flood_example_spain_GOV_Power

May 24, 2023

1 Hazard RiverFlood Example for Spain Using JRC Data

1.1 Summary

A hazard is generated using the Hazard() class from Climada. We have downloaded the flood depth return period data from JRC which is derived from this paper. The data can be loaded from an excel or nc file into the Hazard object although we've decided to do it manually for a better comprehension. The Global Power Plant Database is used as exposures portfolio to derive the economic impacts.

1.2 Links

Find a full tutorial of climada platform here.

1.3 How is this tutorial structured?

```
Part 1: Load JRC Data

Part 2: Generate a HAzard object from scratch

Part 3: Defining exposures from Global Power Plant Database

Part 4: Setting JRC damage functions

Part 5: Deriving flood economic-impact

Part 6: Risk Measures available at Climada
```

1.4 Part 1: Data Source

The data is from the Joint Research Center (JRC) and covers flood depth return periods for Europe. The data can be downloaded from JRC for six different return periods: 10, 20, 50, 100, 200 and 500 years. Since fraction data is nor provided we set it to 1 for every centroid.

First of all we import all the python external libraries that we will need all along the notebook.

The jrc data is heavy to be loaded in RAM Memory for a standard personal computer. For example, the tif file for 50 year return periods consumes 10Gb of RAM Memory. Being this the case, we have loaded the data by 512 block size windows and summarizing each window taking a matrix subsample of the coordinates and the flood intensity.

The coordinates system of the map is EPSG:3035 (ETRS89-extended / LAEA Europe) and needs to be translated to Latitud-Longitud coordinate system. See EPSG official website for more information.

This process is done by load_flood_dph_jrc(rp) function below and takes around 10 minutes for each return period. After a return period is processed once it is stored in h5 format and doesn't need to be processed again.

[1]:
```
# Load External Libreries
import numpy as np
import rasterio
import pandas as pd
import xarray as xr
import math
from scipy import sparse
import pyproj
from tqdm import tqdm
import os
from pyproj.crs import CRS
import geopandas as gpd
from affine import Affine

remove_warnings = True
if remove_warnings:
    import warnings
    warnings.filterwarnings("ignore")
```

[2]:
```
# Function to load jrc data given the return period (rp).
def load_flood_dph_jrc(rp):
    """
    This function loads jrc flood depth data by windows of block size 512.␣
 ↪Translates from EPSD:3035 to EPSD:4326
    coordinates system and return flood depth, latitude and longitude.

    Parameter:
        rp (String): the return period to identify map. Must be 010, 020, 050,␣
 ↪100, 200 or 500.

    Returns:
        bands_ (List): flood depth.
        coords_lon_ (List): longitude coordinates.
        coords_lat_ (List): latitude coordinates.
    """

    # transform from crs 3035 to lat long
    proj = pyproj.Transformer.from_crs(25830, 4326, always_xy=True,␣
 ↪authority='EPSG')
```

```
    inputfile = os.path.join(os.getcwd(), 'esp_gov_flood_data/
↪ESNZSNCZIMPFT010E77.tif')
    with rasterio.open(inputfile) as src:
        block_size = int(512)
        step = int(block_size / 10)
        X, Y = np.mgrid[0:block_size:step, 0:block_size:step]
        width = src.width
        height = src.height

        bands_ = []
        coords_lat_ = []
        coords_lon_ = []
        for i in tqdm(range(0, width, block_size)):
            for j in range(0, height, block_size):

                window = rasterio.windows.Window(i, j, block_size, block_size)
                band = src.read(1, window=window)

                if band.shape == (block_size, block_size):
                    pass
                else:
                    continue

                aux = band[np.where(band > 0)]
                if aux.size == 0:
                    max_flood_depth = src.nodata
                else:
                    max_flood_depth = np.percentile(aux, 50)
                band[X,Y] = max_flood_depth
                bands_.append(band[X,Y])

                cols, rows = np.meshgrid(np.arange(i, i+block_size), np.
↪arange(j, j+block_size))
                xs, ys = rasterio.transform.xy(src.transform, rows, cols)
                lons, lats = proj.transform(np.array(xs),  np.array(ys))

                coords_lon_.append(lons[X,Y])
                coords_lat_.append(lats[X,Y])

        return bands_, coords_lon_, coords_lat_
```

[3]:
```
rp = 10
jrc_df_name = 'jrc{}rp.h5'.format(rp)
if jrc_df_name not in os.listdir(os.path.join(os.getcwd(), 'jrc_flood_data')):
    # Load flood depth in windows for block size 512 and step size.
    bands_, coords_lon_, coords_lat_ = load_flood_dph_jrc(rp)
```

```
    # Convert list to numpy and flatten arrays.
    coords_lat_ = np.array(coords_lat_).flatten()
    coords_lon_ = np.array(coords_lon_).flatten()
    bands_ = np.array(bands_).flatten()

    # Round lat-lon floating point
    coords_lat_ = coords_lat_.round(5)
    coords_lon_ = coords_lon_.round(5)

    jrc_df = pd.DataFrame([bands_, coords_lat_, coords_lon_]).transpose()
    jrc_df.columns = ['dph', 'lat', 'lon']
    jrc_df.to_hdf(os.path.join(os.getcwd(), 'jrc_flood_data', jrc_df_name),␣
 ↪key='df')

else:
    jrc_df = pd.read_hdf(os.path.join(os.getcwd(), 'jrc_flood_data',␣
 ↪jrc_df_name))
    bands_ = jrc_df.dph.values
    coords_lon_ = jrc_df.lon.values
    coords_lat_ = jrc_df.lat.values
```

[4]:
```
jrc_df.head()
```

[4]:

	dph	lat	lon
0	999.0	43.61324	-6.81679
1	999.0	43.61326	-6.81616
2	999.0	43.61328	-6.81552
3	999.0	43.61330	-6.81489
4	999.0	43.61332	-6.81426

1.5 Part 2: Generating a Hazard Object

To create a Hazard object it is mandatory to set the flood intensity, fraction and coordinates in its specific format.

The centroid object inside Hazard object must be created too.

[5]:
```
# Import Hazard class from climada
from climada.hazard.base import Hazard
from climada.hazard.tag import Tag as TagHazard
```

[6]:
```
haz_type = 'RF'
tag = TagHazard(haz_type)
rf = Hazard()
```

1.5.1 Centroid Object Mandatory variables

The lon, lat and geometry arguments are mandatory, the first ones are numpy arrays and the last one a CRS object.

[7]:
```
rf.centroids.geometry = gpd.GeoSeries(crs=CRS.from_epsg(4326))
rf.centroids.lon = np.array(coords_lon_)
rf.centroids.lat = np.array(coords_lat_)
```

[8]:
```
#rf.centroids.set_lat_lon_to_meta()
rf.centroids.meta['transform'] = Affine(10,0,0,0,-10,0)
rf.centroids.meta['width'] = 16
rf.centroids.meta['height'] = coords_lon_.shape[0] / 16
rf.centroids.meta['crs'] = CRS.from_epsg(4326)
```

1.5.2 Hazard Object Mandatory variables

The intensity and fraction are mandantory and of type sparse.csr_matrix. The fraction is not provided by JRC so we set it to 1.

The frequency is needed for impact calculation and is the inverse of the return period.

[9]:
```
flddph = [0 if b == jrc_df.dph.values[0] else b for b in bands_]
rf.intensity = sparse.csr_matrix(flddph)
rf.fraction = sparse.csr_matrix(np.ones(len(bands_)))
rf.units = 'm'
rf.frequency = np.array([1 / float(rp)])
rf.frequency_unit = '1/year'
rf.date = np.array([730303])
rf.event_name = np.array(['2000'])
rf.event_id = np.array([1])
rf.tag = tag
```

1.5.3 Check that the Hazard and Centroid Objects are well defined

[10]:
```
rf.check()
```

1.5.4 The method set_region_id() sets an id for every lat-long point

The id is related to the country code. See IBAN. This operation can take several minutes.

[11]:
```
rf.centroids.set_region_id()
```

1.5.5 The country code for Spain is 724

[12]:
```
country_code = 724
```

[13]:
```
country_code in rf.centroids.region_id
```

[13]:
```
True
```

1.5.6 Filter the data by country code

[14]:
```
rf = rf.select(reg_id = country_code) # ESP
```

1.5.7 How is the event frequency defined?

The frequency is defined to be of 1 year return period by default (frequency_unit attribute).

Frecuency = 1 / return period

[15]:
```
rf.frequency
```

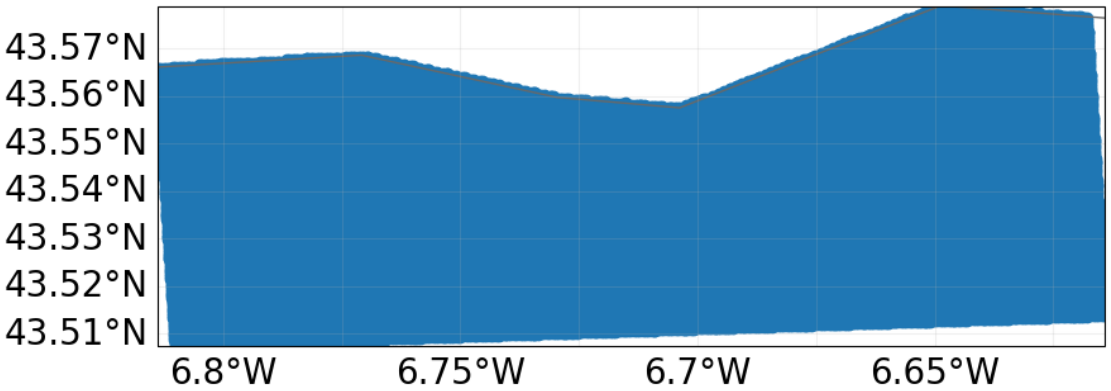
[15]:
```
array([0.1])
```

[16]:
```
rf.frequency_unit
```

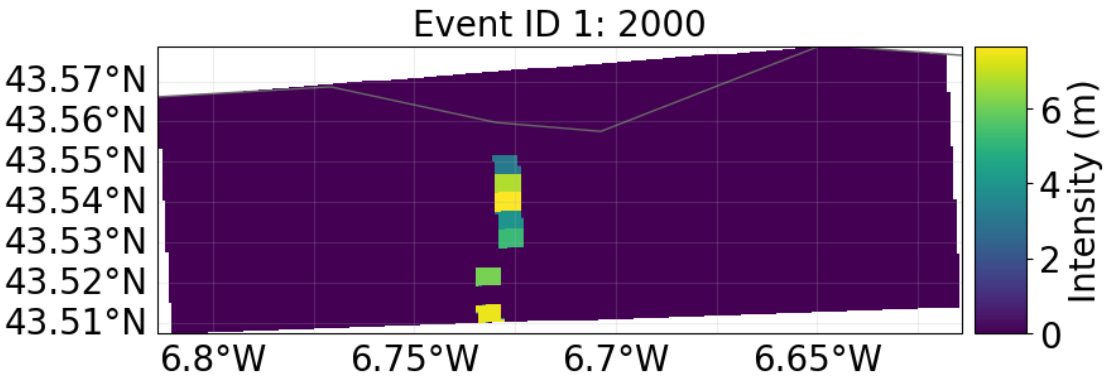
[16]:
```
'1/year'
```

[17]:
```
# plot centroids:
rf.centroids.plot()
```

[17]:
```
<GeoAxesSubplot:>
```



[18]:
```
# plotting intensity (Flood depth in m)
rf.plot_intensity(event=1, smooth = False);
```



1.6 Part 3: Defining exposures from Global Power Plant Database

The exposure datset is taken from Global Power Plant Database, which is a global and open source database of power plants.

The capacity_mw variable is the electrical generating capacity in megawatts and can be used as a proxy for the plant value.

The lat-lon coordinates are in the variables latitude and longitude.

[19]:
```
exp_df_name = 'global_power_plant_database_v_1_3/global_power_plant_database.
 ↪csv'
exp_df = pd.read_csv(os.path.join(os.getcwd(), exp_df_name))
cols = ['latitude','longitude','capacity_mw']
exp_df = exp_df[exp_df.country == 'ESP'][cols]
exp_df.head()
```

[19]:

	latitude	longitude	capacity_mw
20135	43.5528	-5.7231	877.660
20136	39.9427	-3.8548	758.740
20137	41.5485	0.8245	15.000
20138	41.8559	-1.9224	18.000
20139	41.8559	-1.9224	16.334

[20]:
```
exp_df.rename(columns = {'capacity_mw':'value'}, inplace =True)
exp_df.reset_index(inplace=True)
```

1.6.1 Remove Outliers in the data

Surprisingly latitude outliers are found for Spain case.

[21]:
```
exp_df = exp_df[exp_df.latitude > 25]
```

1.6.2 The asset type

The asset type defines what Huizinga damage functions look like (see here. The power plants are industrial type but as this damage functions are not inside climada (only residential) we well use residential as an example. We will try to talk to Climada developers to add all of them on our own and upload them to climada repository.

Damage functions specific for Power Plants could be used also (see here).

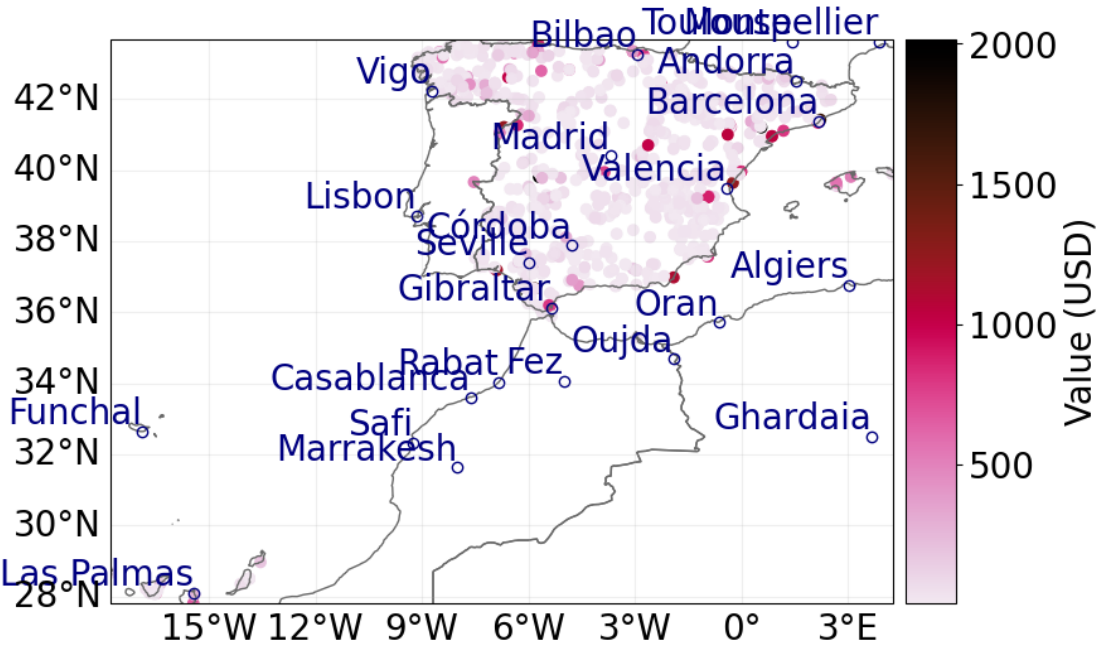
[22]:
```
inmp_RF = 3
exp_df['impf_RF'] = np.ones(exp_df.shape[0], int) * inmp_RF
```

[23]:
```
from climada.entity import Exposures

exp = Exposures(exp_df)
print('\n\x1b[1;03;30;30m' + 'exp has the type:', str(type(exp)))
print('and contains a GeoDataFrame exp.gdf:', str(type(exp.gdf)) +␣
 ↪'\n\n\x1b[0m')

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

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

print(exp)
```

```
exp has the type: <class 'climada.entity.exposures.base.Exposures'>

and contains a GeoDataFrame exp.gdf: <class

'geopandas.geodataframe.GeoDataFrame'>



check method logs:
tag:  File:
 Description:
ref_year: 2018
value_unit: USD
meta: {'crs': 'EPSG:4326'}
crs: EPSG:4326
data:
     index  latitude  longitude    value  impf_RF                    geometry
0    20135   43.5528    -5.7231  877.660        3  POINT (-5.72310 43.55280)
```

```
1     20136   39.9427    -3.8548  758.740        3  POINT (-3.85480 39.94270)
2     20137   41.5485     0.8245   15.000        3   POINT (0.82450 41.54850)
3     20138   41.8559    -1.9224   18.000        3  POINT (-1.92240 41.85590)
4     20139   41.8559    -1.9224   16.334        3  POINT (-1.92240 41.85590)
..      …       …          …       …             …                      …
824   20959   41.6428    -4.8151    9.000        3  POINT (-4.81510 41.64280)
825   20960   39.8710    -6.8320    7.600        3  POINT (-6.83200 39.87100)
826   20961   41.9340    -0.8050    9.000        3  POINT (-0.80500 41.93400)
827   20962   41.9440    -0.8180    9.900        3  POINT (-0.81800 41.94400)
828   20963   38.9420    -5.8750    7.000        3  POINT (-5.87500 38.94200)

[825 rows x 6 columns]
```

[24]:
```
from matplotlib import colors
norm=colors.LogNorm(vmin=1.0e2, vmax=1.0e10)
exp.plot_scatter();
```



1.7 Part 4: Setting JRC damage functions

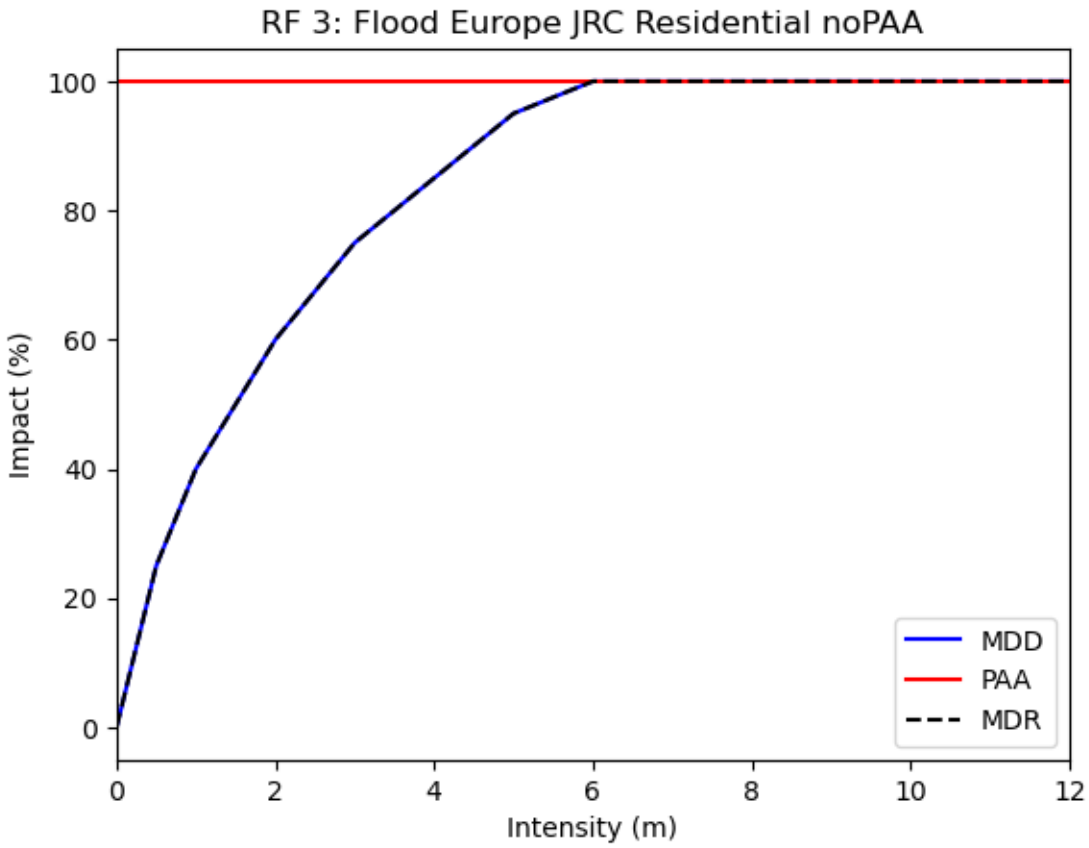
In CLIMADA we currently calculate damage by translating flood-depth into a damage factors. Damage assessments implemented in CLIMADA base on the residential damage functions basing on an empirical estimate published in the JRC report. Individual damage functions are available for six continents:

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

For further information on depth-damage functions see here.

[25]:
```
# import impact function set for RiverFlood using JRC damage functions () for 6␣
 ↪regions
from climada_petals.entity.impact_funcs.river_flood import␣
 ↪ImpfRiverFlood,flood_imp_func_set
impf_set = flood_imp_func_set()
impf_EUR = impf_set.get_func(fun_id=3)
impf_EUR[0].plot()
```

[25]:
```
<AxesSubplot:title={'center':'RF 3: Flood Europe JRC Residential noPAA'},
xlabel='Intensity (m)', ylabel='Impact (%)'>
```



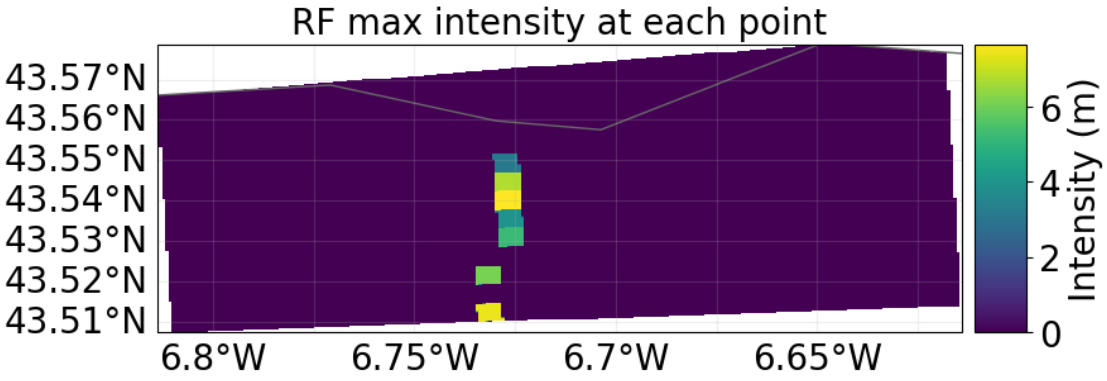
The plots illustrate how flood-depth is translated into a damage factor (0%-100%). The damage factor is then multiplied with the exposed asset in each grid cell to derive a local damage.

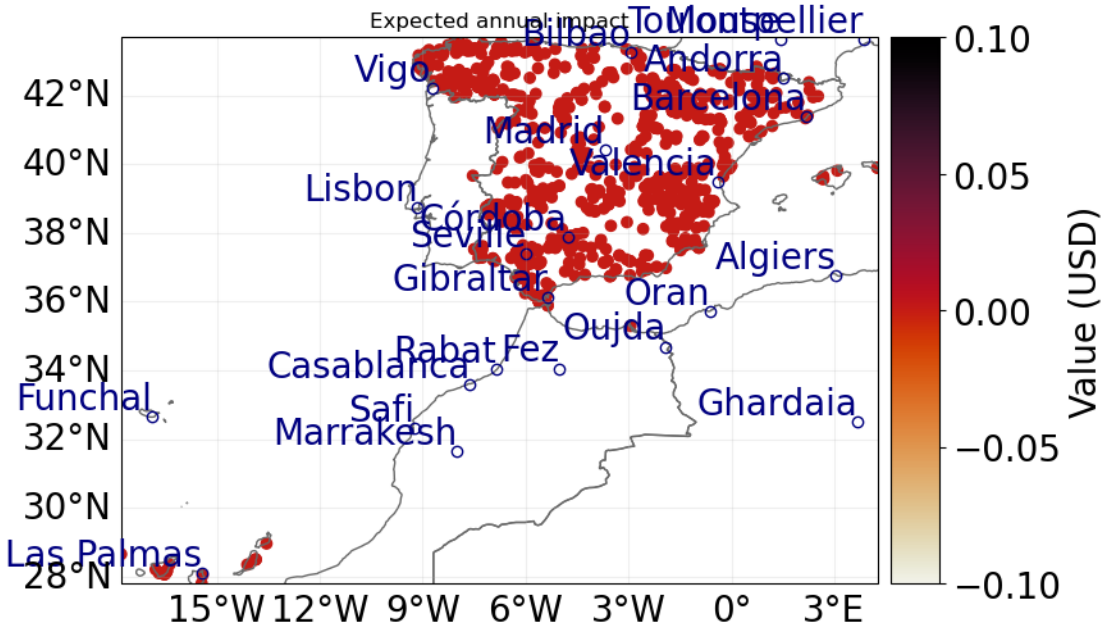
Plot leyend:
mdd:= Mean damage (impact) degree for each intensity (numbers in [0,1]).
paa:= Percentage of affected assets (exposures) for each intensity (numbers in [0,1]).
mdr:= The mean damage ratio (mdr=mdd*paa).

1.8 Part 5: Deriving flood economic-impact

[26]:
```
from climada.engine import Impact
imp=Impact()
imp.calc(exp, impf_set,rf,save_mat=False)
rf.plot_intensity(0)
imp.plot_scatter_eai_exposure();
```

```
2023-05-24 14:52:44,579 - climada.engine.impact - WARNING - The use of
Impact().calc() is deprecated. Use ImpactCalc().impact() instead.
2023-05-24 14:52:44,604 - climada.util.coordinates - WARNING - Distance to
closest centroid is greater than 100km for 665 coordinates.
```





[27]:
```
from climada.engine import Impact
imp=Impact()
imp.calc(exp, impf_set,rf,save_mat=False)
```

```
2023-05-24 14:52:48,749 - climada.engine.impact - WARNING - The use of
Impact().calc() is deprecated. Use ImpactCalc().impact() instead.
2023-05-24 14:52:48,772 - climada.util.coordinates - WARNING - Distance to
closest centroid is greater than 100km for 665 coordinates.
```

1.9 Part 6: Risk Measures available at Climada

Total loss at each event

AAI: Average Agregrated Impact. That is, the impact at event by its frequency all sum up.

EAL: Expected Annual Loss per asset.

[28]:
```
print('Total loss at each event:')
imp.at_event
```

```
Total loss at each event:
```

[28]:
```
array([0.])
```

1.9.1 AAI: Average Agregrated Impact. That is, the impact at event by its frequency all sum up.dd

[29]:
```
print('Average Agregated Impact:')
imp.aai_agg
```

```
Average Agregated Impact:
```

[29]:
```
0.0
```

[30]:
```
# Check AAI computing is correct
sum(imp.at_event*rf.frequency) == imp.aai_agg
```

[30]:
```
True
```

[31]:
```
print('Expected Annual Loss:')
EAL_df = pd.DataFrame(imp.eai_exp, columns = ['EAL (€)'])
EAL_df.head(5)
```

```
Expected Annual Loss:
```

[31]:

	EAL (€)
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

1.9.2 How is the impact derived ?

First of all, the use of Impact().calc() is deprecated. Use ImpactCalc().impact() instead.

The variable gdf inside impcalc.exposures object allocates the intersection between the exposures dataframe and the centroid reference id (column centr_RF), resulting a geopandas dataframe. From this and the hazard data the physical risk can be computed.

[32]:
```
from climada.engine.impact_calc import ImpactCalc
impcalc = ImpactCalc(exp, impf_set, rf)
im = impcalc.impact(save_mat=False, assign_centroids=True)
```

```
2023-05-24 14:52:48,881 - climada.util.coordinates - WARNING - Distance to
closest centroid is greater than 100km for 665 coordinates.
```

[33]:
```
exp_gdf = impcalc.exposures.gdf
exp_gdf.head(5)
```

[33]:

	index	latitude	longitude	value	impf_RF	geometry	\
0	20135	43.5528	-5.7231	877.660	3	POINT (-5.72310 43.55280)	
1	20136	39.9427	-3.8548	758.740	3	POINT (-3.85480 39.94270)	
2	20137	41.5485	0.8245	15.000	3	POINT (0.82450 41.54850)	
3	20138	41.8559	-1.9224	18.000	3	POINT (-1.92240 41.85590)	
4	20139	41.8559	-1.9224	16.334	3	POINT (-1.92240 41.85590)	

	centr_RF
0	46293
1	-1
2	-1
3	-1
4	-1

1.9.3 Build Hazard DataFrame

For that we join fraction, intensity and centroid coordinates.

[34]:
```
frc_cols = ['FR' + str(i) for i in range(1, rf.size+1)]
haz_fraction = pd.DataFrame(impcalc.hazard.fraction.toarray().transpose(),␣
 ↪columns=frc_cols)

dph_cols = ['I' + str(i) for i in range(1, rf.size+1)]
haz_intensity = pd.DataFrame(impcalc.hazard.intensity.toarray().transpose(),␣
 ↪columns=dph_cols)

lat_lon = pd.DataFrame(impcalc.hazard.centroids.coord, columns = ['lat', 'lon'])

haz_df = pd.concat([haz_intensity, haz_fraction, lat_lon], axis = 1)
haz_df.head(5)
```

[34]:

	I1	FR1	lat	lon
0	0.0	1.0	43.56587	-6.81379
1	0.0	1.0	43.56589	-6.81316
2	0.0	1.0	43.56591	-6.81253
3	0.0	1.0	43.56593	-6.81190
4	0.0	1.0	43.56595	-6.81127

1.9.4 Merge Hazard DataFrame and Exposure DataFrame on centroid reference

[35]:
```
exp_gdf['EAL (€)'] = EAL_df['EAL (€)'].values
exp_gdf_ = exp_gdf[exp_gdf.centr_RF != -1]
haz_df_ = haz_df.iloc[exp_gdf_.centr_RF,:]

haz_df_['asset_value'] = exp_gdf_.value.values
haz_df_['EAL (€)'] = exp_gdf_['EAL (€)'].values
haz_df_.index.name = 'Centroid ID'
haz_df_.reset_index(inplace=True)
haz_df_.head(5)
```

[35]:

	Centroid ID	I1	FR1	lat	lon	asset_value	EAL (€)
0	46293	0.0	1.0	43.51383	-6.61418	877.66	0.0
1	46293	0.0	1.0	43.51383	-6.61418	346.84	0.0
2	24564	0.0	1.0	43.51085	-6.70709	56.04	0.0
3	21416	0.0	1.0	43.53534	-6.71937	24.00	0.0
4	46293	0.0	1.0	43.51383	-6.61418	854.17	0.0

1.9.5 Save Results

After merging hazard data, exposures data and risk measures we save it tocompare between datasets.

[36]:
```
haz_df_
```

[36]:

	Centroid ID	I1	FR1	lat	lon	asset_value	EAL (€)
0	46293	0.0	1.0	43.51383	-6.61418	877.660	0.0
1	46293	0.0	1.0	43.51383	-6.61418	346.840	0.0
2	24564	0.0	1.0	43.51085	-6.70709	56.040	0.0
3	21416	0.0	1.0	43.53534	-6.71937	24.000	0.0
4	46293	0.0	1.0	43.51383	-6.61418	854.170	0.0
5	1529	0.0	1.0	43.50746	-6.81010	61.940	0.0
6	1529	0.0	1.0	43.50746	-6.81010	14.940	0.0
7	46293	0.0	1.0	43.51383	-6.61418	82.460	0.0
8	46293	0.0	1.0	43.51383	-6.61418	553.640	0.0
9	1529	0.0	1.0	43.50746	-6.81010	20.460	0.0
10	42951	0.0	1.0	43.51335	-6.62934	38.940	0.0
11	1529	0.0	1.0	43.50746	-6.81010	12.750	0.0
12	1529	0.0	1.0	43.50746	-6.81010	27.000	0.0
13	46293	0.0	1.0	43.51383	-6.61418	34.850	0.0

14	1529	0.0	1.0	43.50746	-6.81010	38.000	0.0
15	1529	0.0	1.0	43.50746	-6.81010	20.000	0.0
16	46293	0.0	1.0	43.51383	-6.61418	24.420	0.0
17	1529	0.0	1.0	43.50746	-6.81010	49.300	0.0
18	46293	0.0	1.0	43.51383	-6.61418	30.600	0.0
19	46293	0.0	1.0	43.51383	-6.61418	44.000	0.0
20	1529	0.0	1.0	43.50746	-6.81010	17.820	0.0
21	1529	0.0	1.0	43.50746	-6.81010	18.801	0.0
22	1529	0.0	1.0	43.50746	-6.81010	19.500	0.0
23	29729	0.0	1.0	43.51669	-6.68337	49.300	0.0
24	1529	0.0	1.0	43.50746	-6.81010	27.200	0.0
25	46293	0.0	1.0	43.51383	-6.61418	49.540	0.0
26	1529	0.0	1.0	43.50746	-6.81010	157.240	0.0
27	1199	0.0	1.0	43.52033	-6.81092	53.150	0.0
28	46293	0.0	1.0	43.51383	-6.61418	346.250	0.0

[37]:
```
haz_df_.to_excel('gov_10rp.xlsx')
```