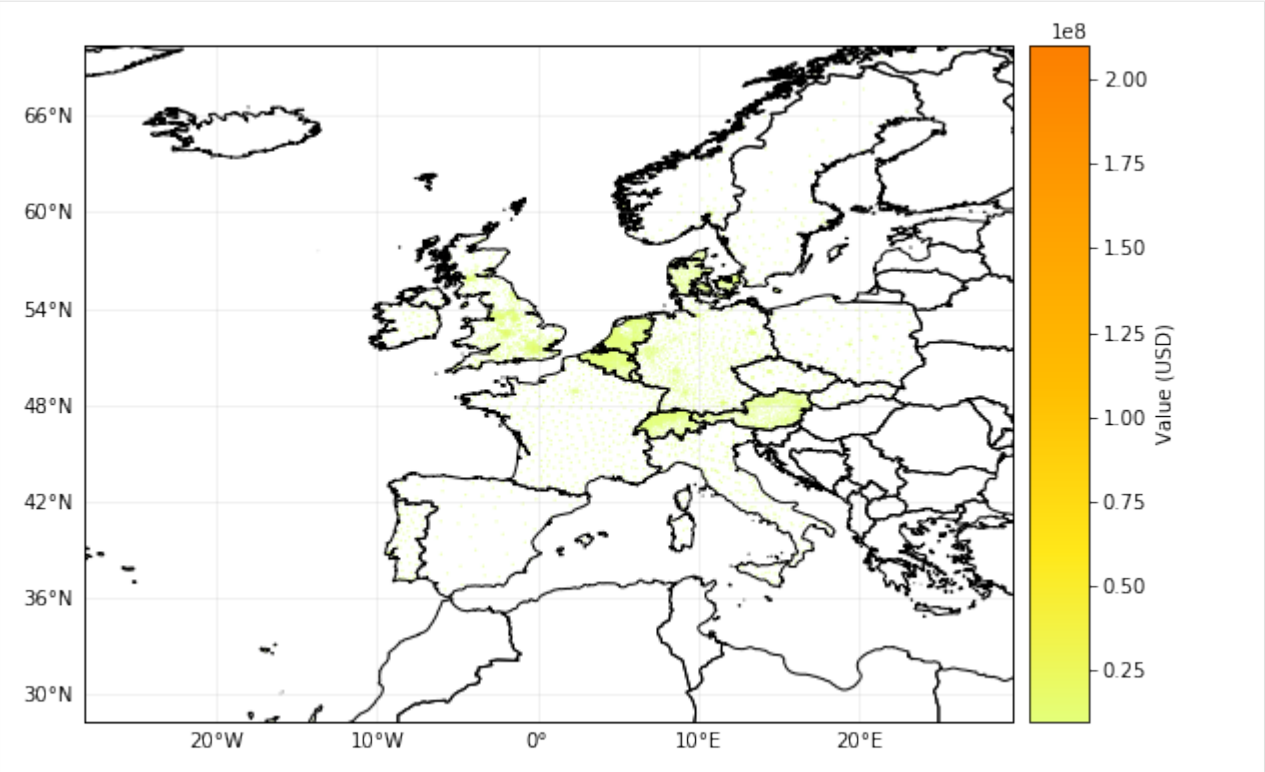
```
imp_exc_curve = imp_ws_eu.calc_freq_curve()
imp_exc_curve.plot()
```

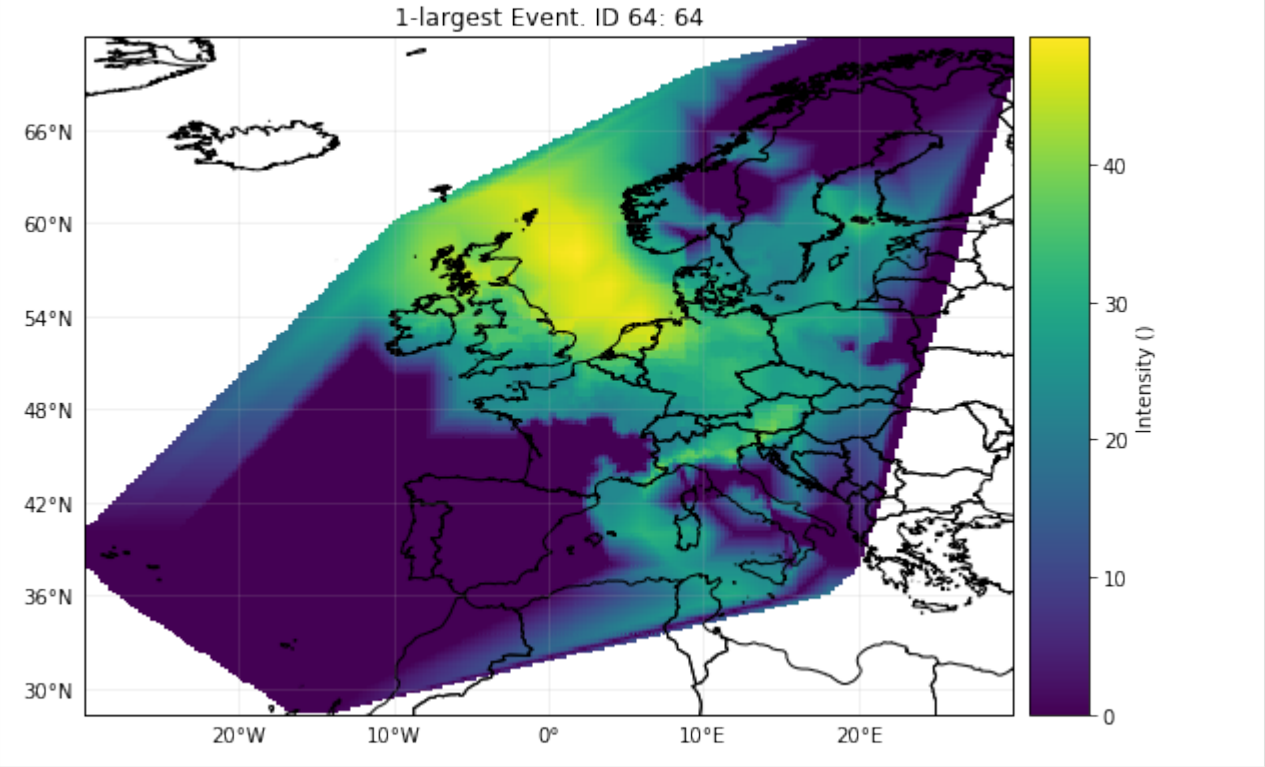
```
2020-09-16 14:52:12,842 - climada.entity.exposures.base - INFO - crs set to default␣
↪value: {'init': 'epsg:4326', 'no_defs': True}
2020-09-16 14:52:12,843 - climada.entity.exposures.base - INFO - tag metadata set to␣
↪default value:  File:
 Description:
2020-09-16 14:52:12,843 - climada.entity.exposures.base - INFO - ref_year metadata␣
↪set to default value: 2018
2020-09-16 14:52:12,844 - climada.entity.exposures.base - INFO - value_unit metadata␣
↪set to default value: USD
2020-09-16 14:52:12,844 - climada.entity.exposures.base - INFO - meta metadata set to␣
↪default value: None
2020-09-16 14:52:12,844 - climada.entity.exposures.base - INFO - centr_ not set.
2020-09-16 14:52:12,845 - climada.entity.exposures.base - INFO - category_id not set.
2020-09-16 14:52:12,845 - climada.entity.exposures.base - INFO - region_id not set.
2020-09-16 14:52:12,846 - climada.entity.exposures.base - INFO - geometry not set.
2020-09-16 14:52:12,932 - climada.hazard.base - INFO - Reading /home/tovogt/code/
↪climada_python/data/demo/WS_ERA40_sample.mat
2020-09-16 14:52:12,948 - climada.hazard.centroids.centr - INFO - Reading /home/
↪tovogt/code/climada_python/data/demo/WS_ERA40_sample.mat
2020-09-16 14:52:12,965 - climada.entity.exposures.base - INFO - Matching 6186␣
↪exposures with 6331 centroids.
2020-09-16 14:52:13,496 - climada.engine.impact - INFO - Calculating damage for 6186␣
↪assets (>0) and 100 events.
```

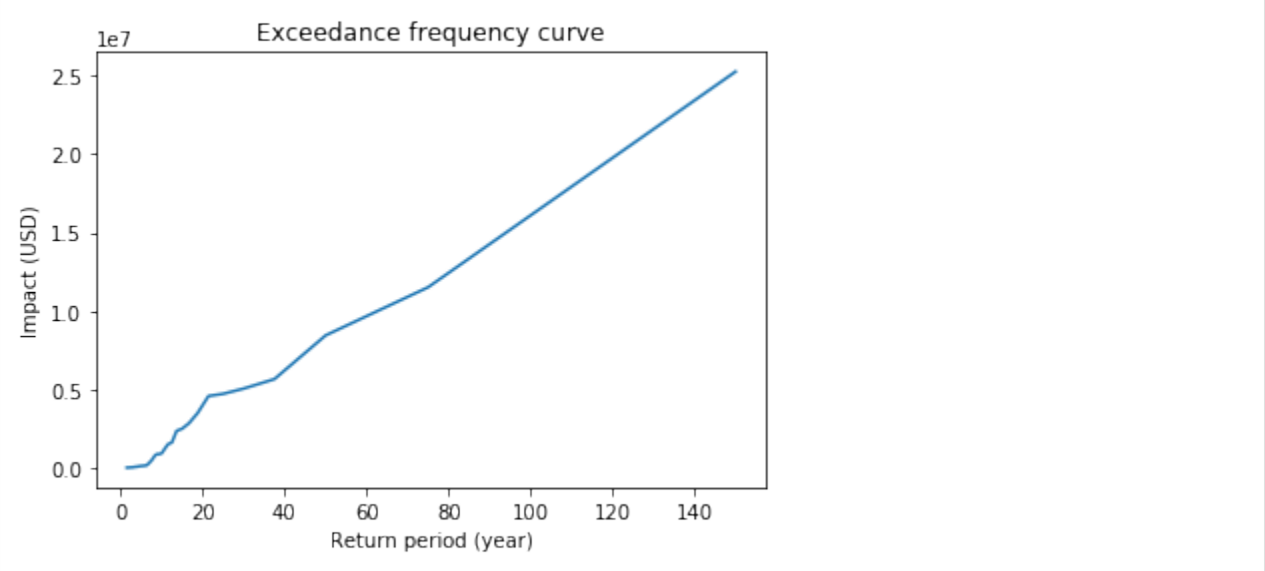
```
/home/tovogt/code/climada_python/climada/util/plot.py:314: UserWarning: Tight layout␣
↪not applied. The left and right margins cannot be made large enough to accommodate␣
↪all axes decorations.
  fig.tight_layout()
```

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1acd5d1828>
```




1e8
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Value (USD)
66°N
60°N
54°N
48°N
42°N
36°N
30°N
20°W
10°W
0°
10°E
20°E


1-largest Event. ID 64: 64
40
30
20
10
0
Intensity ()
66°N
60°N
54°N
48°N
42°N
36°N
30°N
20°W
10°W
0°
10°E
20°E


Exceedance frequency curve
1e7
2.5
2.0
1.5
1.0
0.5
0.0
Impact (USD)
0
20
40
60
80
100
120
140
Return period (year)

CHAPTER

FOUR

DATA DEPENDENCIES

4.1 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.

4.2 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 availabilty 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	•	FIRMS	BushFire	> v1.2.5 | climada_hazard_BushFire.ipynb	
OK	Gridded Population of the World (GPW)	v4.11	GPW v4.11	LitPop	> v1.2.3	climada_entity_LitPop.ipynb
FAILED	Gridded Population of the World (GPW)	v4.10	GPW v4.10	LitPop	>= v1.2.0	climada_entity_LitPop.ipynb

CHAPTER

FIVE

CONFIGURATION OPTIONS

CLIMADA searches for a local configuration file located in the current working directory. A static default configuration file is supplied by the package and used as fallback. The local configuration file needs to be called `climada.conf`. All other files will be ignored.

The climada configuration file is a JSON file and consists of the following values:

- `config`
- `local_data`
- `global`
- `trop_cyclone`

A minimal configuration file looks something like this:

```
{
    "config":
    {
        "env_name": "climada_env"
    },

    "local_data":
    {
        "save_dir": "./results/"
    },

    "global":
    {
        "log_level": "INFO",
        "max_matrix_size": 1.0e8
    },

    "trop_cyclone":
    {
        "random_seed": 54
    }
}
```

5.1 config

Configuration parameters related with configuration settings such as paths.

Option	Description	Default
`env_name`	Name given to CLIMADA's virtual environement. Used for checks of paths of libraries.	"climada_env"

5.2 local_data

Configuration parameters related to local data location.

Option	Description	Default
`save_dir`	Folder were the variables are saved through the `save` command when no absolute path provided.	"./results"

5.3 global

Configuration parameters with global scope within climada's code.

Option	Description	Default
`log_level`	Minimum log level showed by logging: DEBUG, INFO, WARNING, ERROR or CRITICAL.	"INFO"
`max_matrix_size`	Maximum matrix size that can be used. Set a lower value if memory issues.	1.0E8

5.4 trop_cyclone

Configuration parameters related to tropical cyclones.

Option	Description	Default
`random_seed`	Seed used for the stochastic tracks generation.	54

CHAPTER

SIX

CONTRIBUTING

Contributions are very welcome! Please follow these steps:

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/climada_python.git
   ```

   For more information on the Git flow approach to development see *Installation*.

2. **Install the packages** in `climada_python/requirements/env_climada.yml` and `climada_python/requirements/env_developer.yml` (see *Installation*). You might need to install additional environments contained in `climada_python/requirements` when using specific functionalities.

3. 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
   ```

   See CLIMADA-python's branching policies in *Git Flow*.

   General information about Git branches.

4. Follow the coding_conventions. Write small readable methods, classes and functions. Make well commented and clean **commits** to the repository:

   ```
   # get the latest data from the remote repository and update your branch
   git pull

   # see your locally modified files
   git status
   ```

(continues on next page)

(continued from previous page)

```
# 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"
```

Usually you will want a longer commit message than the one-line message above. In this case `git commit` will open your terminal's default text editor for a more detailed description. You can also create your commits interactively through your IDE's version control GUI (Spyder/PyCharm/etc).

5. Make unit and integration **tests** on your code, preferably during development:
   - Unit tests are located in the `test` folder located in same folder as the corresponding module. Unit tests should test all methods and functions using fake data if necessary. The whole test suite should run in less than 20 sec. They are all executed after each push in Jenkins.
   - Integration tests are located in `climada/test/`. They test end-to-end methods and functions. Their execution time can be of minutes. They are executed once a day in Jenkins.
6. Make sure your changes are not introducing new test failures.

   Run unit and integration tests:

   ```
   make unit_test
   make integ_test
   ```

   Compare the result to the results before the change. Current test failures are visible on Jenkins. Fix new test failures before you create a pull request or push to the develop branch of CLIMADA-project/climada_python. See *Continuous Integration* below.
7. Perform a **static code analysis** of your code using `pylint` with CLIMADA's configuration `.pylintrc`. Jenkins executes it after every push. To do it locally, you might use the Interface provided by *Spyder*. To do so, search first for *static code analysis* in *View* and then *Panes*.
8. Add new **data dependencies** used in *Data dependencies* and write a **tutorial** if a new class has been introduced (see *Tutorial*).
9. Add your name to the **AUTHORS** file.
10. Merge any updates to `develop` into your branch.

    There may have been changes to the remote `develop` branch since you created your branch. You can deal with potential conflicts by updating and merging `develop` into your branch:

    ```
    git checkout develop
    git pull
    git checkout feature/feature_name
    git merge develop
    ```

    Then resolve any conflicts. In the case of more complex conflicts, you may want to speak with others who worked on the same code.
11. **Push** the branch to GitHub.

    To push your branch `feature_branch_name` for the first time call:

    ```
    git push -u origin feature/feature_branch_name
    ```

    or, if you're updating a branch that's already on GitHub:

```
git push
```

Only push small bugfixes and comments directly to `develop` - most new code should be pushed as a feature branch, which can then be reviewed with a pull request. Only emergency hotfixes are pushed to `master`.

12. Create a pull request.

    When the branch is ready, create a new **pull request** from the feature branch. About pull requests.

    To do this,

    - On the CLIMADA GitHub page, navigate to your feature branch. Above the list of files is a summary of the branch 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 a 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, and should give a high-level summary of the changes, specific points you want the reviewers' input on, and possibly explanations for decisions you've made.
    - Assign reviewers using the right hand sidebar on the page. Tag anyone who might be interested in reading the code. You should have found someone who is happy to read the whole request and sign it off (this person could also be added to 'Assignees'). A list of potential reviewers can be found in the WIKI.
    - Contact reviewers. GitHub's settings mean that they may not be alerted automatically, so send them a message.

13. Review and merge the pull request.

    For big pull requests, stay in touch with reviewers. When everyone has had the chance to make comments and suggestions, and at least one person has read and approved the whole request, it's ready to be merged.

    If `develop` has been updated during the review process, it may be necessary to resolve merge conflicts again.

    Merging the pull request is done through the GitHub site. Once it's merged you can delete the feature branch and update your local copy of `develop` with `git pull`.

6.1 Update CLIMADA's environment

Remember to regularly update your code as well as climada's environment. You might use the following commands to update the environments:

```
cd climada_python
git pull
source activate climada_env
conda env update --file requirements/env_climada.yml
conda env update --file requirements/env_developer.yml
```

If any problem occurs during this process, consider reinstalling everything from scratch following the :doc:install instructions. You can find more information about virtual environments with conda here.

6.2 Continuous Integration

The results from the Jenkins server are to be taken seriously. Please run unit tests locally on the whole project, by calling *make unit_test* and if possible remotely on Jenkins in a feature branch.

Before pushing to the develop branch they should run without errors or (novel) failures. After pushing, check the CI results on Jenkins, if the commit causes an error there, revert it immediately. If the commit merely introduces novel failures, fix them within 3 days, or revert the commit.

Similar rules apply for the Pylint results on the deveolp branch. Novel high priority warnings are not acceptable on the develop branch. Novel medium priority warnings should be fixed within 3 days.

6.2.1 Tolerance overview

Branch	Unittest		Linter		
	Error	Failure	High	Medium	Low
Master	x	x	x	(x)	-
Develop	x	3 days	x	3 days	-
Feature	(x)	-	-	-	-

x indicates "no tolerance", meaning that any code changes producing such offences should be fixed *before* pushing them to the respective branch.

6.3 Issues

Issues are the main platform for discussing matters. Use them extensively! Each issue should have one categoric label:

- bug
- enhancement
- question
- incident

and optionally others. When closing issues they should get another label for the closing reason:

- fixed
- wontfix
- duplicate
- invalid

(Despite their names, *fixed* and *wontfix* are applicable for questions and enhancements as well.)

6.4 Regular Releases

Regular releases are planned on a quarterly base. Upcoming releases are listed in the WIKI.

CHAPTER

SEVEN

GIT FLOW

In general, our policy and naming of branches follow broadly the conventions of git flow.

7.1 Philosophy

We use a *merge only* philosophy in all shared branches. This is safer when working with people with various levels of git-skills.

We use rather long-lived feature branches, as we are scientist and not software engineers. The creator of a feature branch is the owner/responsible for its content. You can use the workflow you prefer inside of your feature branch. It must be regularly merged into the develop branch, and at this point the branch owner must organize a code review. This allows for a smooth development of both the code and the science, without compromising the code legacy.

Scientific publication should, once accepted, be made into a minimal working example and pushed onto the Climada_papers repository.

Do not forget to update the Jenkins test and the CLIMADA tutorial.

7.2 Release cycle

When a new release is made, everything in the develop branch is merged into the master branch.

7.3 Fork or clone?

Core developers should clone the project.

External developers can fork the project. If you want to become a core developer, please contact us.

7.4 Branches

The branching system is adapted for the scientific setting from

- The `master` branch is used for the quarterly releases of stable code.
- The `develop` branch is used to gather working/tested code in between releases.
- Development is done in feature branches.

- Feature branches are used for any CLIMADA development. Note that a feature branch should not live forever and should contribute to the `develop` branch. Hence, try to merge your working code for each release, or every second release at least. Ideally, a branch is cleaned after max. one year and then archived.
- Small fixes, bugs, code updates, improvements etc. are done as feature branches and should use a clear prefix such as 'Fix-'. These branches are thought of to be short lived, but still require a review! After the review, these branches are usually deleted.
- Code for a scientific project/paper (i.e. CLIMADA application, not development) is not pushed to this repository. A minimal working example should be pushed to the climada_papers repository.

7.5 What files to commit?

- Git is for the code, *not* the data. The only exceptions are small data samples for the unit tests. Small means (<1mb). A very very strong reason must be given to commit larger files. A more systematic way to handle data will be developed soon.

7.6 .gitignore

See herefor details on how to use the .gitignore file.

- If your script (not a paper script, but a core CLIMADA script) produces files, add these to the gitignore file so that they are not committed. Add .gitignore to a commit. Do only add your file.
- Remember to remove a file from the gitignore if it is not produced by the code anymore.

7.7 Don'ts

- Do not rebase on the `develop`/`master` branches, or on any branch that has already been pushed to GitHub.
- Do not use fast-forwarding on the remote branches.

7.8 Creating feature branches

Before/After starting a new scientific feature branch, present a 2-3 mins plan at the bi-weekly developers meeting.

Naming convention: feature/my_feature_name

For small features (fixes, improvements,...), use a clear prefix such as 'Fix-'.

Naming convention: feature/Fix-my_feature_name

CHAPTER

EIGHT

HOW TO USE GIT FOR CLIMADA

8.1 How to use Git?

Please check the Git book tutorial to get a basic understanding of git.

Recommended reading (to begin with):

- Chapter 1 Getting started
- Chapter 2 Git Basics
- Chapter 3 Git Branching,
- Chapter 6 GitHub

Also checkout this cheatsheet for git commands.

8.2 GUI or command line

- The probably most complete way to use git is through the command line. However, this is not very visual, in particular at the beginning. Consider using a GUI program such as "git desktop" or "Gitkraken". Your python IDE is also likely to have a visual git interface.
- Consider using an external merging and conflict resolution tool

8.3 Commit messages

Basic syntax guidelines taken from here (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.

8.4 Git commands for CLIMADA

Below should be all the commands you need to get started for working on a feature branch (assuming it already exists). More features are available in git, and feel free to use them (e.g. stashing or cherry picking). However, you should follow the dont's (do not rebase *on* the develop branch, and do not fast-foward on remote branches).

A) Regular / daily commits locally

0. `git fetch --all` (make your local git know the changes that happened on the repository)
1. `git checkout feature/feature_name` (be sure to be on your branch)
2. `git status`
3. `git add file1`
4. `git commit -m "Remove function xyz from feature.py"`
5. `git status` (verify that there are no more tracked files to be committed)

B) Push to remote branch (at least once/week, ideally daily)

1. `git fetch --all`
2. `git checkout feature/feature_name` (be sure to be on your branch)
3. Make all commits according to A
4. `git status` (check whether your local branch is behind the remote)
5. `git pull --rebase` (resolve all conflicts if there are any)
6. `git push -u origin feature/feature_name` if this is the first time you're pushing to the remote repository. Or just `git push` if the branch already exists there.

C) Merge develop into your branch (regularly/when develop changes)

1. `git fetch --all`
2. Make all commit according to A
3. `git status` (verify that there are no tracked files that are uncommitted)
4. `git checkout develop`
5. `git pull --rebase`
6. `git checkout feature/feature_name`
7. `git merge --no-ff develop`
8. resolve all conflicts if there are any
9. `git push origin feature/feature_name` if this is the first time you're pushing to the remote repository. Or just `git push` if the branch already exists there.

D) Prepare to merge into develop (ideally before every release)

1. `git fetch --all`
2. `git checkout feature/feature_name`
3. `git status` (see how many commits the branch is behind the remote)

4. Make all commits according to A
5. Merge develop into your branch according to C
6. Push the branch to GitHub. If parts of the feature are incomplete and not everything is ready to go into `develop`, create a new branch `feature/feature_name-release` with
    - `git checkout feature/feature_name-release`
    - Clean the code so that only changes to be pushed remain
    - Check that the code on the new branch passes unit and integration testing.
    - Commit all changes according to A)
    - `git push -u origin feature/feature_name-release`
7. Make a pull request
8. Find someone to do a code review on `feature/feature\_name-release`. Implement the code review suggestions (once done, redo steps 4 - 6))

CHAPTER

NINE

SOFTWARE DOCUMENTATION

Documents functions, classes and methods:

9.1 Software documentation per package

9.1.1 climada.engine package

climada.engine.calibration_opt module

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. Key: measure name ('no measure' used for case without measure), Value: np.array

Type dict

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__()
Initilization

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** (*Hazard*) – hazard
- **entity** (*Entity*) – entity
- **haz_future** (*Hazard, optional*) – hazard in the future (future year provided at ent_future)
- **ent_future** (*Entity, optional*) – entity in the future
- **future_year** (*int, optional*) – future year to consider if no ent_future 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: None.
- **save_imp** (*bool, optional*) – True if Impact of each measure is saved. Default: 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: None.
- **risk_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).

Returns 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.

Paramters: 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): factor to which to multiply the insurance layer to compute its cost. Default: 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: 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 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 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** (*Hazard*) – hazard
- **entity** (*Entity*) – entity
- **haz_future** (*Hazard*) – hazard in the future (future year provided at ent_future)
- **ent_future** (*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 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** (*Hazard*) – hazard
- **entity** (*Entity*) – entity
- **ent_future** (*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. 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 matplotlib.axes._subplots.AxesSubplot

climada.engine.cost_benefit.**risk_aai_agg**(*impact*)

Risk measurement as average annual impact aggregated.

Parameters **impact** (*Impact*) – an Impact instance

Returns float

climada.engine.cost_benefit.**risk_rp_100**(*impact*)

Risk measurement as exceedance impact at 100 years return period.

Parameters **impact** (*Impact*) – an Impact instance

Returns float

climada.engine.cost_benefit.**risk_rp_250**(*impact*)

Risk measurement as exceedance impact at 250 years return period.

Parameters **impact** (*Impact*) – an Impact instance

Returns float

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(), 'if_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__()
Initialize self. See help(type(self)) for accurate signature.

plot(*axis=None*, *log_frequency=False*, ***kwargs*)
Plot impact frequency curve.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **log_frequency** (*boolean*) – plot logarithmioc exceedance frequency on x-axis
- **kwargs** (*optional*) – arguments for plot matplotlib function, e.g. color='b'

Returns 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(), 'if_set': Tag(), 'haz': TagHazard()}

Type dict

event_id
id (>0) of each hazard event

Type np.array

event_name
name of each hazard event

Type list

date
date of events

Type np.array

coord_exp
exposures coordinates [lat, lon] (in degrees)

Type np.ndarray

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.arrray

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 ImpactFreqCurve

calc(*exposures*, *impact_funcs*, *hazard*, *save_mat=False*)
Compute impact of an hazard to exposures.

Parameters

- **exposures** (*Exposures*) – exposures
- **impact_funcs** (*ImpactFuncSet*) – impact functions
- **hazard** (*Hazard*) – hazard
- **self_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 traaditional risk transfer over impact. Returns new impact with risk transfer applied and the insurance layer resulting Impact metrics.

Parameters

- **attachment** (*float*) – attachment (deductible)
- **cover** (*float*) – cover

Returns Impact, Impact

plot_hexbin_eai_exposure(*mask=None*, *ignore_zero=True*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, ***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 cartopy.mpl.geoaxes.GeoAxesSubplot

plot_scatter_eai_exposure(*mask=None*, *ignore_zero=True*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, ***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
- **kwargs** (*optional*) – arguments for hexbin matplotlib function

Returns 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*, ***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
- **kwargs** (*optional*) – arguments for imshow matplotlib function

Returns 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 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*, ***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

Returns 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 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=[]*)
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 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 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 log10(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 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*, *if_set*, *haz_list*, *file_name=''*, *writer=<matplotlib.animation.PillowWriter object>*, *imp_thresh=0*, *args_exp={}*, *args_imp={}*)

Computes and generates video of accumulated impact per input events over exposure.

Parameters

- **exp** (*Exposures*) – exposures instance, constant during all video
- **if_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 list(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:

input files (paths): intensity: sparse matrix with hazards as rows and grid points as cols, values only at location with impacts names: identifier for each hazard (i.e. IBtracID) (rows of the matrix) reg_id: ISO country ID of each grid point (cols of the matrix) date: start date of each hazard (rows of the matrix) emdat_data: pd.dataframe with EMdat data start: start date of events to be assigned 'yyyy-mm-dd' end: end date of events to be assigned 'yyyy-mm-dd' disaster_subtype: EMdat disaster subtype

Returns 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:

input files: intensity: sparse matrix with hazards as rows and grid points as cols, values only at location with impacts names: identifier for each hazard (i.e. IBtracID) (rows of the matrix) reg_id: ISO country ID of each grid point (cols of the matrix) date: start date of each hazard (rows of the matrix)

Returns 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** – start date of events to be assigned 'yyyy-mm-dd'
- **end** – end date of events to be assigned 'yyyy-mm-dd'
- **disaster_subtype** – EMdat disaster subtype

Returns pd.dataframe lookup

climada.engine.impact_data.**emdat_possible_hit**(*lookup*, *hit_countries*, *delta_t*)

relate EM disaster to hazard using hit countries and time

Parameters **input files** – lookup: pd.dataframe to relate EMdatID to hazard tracks: pd.dataframe with all hit countries per hazard delta_t: max time difference of start of EMdat event and hazard hit_countries: start: start date of events to be assigned end: end date of events to be assigned disaster_subtype: EMdat disaster subtype

Returns 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 list with all possible hits per EMdat ID

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 confidene into the procedure, hazards get only assigned if there is no other hazard occuring 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 of possible hits with smaller time horizon
- **possible_tracks_2** – list of possible hits with larger time horizon
- **level** – level of confidence

Returns pd.dataframe lookup with assigend tracks and possible hits

climada.engine.impact_data.**check_assigned_track**(*lookup*, *checkset*)

compare lookup with assigned tracks to a set with checked sets

Parameters

- **lookup** – pd.dataframe to relate EMdatID to hazard
- **checkset** – pd.dataframe with already checked hazards

Returns 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 or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV

Optional parameters:

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 accordung 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 DataFrame containing cleaned and filtered EM-DAT impact data

Return type df_data (pandas.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 or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV

Optional Parameters:

hazard (list or str): List of Disaster (sub-)type accordung 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 (tuple of integers or None): range of years to consider, i.e. (1950, 2000) default is None, i.e. consider all years

Returns

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

Return type countries_iso3a

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** (*str or DataFrame*) – Either string with full path to CSV-file or

pandas.DataFrame loaded from EM-DAT CSV

Optional parameters:

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 accordung 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): given EM-DAT data format version (i.e. year of download), changes naming of columns/variables (default: 2020)

reference_year (int): reference year of exposures. Impact is scaled proportional to GDP to the value of the reference year. No scaling for 0 (default)

imp_str (str): Column name of impact metric in EMDAT CSV, default = “Total Damages (‘000 US$)”

Returns

DataFrame with summed impact and scaled impact per year and country.

Return type out (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*) – Full path to EMDAT-file (CSV), i.e.: emdat_file_csv = os.path.join(SYSTEM_DIR, 'emdat_201810.csv')

Optional parameters:

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 accordung 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. No scaling for 0 (default)

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 (defaul: 2020)

Returns

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 out (pandas 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*) – Full path to EMDAT-file (CSV), i.e.: emdat_file_csv = os.path.join(SYSTEM_DIR, 'emdat_201810.csv')
- **hazard_type_climada** (*str*) – Hazard type CLIMADA abbreviation, i.e. 'TC' for tropical cyclone

Optional parameters:

hazard_type_emdat (list or str): List of Disaster (sub-)type accordung EMDAT terminology, e.g.: 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. If not given, it is deducted from hazard_type_climada

year_range (list with 2 integers): start and end year e.g. [1980, 2017] default: None –> take year range from EM-DAT file

countries (list of str): country ISO3-codes or names, e.g. ['JAM']. Set to None or ['all'] for all countries (default)

reference_year (int): reference year of exposures. 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, default = "Total Damages ('000 US$)"

Returns

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

Return type impact_instance (instance of climada.engine.Impact)

9.1.2 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

years

Type np.array

rates

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 DiscRates

append(*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** –

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 float

plot(*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_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*) – 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': {'disc': 'discount_rate', 'year': 'year'}, 'sheet_name': 'discount'}*)
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': {'disc': 'discount_rate', 'year': 'year'}, 'sheet_name': 'discount'}*)
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.exposures package

climada.entity.exposures.base module

class climada.entity.exposures.base.**Exposures**(**args*, ***kwargs*)
Bases: geopandas.geodataframe.GeoDataFrame

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

tag
metada - information about the source data

Type *Tag*

ref_year
metada - reference year

Type int

value_unit
metada - 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

if_
e.g. if_TC. impact functions id for hazard TC. There might be different hazards defined: if_TC, if_FL, ... If not provided, set to default '**if_**' 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 = ['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.

__init__(**args*, ***kwargs*)
Initialize. Copy attributes of input DataFrame.

check()
Check which variables are present

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 vector hazard. Nearest neighbor (NN) default
- **distance** (*str, optional*) – distance to use in vector hazard. Haversine default
- **threshold** (*float*) – distance threshold in km over which no neighbor will be found in vector hazard. Those are assigned with a -1. Default 100 km.

set_geometry_points(*scheduler=None*)
Set geometry attribute of GeoDataFrame with Points from latitude and longitude attributes.

Parameters **scheduler** (*str*) – 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: 0>*)
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
- **wdith** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp,.Resampling optional*) – resampling function used for reprojection to dst_crs

plot_scatter(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, ***kwargs*)
Plot exposures geometry's value sum scattered over Earth's map. The plot will we 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

- **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
- **kwargs** (*optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'. Default: 'Wistia'

Returns: cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hexbin(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, ***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 we 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
  - **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
  - **kwargs** (*optional*) – arguments for hexbin matplotlib function, e.g. reduce_C_function=np.average. Default: reduce_C_function=np.sum

Returns: 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*, ***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
- **kwargs** (*optional*) – arguments for imshow matplotlib function

Returns 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
- **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 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.

Optional Parameters:

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*)

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.

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

climada.entity.exposures.base.**add_sea** (*exposures*, *sea_res*)
Add sea to geometry's surroundings with given resolution. region_id set to -1 and other variables to 0.

Parameters **sea_res** (*tuple*) – (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

Returns Exposures

climada.entity.exposures.black_marble module

class climada.entity.exposures.black_marble.**BlackMarble** (**args*, ***kwargs*)
Bases: *climada.entity.exposures.base.Exposures*

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

set_countries (*countries*, *ref_year=2016*, *res_km=None*, *from_hr=None*, *admin_file='admin_0_countries'*, ***kwargs*)
Model countries using values at reference year. If GDP or income group not available for that year, consider the value of the closest available year.

Parameters

- **countries** (*list or dict*) – list of country names (admin0 or subunits) or dict with key = admin0 name and value = [admin1 names]
- **ref_year** (*int, optional*) – reference year. Default: 2016
- **res_km** (*float, optional*) – approx resolution in km. Default: nightlights resolution.
- **from_hr** (*bool, optional*) – force to use higher resolution image, independently of its year of acquisition.
- **admin_file** (*str*) – file name, admin_0_countries or admin_0_map_subunits

- **kwargs** (*optional*) – 'gdp' and 'inc_grp' dictionaries with keys the country ISO_alpha3 code. 'poly_val' polynomial transformation [1,x,x^2,...] to apply to nightlight (DEF_POLY_VAL used if not provided). If provided, these are used.

climada.entity.exposures.crop_production module

climada.entity.exposures.crop_production.**DEF_HAZ_TYPE = 'RC'**
Default hazard type used in impact functions id.

climada.entity.exposures.crop_production.**BBOX = array([-180, -85, 180, 85])**
"Default geographical bounding box of the total global agricultural land extent

climada.entity.exposures.crop_production.**YEARCHUNKS = {'1860soc': {'endyear': 1860, 'sta**
start and end years per senario as in ISIMIP-filenames

climada.entity.exposures.crop_production.**FN_STR_VAR = 'landuse-15crops_annual'**
fix filename part in input data

climada.entity.exposures.crop_production.**CROP_NAME = {'mai': {'fao': 'Maize', 'input':**
mapping of crop names

climada.entity.exposures.crop_production.**IRR_NAME = {'combined': {'name': 'combined'}, '**
mapping of irrigation parameter long names

climada.entity.exposures.crop_production.**YEARS_FAO = array([2000, 2018])**
Default years from FAO used (data file contains values for 1991-2018)

class climada.entity.exposures.crop_production.**CropProduction**(**args*, ***kwargs*)
Bases: *climada.entity.exposures.base.Exposures*

Defines agriculture exposures from ISIMIP input data and FAO crop data

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

crop
crop type f.i. 'mai', 'ric', 'whe', 'soy'

Type str

set_from_single_run(*input_dir='/home/docs/checkouts/readthedocs.org/user_builds/climada-python/checkouts/v1.5.1/data/ISIMIP_crop/Input/Exposure'*, *filename=None*, *hist_mean='/home/docs/checkouts/readthedocs.org/user_builds/climada-python/checkouts/v1.5.1/data/ISIMIP_crop/Output/Hist_mean'*, *bbox=array([- 180, - 85, 180, 85])*, *yearrange=array([1976, 2005])*, *cl_model=None*, *scenario='histsoc'*, *crop=None*, *irr=None*, *unit='USD'*, *fn_str_var='landuse-15crops_annual'*)
Wrapper to fill exposure from nc_dis file from ISIMIP :Parameters: * **input_dir** (*string*) – path to input data directory

- **filename** (*string*) – name of the landuse data file to use, e.g. "histsoc_landuse-15crops_annual_1861_2005.nc""
- **hist_mean** (*str or array*) – historic mean crop yield per centroid (or path)
- **bbox** (*list of four floats*) – bounding box: [lon min, lat min, lon max, lat max]
- **yearrange** (*int tuple*) – year range for exposure set f.i. (1990, 2010)
- **scenario** (*string*) – climate change and socio economic scenario f.i. '1860soc', 'histsoc', '2005soc', 'rcp26soc','rcp60soc','2100rcp26soc'

- **cl_model** (*string*) – abbrev. climate model (only for future projections of lu data) f.i. 'gfdl-esm2m', 'hadgem2-es', 'ipsl-cm5a-lr','miroc5'
- **crop** (*string*) – crop type f.i. 'mai', 'ric', 'whe', 'soy'
- **irr** (*string*) – irrigation type f.i 'firr' (full irrigation), 'noirr' (no irrigation) or 'combined'= firr+noirr
- **unit** (*string*) – unit of the exposure (per year) f.i 'USD' or 't'
- **fn_str_var** (*string*) – FileName STRing depending on VARiable and ISIMIP simuation round

Returns Exposure

set_mean_of_several_models(*input_dir='/home/docs/checkouts/readthedocs.org/user_builds/climada-python/checkouts/v1.5.1/data/ISIMIP_crop/Input/Exposure'*, *hist_mean='/home/docs/checkouts/readthedocs.org/user_builds/climada-python/checkouts/v1.5.1/data/ISIMIP_crop/Output/Hist_mean'*, *bbox=array([- 180, - 85, 180, 85])*, *yearrange=array([1976, 2005])*, *cl_model=None*, *scenario=None*, *crop=None*, *irr=None*, *unit='USD'*, *fn_str_var='landuse-15crops_annual'*)

Wrapper to fill exposure from several nc_dis files from ISIMIP

Optional Parameters: input_dir (string): path to input data directory historic mean (array): historic mean crop production per centroid bbox (list of four floats): bounding box:

[lon min, lat min, lon max, lat max]

yearrange (int tuple): year range for exposure set, f.i. (1976, 2005) scenario (string): climate change and socio economic scenario

f.i. 'histsoc' or 'rcp60soc'

cl_model (string): abbrev. climate model (only when landuse data is future projection)

f.i. 'gfdl-esm2m' etc.

crop (string): crop type f.i. 'mai', 'ric', 'whe', 'soy'

irr (string): irrigation type f.i 'rainfed', 'irrigated' or 'combined'= rainfed+irrigated

unit (string): unit of the exposure (per year) f.i 'USD' or 't'

fn_str_var (string): FileName STRing depending on VARiable and ISIMIP simuation round

Returns Exposure

set_to_usd(*input_dir='/home/docs/checkouts/readthedocs.org/user_builds/climada-python/checkouts/v1.5.1/data/ISIMIP_crop/Input/Exposure'*, *yearrange=array([2000, 2018])*)

Calculates the exposure in USD using country and year specific data published by the FAO.

Optional Parameters: input_dir (string): directory containing the input (FAO pricing) data yearrange (array): year range for prices, can also be set to a single year

Default is set to the arbitrary time range (2000, 2018) The data is available for the years 1991-2018

crop (str): crop type f.i. 'mai', 'ric', 'whe', 'soy'

Returns Exposure

aggregate_countries()
Aggregate exposure data by country.

Returns country codes (numerical ISO3) country_values (array): aggregated exposure value

Return type list_countries (list)

climada.entity.exposures.crop_production.**init_full_exposure_set**(*input_dir='/home/docs/checkouts/readt*
python/checkouts/v1.5.1/data/ISIMIP_c
file-
name=None,
hist_mean_dir='/home/docs/checkouts/
python/checkouts/v1.5.1/data/ISIMIP_c
output_dir='/home/docs/checkouts/rea
python/checkouts/v1.5.1/data/ISIMIP_c
bbox=array([-
180, - 85, 180,
85]), *year-*
range=array([1976,
2005]),
unit='t', *re-*
turn_data=False)

Generates CropProduction exposure sets for all files contained in the input directory and saves them as hdf5 files in the output directory

Parameters: input_dir (string): path to input data directory filename (string): if not specified differently, the file

'histsoc_landuse-15crops_annual_1861_2005.nc' will be used

output_dir (string): path to output data directory bbox (list of four floats): bounding box:

[lon min, lat min, lon max, lat max]

yearrange (array): year range for hazard set, f.i. (1976, 2005) unit (str): unit in which to return exposure (t/y or USD/y) return_data (boolean): returned output

False: returns list of filenames only, True: returns also list of data

Returns all filenames of saved initiated exposure files output_list (list): list containing all inisiated Exposure instances

Return type filename_list (list)

climada.entity.exposures.crop_production.**normalize_with_fao_cp**(*exp_firr*,
exp_noirr,
input_dir='/home/docs/checkouts/readthe
python/checkouts/v1.5.1/data/ISIMIP_cr
year-
range=array([2008,
2018]),
unit='t', *re-*
turn_data=True)

Normalize the given exposures countrywise with the mean crop production quantity documented by the FAO. Refer to the beginning of the script for guidance on where to download the needed FAO data.

Parameters

- **exp_firr** (*crop_production*) – exposure under full irrigation

- **exp_noirr** (*crop_production*) – exposure under no irrigation

Optional Parameters: input_dir (str): directory containing exposure input data yearrange (array): the mean crop production in this year range is used to normalize

the exposure data Default is set to the arbitrary time range (2008, 2018) The data is available for the years 1961-2018

unit (str): unit in which to return exposure (t/y or USD/y) return_data (boolean): returned output

True: returns country list, ratio = FAO/ISIMIP, normalized exposures, crop production per country as documented by the FAO and calculated by the ISIMIP dataset False: country list, ratio = FAO/ISIMIP, normalized exposures

Returns

List of country codes (numerical ISO3) ratio (list): List of ratio of FAO crop production and aggregated exposure

for each country

exp_firr_norm (CropProduction): Normalized CropProduction (full irrigation) exp_noirr_norm (CropProduction): Normalized CropProduction (no irrigation)

Return type country_list (list)

Returns (optional): fao_crop_production (list): FAO crop production value per country exp_tot_production(list): Exposure crop production value per country

(before normalization)

climada.entity.exposures.crop_production.**normalize_several_exp**(*input_dir='/home/docs/checkouts/readthe python/checkouts/v1.5.1/data/ISIMIP_cr output_dir='/home/docs/checkouts/readt python/checkouts/v1.5.1/data/ISIMIP_cr yearrange=array([2008, 2018])*, *unit='t'*, *return_data=True*)

Optional Parameters: input_dir (str): directory containing exposure input data output_dir (str): directory containing exposure datasets (output of exposure creation) yearrange (array): the mean crop production in this year range is used to normalize

the exposure data (default 2008-2018)

unit (str): unit in which to return exposure (t/y or USD/y) return_data (boolean): returned output

True: lists containing data for each exposure file. Lists: crops, country list, ratio = FAO/ISIMIP, normalized exposures, crop production per country as documented by the FAO and calculated by the ISIMIP dataset False: lists containing data for each exposure file. Lists: crops, country list, ratio = FAO/ISIMIP, normalized exposures

Returns

List of crops country_list (list): List of country codes (numerical ISO3) ratio (list): List of ratio of FAO crop production and aggregated exposure

for each country

exp_firr_norm (list): List of normalized CropProduction Exposures (full irrigation)
exp_noirr_norm (list): List of normalize CropProduction Exposures (no irrigation)

Return type crop_list (list)

Returns (optional): fao_crop_production (list): FAO crop production value per country
exp_tot_production(list): Exposure crop production value per country

(before normalization)

climada.entity.exposures.crop_production.**semilogplot_ratio**(*crop*, *countries*, *ratio*, *output_dir='/home/docs/checkouts/readthedocs python/checkouts/v1.5.1/data/ISIMIP_crop/Ou save=True*)

Plot ratio = FAO/ISIMIP against country codes.

Parameters

- **crop** (*str*) – crop to plot
- **countries** (*list*) – country codes of countries to plot
- **ratio** (*array*) – ratio = FAO/ISIMIP crop production data of countries to plot
- **output_dir** (*str*) – directory to save figure

Optional Parameters: save (boolean): True saves figure, else figure is not saved.

Returns fig (plt figure handle) axes (plot axes handle)

climada.entity.exposures.gdp_asset module

class climada.entity.exposures.gdp_asset.**GDP2Asset**(**args*, ***kwargs*)

Bases: *climada.entity.exposures.base.Exposures*

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

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

Parameters

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

climada.entity.exposures.gpw_import module

climada.entity.exposures.gpw_import.**check_bounding_box**(*coord_list*)

Check if a bounding box is valid. :Parameters: **coord_list** (*4x1 array*) – bounding box to be checked.

OUTPUT: isCorrectType (boolean): True if bounding box is valid, false otehrwise

climada.entity.exposures.gpw_import.**get_box_gpw**(***parameters*)

Reads data from GPW GeoTiff file and cuts out the data along a chosen bounding box.

Parameters

- **gpw_path** (*str*) – Absolute path where files are stored. Default: SYSTEM_DIR
- **resolution** (*int*) – The resolution in arcsec in which the data output is created.
- **country_cut_mode** (*int*) – Defines how the country is cut out: If 0, the country is only cut out with a bounding box. If 1, the country is cut out along it's borders Default: 0. #TODO: Unimplemented
- **cut_bbox** (*array-like, shape (1,4)*) – Bounding box (ESRI type) to be cut out. The layout of the bounding box corresponds to the bounding box of the ESRI shape files and is as follows: [minimum longitude, minimum latitude, maximum longitude, maxmimum latitude] If country_cut_mode = 1, the cut_bbox is overwritten/ignored.
- **return_coords** (*int*) – Determines whether latitude and longitude are delievered along with gpw data (0) or only gpw_data is returned. Default: 0.
- **add_one** (*boolean*) – Determine whether the integer one is added to all cells to eliminate zero pixels. Default: 0. #TODO: Unimplemented
- **reference_year** (*int*) – reference year, available years are: 2000, 2005, 2010, 2015 (default), 2020

Returns

- **tile_temp** (*pandas.arrays.SparseArray*) – GPW data
- **lon** (*list*) – List with longitudinal infomation on the GPW data. Same dimensionality as tile_temp (only returned if return_coords is 1).
- **lat** (*list*) – list with latitudinal infomation on the GPW data. Same dimensionality as tile_temp (only returned if return_coords is 1).

climada.entity.exposures.litpop module

climada.entity.exposures.litpop.**LOGGER = <Logger climada.entity.exposures.litpop (INFO)>**
Define LitPop class.

climada.entity.exposures.litpop.**WORLD_BANK_INC_GRP = 'http://databank.worldbank.org/data/d**
Income group historical data from World bank.

climada.entity.exposures.litpop.**DEF_RES_NASA_KM = 0.5**
Default approximate resolution for NASA's nightlights in km.

climada.entity.exposures.litpop.**DEF_RES_GPW_KM = 1**
Default approximate resolution for the GPW dataset in km.

climada.entity.exposures.litpop.**DEF_RES_NASA_ARCSEC = 15**
Default approximate resolution for NASA's nightlights in arcsec.

climada.entity.exposures.litpop.**DEF_RES_GPW_ARCSEC = 30**
Default approximate resolution for the GPW dataset in arcsec.

climada.entity.exposures.litpop.**DEF_HAZ_TYPE = ''**
Default hazard type used in impact functions id, i.e. TC

class climada.entity.exposures.litpop.**LitPop**(**args*, ***kwargs*)
Bases: *climada.entity.exposures.base.Exposures*

Defines exposure values from nightlight intensity (NASA), Gridded Population data (SEDAC); distributing produced capital (World Bank), GDP (World Bank) or non-financial wealth (Global Wealth Databook by the Credit Suisse Research Institute.)

Calling sequence example: ent = LitPop() country_name = ['Switzerland', 'Austria'] ent.set_country(country_name) ent.plot()

clear()

Appending the base class clear attribute to also delete attributes which are only used here.

set_country(*countries*, ***args*)

Get LitPop based exposre for one country or multiple countries using values at reference year. If produced capital, GDP, or income group, etc. not available for that year, consider the value of the closest available year.

Parameters **countries** (*str or list*) – list of countries or single county as a sting. Countries can either be country names ('France') or country codes ('FRA'), even a mix is possible in the list.

args: Keyword arguments. The following keywords are recognised: res_km (float, optional): approx resolution in km. Default: 1km. res_arcsec (float, optional): resolution in arc-sec. Overrides

res_km if both are delivered

check_plot (boolean, optional): choose if a plot is shown at the end of the operation.

exponents (list of two integers, default = [1, 1]) defining power with which lit (nightlights) and pop (gpw) go into LitPop. To get nightlights^3 alone: [3, 0]. To use population count alone: [0, 1].

fin_mode (str, optional): define what total country economic value is to be used as an asset base and distributed to the grid: - 'gdp': gross-domestic product (Source: World Bank) - 'income_group': gdp multiplied by country's income group+1 - 'nfw': non-financial wealth (Source: Credit Suisse, of households only) - 'tw': total wealth (Source: Credit Suisse, of households only) - 'pc': produced capital (Source: World Bank), incl. manufactured or

built assets such as machinery, equipment, and physical structures (pc is in constant 2014 USD)

- 'norm': normalized by country
- 'none': LitPop per pixel is returned unchanged

admin1_calc (boolean): distribute admin1-level GDP if available? (default False)

conserve_cntrytotal (boolean): given admin1_calc, conserve national total asset value (default True) reference_year (int) adm1_scatter (boolean): produce scatter plot for admin1 validation?

plot_log(*admin1_plot=1*)

Plots the LitPop data with the color scale reprenting the values in a logarithmic scale.

Parameters **admin1_plot** (*boolean*) – whether admin1 borders should be plotted. Default=1

climada.entity.exposures.litpop.**read_bm_file**(*bm_path*, *filename*)

Reads a single NASA BlackMarble GeoTiff and returns the data. Run all required checks first.

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.**get_bm**(*required_files=array([1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0])*, ***parameters*)

Potential TODO: put cutting before zooming (faster), but with expanding bbox in order to preserve additional pixels for interpolation...

climada.entity.exposures.litpop.**admin1_validation**(*country*, *methods*, *exponents*, ***args*)

Get LitPop based exposre for one country or multiple countries using values at reference year. If GDP or income group not available for that year, consider the value of the closest available year.

Parameters

- **country** (*str*) – list of countries or single county as a string. Countries can either be country names ('France') or country codes ('FRA'), even a mix is possible in the list.
- **methods_name** (*list of str*) –
  - ['LitPop' for LitPop,
  - ['Lit', 'Pop'] for Lit and Pop,
  - ['Lit3'] for cube of night lights (Lit3)
- **exponents** (*list of 2-vectors*) –
  - [[1, 1]] for LitPop,
  - [[1, 0], [0, 1]] for Lit and Pop,
  - [[3, 0]] for cube of night lights (Lit3)

args: Keyword arguments. The following keywords are recognised: res_km (float, optional): approx resolution in km. Default: 1km. res_arcsec (float, optional): resolution in arc-sec. Overrides

res_km if both are delivered

check_plot (boolean, optional): choose if a plot is shown at the end of the operation.

climada.entity.exposures.litpop.**exposure_set_admin1**(*exposure*, *res_arcsec*)

add admin1 ID and name to exposure dataframe.

Parameters

- **exposure** – exposure instance
- **res_arcsec** – resolution in arc seconds, needs to match exposure resolution

Returns exposure instance with 2 extra columns: admin1 & admin1_ID

Return type exposure

climada.entity.exposures.litpop.**to_sparse_dataframe**(*ndarr*)

Turns a 2-dim ndarray into a DataFrame with little memory footprint.

Parameters **ndarr** (*numpy.ndarray*) – 2 dimensional

Returns **sparse dataframe**

Return type pandas.DataFrame

climada.entity.exposures.nightlight module

climada.entity.exposures.nightlight.**NOAA_SITE = 'https://ngdc.noaa.gov/eog/data/web_data/v**
NOAA's URL used to retrieve nightlight satellite images.

climada.entity.exposures.nightlight.**NOAA_RESOLUTION_DEG = 0.008333333333333333**
NOAA nightlights coordinates resolution in degrees.

climada.entity.exposures.nightlight.**NASA_RESOLUTION_DEG = 0.004166666666666667**
NASA nightlights coordinates resolution in degrees.

climada.entity.exposures.nightlight.**NASA_TILE_SIZE = (21600, 21600)**
NASA nightlights tile resolution.

climada.entity.exposures.nightlight.**NOAA_BORDER = (-180, -65, 180, 75)**
NOAA nightlights border (min_lon, min_lat, max_lon, max_lat)

climada.entity.exposures.nightlight.**NASA_SITE = 'https://www.nasa.gov/specials/blackmarble/**
NASA nightlight web url.

climada.entity.exposures.nightlight.**BM_FILENAMES = ['BlackMarble_*_A1_geo_gray.tif', 'Black**
Nightlight NASA files which generate the whole earth when put together.

climada.entity.exposures.nightlight.**check_required_nl_files**(*bbox*, **coords*)

Determines which of the satellite pictures are necessary for a certain bounding box (e.g. country)

Parameters

- **either** –

  bbox (1x4 tuple): bounding box from shape (min_lon, min_lat, max_lon, max_lat)

- **or** – min_lon (float): (=min_lon) Western-most point in decimal degrees min_lat (float): Southern-most point in decimal degrees max_lon (float): Eastern-most point in decimal degrees max_lat (float): Northern-most point in decimal degrees

Returns

Array indicating the required files for the current operation with a Boolean value (1: file is required, 0: file is not required).

Return type req_files (array)

climada.entity.exposures.nightlight.**check_nl_local_file_exists**(*required_files=array([1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0])*, *check_path='/home/docs/checkouts/readt python/checkouts/v1.5.1/data/system'*, *year=2016*)

Checks if BM Satellite files are avaialbe and returns a vector denoting the missing files.

Parameters

- **check_path** (*str*) – absolute path where files are stored. Default: SYSTEM_DIR
- **required_files** (*array*) – Boolean array of dimension (8,) with which some files can be skipped. Only files with value 1 are checked, with value zero are skipped
- **year** (*int*) – year of the image

Returns

Denotes if the all required files exist (Boolean values)

Return type files_exist (array)

climada.entity.exposures.nightlight.**download_nl_files**(*req_files=array([1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0])*, *files_exist=array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0])*, *dwnl_path='/home/docs/checkouts/readthedocs.org/us python/checkouts/v1.5.1/data/system'*, *year=2016*)

Attempts to download nightlight files from NASA webpage.

Parameters

- **req_files** (*array*) – Boolean array which indicates the files required for the current operation (0-> skip, 1-> download). Can be obtained by check_required_nightlight_files
- **files_exists** (*array*) – Boolean array which indicates if the files already exist locally and should not be downloaded (0-> download, 1-> skip). Can be obtained by function check_nightlight_local_file_exists
- **dwnl_path** (*str*)

Returns Absolute path to file storage.

Return type path_str (str)

climada.entity.exposures.nightlight.**load_nightlight_nasa**(*bounds*, *req_files*, *year*)

Get nightlight from NASA repository that contain input boundary.

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.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 str (file_name of unzipped file) sparse.csr_matrix (nightlight)

climada.entity.exposures.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 (str)

climada.entity.exposures.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*) – reference year
- **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.open_street_map module

climada.entity.exposures.open_street_map.**get_features_OSM**(*bbox*, *types*, *save_path='/home/docs/checkouts/readthedocs.o python/checkouts/v1.5.1/doc'*, *check_plot=1*)

Get shapes from all types of objects that are available on Open Street Map via an API query and save them as geodataframe.

Parameters

- **bbox** (*array*) – List of coordinates in format [South, West, North, East]
- **types** (*list*) – List of features items that should be downloaded from OSM, e.g. {'natural','waterway','water', 'landuse=forest','landuse=farmland', 'landuse=grass','wetland'}
- **save_path** (*str*) – String with absolute path for saving output. Default is cwd
- **check_plot** – default is 1 (yes), else 0.

Returns

combined GeoDataframe with all features saved as "OSM_features_lat_lon". Shapefiles with correct geometry (LineStrings,Polygons, MultiPolygons)

for each of requested OSM feature saved as "item_gdf_all_lat_lon"

Return type OSM_features_gdf_combined(gdf)

Example 1: Houses_47_8 = get_features_OSM([47.16, 8.0, 47.3, 8.0712], {'building'}, save_path = save_path, check_plot=1)

Example 2:

Low_Value_gdf_47_8 = get_features_OSM([47.16, 8.0, 47.3, 8.0712], {'natural','water', 'waterway', 'landuse=forest', 'landuse=farmland', 'landuse=grass', 'wetland'}, save_path = save_path, check_plot=1)

climada.entity.exposures.open_street_map.**get_highValueArea**(*bbox*, *save_path='/home/docs/checkouts/readthedocs python/checkouts/v1.5.1/doc'*, *Low_Value_gdf=None*, *check_plot=1*)

In case low-value features were queried with get_features_OSM(), calculate the "counter-shape" representig high value area for a given bounding box.

Parameters

- **bbox** (*array*) – List of coordinates in format [South, West, North, East]
- **save_path** (*str*) – path for results
- **Low_Value_gdf** (*str*) – absolute path of gdf of low value items which is to be inverted. If left empty, searches for OSM_features_gdf_combined_lat_lon.shp in save_path.
- **checkplot**

Returns GeoDataFrame of High Value Area as High_Value_Area_lat_lon

Return type High_Value_Area (gdf)

Example

High_Value_gdf_47_8 = get_highValueArea([47.16, 8.0, 47.3, 8.0712], save_path = save_path, Low_Value_gdf = save_path+'/Low_Value_gdf_combined_47_8.shp')

important: Use same bbox and save_path as for get_features_OSM().

climada.entity.exposures.open_street_map.**get_osmstencil_litpop**(*bbox*, *country*, *mode*, *highValueArea=None*, *save_path='/home/docs/checkouts/readth python/checkouts/v1.5.1/doc'*, *check_plot=1*, ***kwargs*)

Generate climada-compatible exposure by downloading LitPop exposure for a bounding box, corrected for centroids which lie inside a certain high-value multipolygon area from previous OSM query.

Parameters

- **bbox** (*array*) – List of coordinates in format [South, West, North, East]
- **Country** (*str*) – ISO3 code or name of country in which bbox is located
- **highValueArea** (*str*) – path of gdf of high-value area from previous step. If empty, searches for cwd/High_Value_Area_lat_lon.shp
- **mode** (*str*) – mode of re-assigning low-value points to high-value points. "nearest", "even", or "proportional"
- **kwargs** (*dict*) – arguments for LitPop set_country method

Returns

(CLIMADA-compatible) with re-allocated asset values with name exposure_high_lat_lon

Return type exp_sub_high_exp (Exposure)

Example

exposure_high_47_8 = get_osmstencil_litpop([47.16, 8.0, 47.3, 8.0712], 'CHE',"proportional", highValueArea = save_path + '/High_Value_Area_47_8.shp' , save_path = save_path)

climada.entity.exposures.open_street_map.**make_osmexposure**(*highValueArea*, *mode='default'*, *country=None*, *save_path='/home/docs/checkouts/readthedocs.o python/checkouts/v1.5.1/doc'*, *check_plot=1*, ***kwargs*)

Generate climada-compatiple entity by assigning values to midpoints of individual house shapes from OSM query, according to surface area and country.

Parameters

- **highValueArea** (*str*) – absolute path for gdf of building features queried from get_features_OSM()
- **mode** (*str*) – "LitPop" or "default": Default assigns a value of 5400 Chf to each m2 of building, LitPop assigns total LitPop value for the region proportionally to houses (by base area of house)

- **Country** (*str*) – ISO3 code or name of country in which entity is located. Only if mode = LitPop
- **kwargs** (*dict*) – arguments for LitPop set_country method

Returns

(CLIMADA-compatible) with allocated asset values. Saved as exposure_buildings_mode_lat_lon.h5

Return type exp_building (Exposure)

Example

buildings_47_8 = make_osmexposure(save_path + '/OSM_features_47_8.shp', mode="default", save_path = save_path, check_plot=1)

climada.entity.exposures.spam_agrar module

climada.entity.exposures.spam_agrar.**DEF_HAZ_TYPE = 'CP'**
Default hazard type used in impact functions id.

climada.entity.exposures.spam_agrar.**FILENAME_SPAM = 'spam2005V3r2_global'**
Add Docstring!

Type TODO

climada.entity.exposures.spam_agrar.**FILENAME_CELL5M = 'cell5m_allockey_xy.csv'**
Add Docstring!

Type TODO

climada.entity.exposures.spam_agrar.**FILENAME_PERMALINKS = 'spam2005V3r2_download_permalinks**
Add Docstring!

Type TODO

climada.entity.exposures.spam_agrar.**BUFFER_VAL = -340282306073709652508363335590014353408**
Hard coded value which is used for NANs in original data

class climada.entity.exposures.spam_agrar.**SpamAgrar**(**args*, ***kwargs*)
Bases: *climada.entity.exposures.base.Exposures*

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

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

init_spam_agrar(***parameters*)
initiates agriculture exposure from SPAM data:

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

Optional parameters:

data_path (str): absolute path where files are stored. Default: SYSTEM_DIR

country (str): Three letter country code of country to be cut out. No default (global)

name_adm1 (str): Name of admin1 (e.g. Federal State) to be cut out. No default

name_adm2 (str): Name of admin2 to be cut out. No default

spam_variable (str): select one agricultural variable: 'A' physical area 'H' harvested area 'P' production 'Y' yield 'V_agg' value of production, aggregated to all crops,

food and non-food (default)

Warning: for A, H, P and Y, currently all crops are summed up

spam_technology (str): select one agricultural technology type: 'TA' all technologies together, ie complete crop (default) 'TI' irrigated portion of crop 'TH' rainfed high inputs portion of crop 'TL' rainfed low inputs portion of crop 'TS' rainfed subsistence portion of crop 'TR' rainfed portion of crop (= TA - TI, or TH + TL + TS) ! different impact_ids are assigned to each technology (1-6)

save_name_adm1 (Boolean): Determines how many aditional data are saved: False: only basics (lat, lon, total value), region_id per country True: like 1 + name of admin1

haz_type (str): hazard type abbreviation, e.g. 'DR' for Drought or 'CP' for CropPotential

Returns:

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 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 matplotlib.axes._subplots.AxesSubplot

check()
Check consistent instance data.

Raises **ValueError** –

climada.entity.impact_funcs.drought module

class climada.entity.impact_funcs.drought.**IFDrought**
Bases: *climada.entity.impact_funcs.base.ImpactFunc*

Impact function for droughts.

__init__()
Empty initialization.

Parameters

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

Raises **ValueError** –

set_default()

set_default_sum()

set_default_sumthr()

set_step()

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 suppossed 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 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 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 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.relative_cropyield module

class climada.entity.impact_funcs.relative_cropyield.**IFRelativeCropyield**

Bases: *climada.entity.impact_funcs.base.ImpactFunc*

Impact functions for agricultural droughts.

__init__()

Empty initialization.

set_relativeyield()

Impact functions defining the impact as intensity

climada.entity.impact_funcs.river_flood module

class climada.entity.impact_funcs.river_flood.**IFRiverFlood**

Bases: *climada.entity.impact_funcs.base.ImpactFunc*

Impact functions for tropical cyclones.

__init__()

Empty initialization.

set_RF_IF_Africa()

set_RF_IF_Asia()

set_RF_IF_Europe()

set_RF_IF_NorthAmerica()

set_RF_IF_Oceania()

set_RF_IF_SouthAmerica()

climada.entity.impact_funcs.storm_europe module

class climada.entity.impact_funcs.storm_europe.**IFStormEurope**
Bases: *climada.entity.impact_funcs.base.ImpactFunc*

Impact functions for tropical cyclones.

__init__()
Empty initialization.

set_schwierz(*if_id=1*)
Using the impact functions of Schwierz et al. 2011.

set_welker(*if_id=1*)
Using the impact functions of Welker et al. 2020 (in submission). 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.**IFTropCyclone**
Bases: *climada.entity.impact_funcs.base.ImpactFunc*

Impact functions for tropical cyclones.

__init__()
Empty initialization.

set_emanuel_usa(*if_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

- **if_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<=scale<=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 action

Type str

haz_type
related hazard type (peril), e.g. TC

Type str

color_rgb
integer array of size 3. Gives 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

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

mdd_impact
parameter a and b of the impact over the mean damage degree

Type tuple

paa_impact
parameter a and b of the impact over the percentage of affected assets

Type tuple

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** (*Exposures*) – exposures instance
- **imp_fun_set** (*ImpactFuncSet*) – impact functions instance
- **hazard** (*Hazard*) – hazard instance

Returns Impact (resulting impact), Impact (insurance layer)

apply(*exposures*, *imp_fun_set*, *hazard*)
Implement measure with all its defined parameters.

Parameters

- **exposures** (*Exposures*) – exposures instance
- **imp_fun_set** (*ImpactFuncSet*) – impact functions instance
- **hazard** (*Hazard*) – hazard instance

Returns Exposures, ImpactFuncSet, Hazard

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
cotains Measure classes. It's not suppossed 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 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 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_frq': '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_frq': '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', '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_frq': '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', '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 initializator

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.

9.1.3 climada.hazard package

climada.hazard.centroids package

climada.hazard.centroids.centr module

class climada.hazard.centroids.centr.**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 (transform needs to contain upper left corner!)

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_la
Variables whose size will be checked

__init__()
Initialize to None raster and vector

check()
Check that either raster meta attribute is set or points lat, lon and geometry.crs. Check attributes sizes

equal(*centr*)
Return true if two centroids equal, false otherwise

Parameters **centr** (*Centroids*) – centroids to compare

Returns 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.

```
>>> 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={'init': 'epsg:4326', 'no_defs': True}*)
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={'init': 'epsg:4326', 'no_defs': True}*)
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={'init': 'epsg:4326', 'no_defs': True}*)
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=[1], src_crs=None, window=False, geometry=False, dst_crs=False, transform=None, width=None, height=None, resampling=<Resampling.nearest: 0>*)
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 proving dst_crs and/or (transform, width, height).

Parameters

- **file_pth** (*str*) – path of the file
- **band** (*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
- **wdith** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp,.Resampling optional*) – resampling function used for reprojection to dst_crs

Raises **ValueError** –

Returns np.array

set_vector_file (*file_name*, *inten_name=['intensity']*, *dst_crs=None*)
Read vector file format supported by fiona. Each intensity name is considered an event. Returns intensity array with shape (len(inten_name), len(geometry)).

Parameters

- **file_name** (*str*) – vector file with format supported by fiona and 'geometry' field.
- **inten_name** (*list(str)*) – list of names of the columns of the intensity of each event.
- **dst_crs** (*crs, optional*) – reproject to given crs

Returns np.array

read_mat (*file_name*, *var_names={'field_names': ['centroids', 'hazard'], 'var_name': {'admin0_iso3': 'admin0_ISO3', 'admin0_name': 'admin0_name', 'comment': 'comment', 'dist_coast': 'distance2coast_km', 'lat': 'lat', 'lon': 'lon', 'region_id': 'NatId'}}*)
Read centroids from CLIMADA's MATLAB version

Parameters

- **file_name** (*str*) – absolute or relative file name
- **var_names** (*dict, default*) – name of the variables

Raises **KeyError** –

read_excel (*file_name*, *var_names={'col_name': {'lat': 'latitude', 'lon': 'longitude', 'region_id': 'region_id'}, 'sheet_name': 'centroids'}*)
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 raster or points. Raster needs to have the same resolution

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), y_close (float), idx_close (int)

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*)
Computes approximated area_pixel values: differentiated per latitude. area in m*m. 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_dist_coast(*signed=False*, *precomputed=False*, *scheduler=None*)
Set dist_coast attribute for every pixel or point. Distance to coast is computed 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 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.
- **sel_cen** (*np.array*) – 1-dim mask, overrides reg_id and extent

Returns 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*, ***kwargs*)
Plot centroids scatter points over earth.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for scatter matplotlib function

Returns 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 gpd.GeoSeries

empty_geometry_points()
Removes 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 size 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. Might take some time.

set_geometry_points(*scheduler=None*)
Set geometry attribute of gpd.GeoSeries with Points from latitude and longitude attributes if geometry not present.

Parameters **scheduler** (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

climada.hazard.emulator package

climada.hazard.emulator.const module

climada.hazard.emulator.emulator module

climada.hazard.emulator.geo module

climada.hazard.emulator.random module

climada.hazard.emulator.stats module

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', 'u
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').

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.

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: 0>*)
Append intensity and fraction from raster file. 0s put to the masked values. File can be partially read using window OR geometry. Alternatively, CRS and/or transformation can be set using dst_crs and/or (transform, width and height).

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
- **wdith** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp,.Resampling optional*) – resampling function used for reprojection 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: 0>*, *resampl_fract=<Resampling.nearest: 0>*)
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
- **wdith** (*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 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, default*) – 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 within provided date and/or (historical or synthetical) and/or region. Frequency of the events may need to be recomputed!

Parameters

- **event_names** (*list(str), optional*) – names of event
- **date** (*tuple(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)
- **reg_id** (*int, optional*) – region identifier of the centroids's region_id attibute
- **reset_frequency** (*boolean*) – change frequency of events proportional to difference between first and last year (old and new) default = False

Returns Hazard or children

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 np.array

plot_rp_intensity (*return_periods=25, 50, 100, 250*, *smooth=True*, *axis=None*, ***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
- **kwargs** (*optional*) – arguments for pcolormesh matplotlib function used in event plots

Returns matplotlib.axes._subplots.AxesSubplot, np.ndarray (return_periods.size x num_centroids)

plot_intensity(*event=None*, *centr=None*, *smooth=True*, *axis=None*, ***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 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 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 np.array(int)

get_event_name(*event_id*)
Get the name of an event id.

Parameters **event_id** (*int*) – id of the event

Returns 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 list(str)

calc_year_set()
From the dates of the original events, get number yearly events.

Returns key are years, values array with event_ids of that year

Return type dict

append(*hazard*)
Append events and centroids in hazard.

Parameters **hazard** (*Hazard*) – Hazard instance to append to current

Raises **ValueError** –

remove_duplicates()
Remove duplicate events (events with same name and date).

set_frequency(*yearrange=None*)
Set hazard frequency from yearrange or intensity matrix.

Optional parameters:

yearrange (tuple or list): 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
Returns 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

concatenate(*haz_src*, *append=False*)
Concatenate events of several hazards

Parameters

- **haz_src** (*list*) – Hazard instances with same centroids and units

- **append** (*bool*) – If True, append the concatenated hazards to this instance, otherwise replace all data in this instance by the concatenated data. Default: False.

climada.hazard.drought module

climada.hazard.isimip_data module

climada.hazard.landslide module

climada.hazard.low_flow module

climada.hazard.relative_cropyield module

climada.hazard.river_flood module

climada.hazard.storm_europe module

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_rainfield module

climada.hazard.tc_tracks module

climada.hazard.tc_tracks_forecast module

climada.hazard.tc_tracks_synth module

climada.hazard.trop_cyclone module

9.1.4 climada.util package

climada.util.alpha_shape module

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*) – nump array used as default value

Raises **ValueError** –

Returns Filled array

climada.util.config module

climada.util.config.**setup_logging**(*log_level='DEBUG'*)
Setup logging configuration

climada.util.config.**setup_conf_user**()
Setup climada configuration

climada.util.constants module

climada.util.constants.**SOURCE_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/clima**
climada directory

climada.util.constants.**DATA_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/climada**
Folder containing the data

climada.util.constants.**SYSTEM_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/clima**
Folder containing the data used internally

climada.util.constants.**HAZ_DEMO_MAT = '/home/docs/checkouts/readthedocs.org/user_builds/cli**
hurricanes from 1851 to 2011 over Florida with 100 centroids.

Type Hazard demo from climada in MATLAB

climada.util.constants.**HAZ_DEMO_FLDDPH = '/home/docs/checkouts/readthedocs.org/user_builds/**
NetCDF4 Flood depth from isimip simulations

climada.util.constants.**HAZ_DEMO_FLDFRC = '/home/docs/checkouts/readthedocs.org/user_builds/**
NetCDF4 Flood fraction from isimip simulations

climada.util.constants.**ENT_TEMPLATE_XLS = '/home/docs/checkouts/readthedocs.org/user_builds**
Entity 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 = '/home/docs/checkouts/readthedocs.org/user_build**
Global centroids

climada.util.constants.**GLB_CENTROIDS_NC = '/home/docs/checkouts/readthedocs.org/user_builds**
For backwards compatibility, it remains available under its old name.

climada.util.constants.**ISIMIP_GPWV3_NATID_150AS = '/home/docs/checkouts/readthedocs.org/use**
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: '/home/docs/checkouts/readthedocs.org/us**
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.**DEMO_GDP2ASSET = '/home/docs/checkouts/readthedocs.org/user_builds/c**
Exposure demo file for GDP2Asset

climada.util.constants.**RIVER_FLOOD_REGIONS_CSV = '/home/docs/checkouts/readthedocs.org/use**
Look-up table for river flood module

climada.util.constants.**TC_ANDREW_FL = '/home/docs/checkouts/readthedocs.org/user_builds/cl**
Tropical cyclone Andrew in Florida

climada.util.constants.**HAZ_DEMO_H5 = '/home/docs/checkouts/readthedocs.org/user_builds/cli**
IBTrACS from 1990 to 2004 over Florida with 2500 centroids.

Type Hazard demo in hdf5 format

climada.util.constants.**EXP_DEMO_H5 = '/home/docs/checkouts/readthedocs.org/user_builds/cli**
Exposures over Florida

climada.util.constants.**WS_DEMO_NC = ['/home/docs/checkouts/readthedocs.org/user_builds/cli**
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.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 = '/home/docs/checkouts/readthedocs.org/user_bu**
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 dowload

climada.util.coordinates.**latlon_to_geosph_vector**(*lat*, *lon*, *rad=False*, *basis=False*)
Convert lat/lon coodinates 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 < lon - center <= 180

The input data is modified in place (!) using the following operations:

(lon) -> (lon $\pm$ 360)

Parameters

- **lon** (*np.array*) – Longitudinal coordinates
- **center** (*float, optional*) – Central longitude value to use instead of 0.

Returns np.array (same as input)

climada.util.coordinates.**latlon_bounds**(*lat*, *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 *lon_min* <= *lon* <= *lon_max* in general!

Usually, an application of this function is followed by a renormalization of longitudinal values around the longitudinal middle value:

```
>>> bounds = latlon_bounds(lat, 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

```
>>> 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 tuple (lon_min, lat_min, lon_max, lat_max)

climada.util.coordinates.**dist_approx**(*lat1*, *lon1*, *lat2*, *lon2*, *log=False*, *normalize=True*, *method='equirect'*)

Compute approximation of geodistance in km

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": equirect-angular; very fast, good only at small distances. * "geosphere": spherical approximation, slower, but much higher accuracy. Default: "equirect".

Returns

- **dists** (*ndarray of floats, shape (nbatch, nx, ny)*) – Approximate distances in km.
- **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.**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 Polygones 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 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 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 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*) –
  - 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*) –
  - 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 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 np.array

climada.util.coordinates.**get_land_geometry**(*country_names=None*, *extent=None*, *resolution=10*)

Get union of all the countries or the provided ones 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 shapely.geometry.multipolygon.MultiPolygon

climada.util.coordinates.**coord_on_land**(*lat*, *lon*, *land_geom=None*)

Check if point is on land (True) or water (False) of provided coordinates. All globe considered if no input countries.

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*) – profiles of land.

Returns 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 str

Raises `ValueError` –

climada.util.coordinates.**get_country_geometries**(*country_names=None*, *extent=None*, *resolution=10*)

Returns a gpd GeoSeries of natural earth multipolygons of the specified countries, resp. the countries that lie within the specified 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 ISO3 names of countries, e.g ['ZWE', 'GBR', 'VNM', 'UZB']
- **extent** (*tuple, optional*) – (min_lon, max_lon, min_lat, max_lat) 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 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.**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_iso_alpha2numeric**(*isos*)

Convert ISO 3166-1 alpha-3 to numeric-3 codes

Parameters **isos** (*str or list of str*) – ISO codes of countries (or single code).

Returns int or list of int

climada.util.coordinates.**country_natid2iso**(*natids*)
Convert internal NatIDs to ISO 3166-1 alpha-3 codes

Parameters **natids** (*int or list of int*) – Internal NatIDs of countries (or single ID).

Returns str or list of str

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 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 GeoDataFrame.

Returns 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 np.array(int)

climada.util.coordinates.**get_admin1_info**(*country_names*)
Provide 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) admin1_shapes (dict)

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 float

climada.util.coordinates.**get_resolution**(**coords*, *min_resol=1e-08*)
Compute resolution of 2-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 pair of floats

climada.util.coordinates.**pts_to_raster_meta**(*points_bounds*, *res*)

Transform vector data coordinates to raster. Returns number of rows, columns and affine transformation

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 int, int, affine.Affine

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.**equal_crs**(*crs_one*, *crs_two*)

Compare two crs

Parameters

- **crs_one** (*dict or string or wkt*) – user crs
- **crs_two** (*dict or string or wkt*) – user crs

Returns 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: 0>*)

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 proving dst_crs and/or (transform, width, height). Returns matrix in 2d: band x coordinates in 1d (can be reshaped to band x height x width)

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
- **wdith** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform

- **resampling** (*rasterio.warp.Resampling optional*) – resampling function used for reprojection to dst_crs

Returns dict (meta), np.array (band x coordinates_in_1d)

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 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 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 numpy 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 np.array, affine.Affine

climada.util.coordinates.**read_vector**(*file_name*, *field_name*, *dst_crs=None*)

Read vector file format supported by fiona. Each field_name name is considered an event.

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 np.array (lat), np.array (lon), geometry (GeiSeries), np.array (value)

climada.util.coordinates.**write_raster**(*file_name*, *data_matrix*, *meta*, *dtype=<class 'numpy.float32'>*)

Write raster in GeoTiff format

Parameters

- **fle_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 (transform needs to contain upper left corner!)
- **dtype** (*numpy dtype*) – a numpy dtype

climada.util.coordinates.**points_to_raster**(*points_df*, *val_names=None*, *res=0.0*, *raster_res=0.0*, *scheduler=None*)

Compute raster matrix and transformation from value column

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
- **scheduler** (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

Returns np.array, affine.Affine

climada.util.coordinates.**set_df_geometry_points**(*df_val*, *scheduler=None*)
Set given geometry to given dataframe using dask if scheduler

Parameters

- **df_val** (*DataFrame or GeoDataFrame*) – contains latitude and longitude columns
- **scheduler** (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

climada.util.coordinates.**fao_code_def**()
Generates list of FAO country codes and corresponding ISO numeric-3 codes

Returns list of ISO numeric-3 codes faocode_list (list): list of FAO country codes

Return type iso_list (list)

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 ISO numeric-3 codes of countries (or single code)

Return type output_iso (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 FAO country codes of countries (or single code)

Return type output_faocode (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 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 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 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 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 int

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*) – number of expect list elements
- **values** (*object or list(object)*) – values to check and transform
- **val_name** (*str*) – name of the variable values

Returns 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 list

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 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 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("/pathto/dummy.mat")
```

Contents contains the Matlab data and its reference in a dictionary.

```
>>> contents = read("/pathto/dummy.mat", True)
```

Raises **Exception while reading** –

climada.util.hdf5_handler.**get_string**(*array*)
Form string from input array of unisgned integers.

Parameters **array** – array of integers

Returns 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 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 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 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

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 multiplicated 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>*, *axes=None*, ***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.
- **buffer** (*float, optional*) – border to add to coordinates
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **proj** (*ccrs*) – coordinate reference system used in coordinates
- **kwargs** (*optional*) – arguments for hexbin matplotlib function

Returns cartopy.mpl.geoaxes.GeoAxesSubplot

Raises **ValueError** –

climada.util.plot.**geo_im_from_array**(*array_sub*, *coord*, *var_name*, *title*, *proj=None*, *smooth=True*, *axes=None*, ***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 ploted 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*) – coordinate reference system used in coordinates
- **smooth** (*bool, optional*) – smooth plot to RESOLUTIONxRESOLUTION. Default: True.
- **kwargs** (*optional*) – arguments for pcolormesh matplotlib function.

Returns cartopy.mpl.geoaxes.GeoAxesSubplot

Raises **ValueError** –

climada.util.plot.**make_map**(*num_sub=1*, *figsize=(9, 13)*, *proj=<cartopy.crs.PlateCarree object>*)
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, PlateCarree default.

Returns 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, PlateCarree default.

climada.util.plot.**add_populated_places**(*axis*, *extent*, *proj=<cartopy.crs.PlateCarree object>*)
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, PlateCarree default.

climada.util.plot.**add_cntry_names**(*axis*, *extent*, *proj=<cartopy.crs.PlateCarree object>*)
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, PlateCarree default.

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*)

Returns 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.

- genindex
- modindex

CHAPTER

TEN

LICENSE

BIBLIOGRAPHY

[IPCC2014] 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.

PYTHON MODULE INDEX

C

INDEX

Symbols

A

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Y