flood_example_spain_ISIMIP2b_Power

May 24, 2023

1 Hazard RiverFlood Example for Spain Using ISIMIP2b Data

1.1 Summary

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

1.2 Links

Find a full tutorial of climada platform here.

A similar example for Germany and switzerland can be found here.

1.3 How is this tutorial structured?

```
Part 1: Data availability and use

Part 2: Generating a RiverFlood Hazard

Part 3: Calculating Flooded Area

Part 4: Defining exposures from Global Power Plant Database

Part 5: Setting JRC damage functions

Part 6: Deriving flood economic-impact

Part 7: Risk Measures available at Climada
```

1.4 Part 1: Data availability and Use

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

The input data can be found at ISMIP Archive.

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

1.5 ISIMIP2a

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

<indicator_resolution_GHM_ClimateForcingDataset_ProtectionStandard.nc>

1.6 ISIMIP2b

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

<indicator_resolution_GHM_GCM_ProtectionStandard.nc>

1.7 Part 2: Generating a RiverFlood Hazard

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

Countries or regions can either be set with corresponding ISIMIPNatID centroids (ISINatIDGrid = True) or with Natural Earth Multipolygons (default). It is obligatory to set paths for flood depth and flood fraction, here we present example files from floods for the years 2030-2040.

1.7.1 Setting floods for countries with Natural Earth Multipolygons:

[1]:
```
# Dont show warnings
import warnings
from shapely.errors import ShapelyDeprecationWarning
warnings.filterwarnings("ignore")

# Import External Libraries
import os
import numpy as np
import pandas as pd
from pathlib import WindowsPath
import matplotlib.pyplot as plt

# Import Climada Library
from climada_petals.hazard.river_flood import RiverFlood
from climada.hazard.centroids import Centroids
```

```python
from climada_petals.util.constants import HAZ_DEMO_FLDDPH, HAZ_DEMO_FLDFRC
```

[2]:
```python
# https://files.isimip.org/cama-flood/
# indicator_resolution_GHM_ClimateForcingDataset_ProtectionStandard.nc
dhp_filename = 'flddph_150arcsec_clm45_gfdl-esm2m_0.nc'
frc_filename = 'fldfrc_150arcsec_clm45_gfdl-esm2m_0.nc'

ISIMIP2b_dph_path = WindowsPath(os.path.join(os.getcwd(), 'isimip_flood_data/
↪2b', dhp_filename))
ISIMIP2b_frc_path = WindowsPath(os.path.join(os.getcwd(), 'isimip_flood_data/
↪2b', frc_filename))

countries_ = ['ESP'] # https://www.iban.com/country-codes
years_ = list(range(2030, 2041))

# generating RiverFlood hazard from netCDF file
# uses centroids from Natural Earth Multipolygon for Spain
rf = RiverFlood.from_nc(countries = countries_, years=years_,␣
↪dph_path=ISIMIP2b_dph_path, frc_path=ISIMIP2b_frc_path)
rf.event_name
```

```
2023-05-24 14:55:02,447 - climada.hazard.base - WARNING - The use of
Hazard.set_raster is deprecated.Use Hazard.from_raster instead.
```

[2]:
```
['2030',
 '2031',
 '2032',
 '2033',
 '2034',
 '2035',
 '2036',
 '2037',
 '2038',
 '2039',
 '2040']
```

1.7.2 How is the event frequency defined?

To define a Climada Hazard Class there are three mandatory variables that are later used to compute the risk measures: event frequency, intensisty and fraction.

In the cell above the RiverFlood class is created given the intensity and the fraction. So, how is the frequency computed?

The frequency is computed autimatically and distributed uniformly between events. That means that if I select 3 years (1 event per year) then the frequency is 1/3. The frequency unit is 1/year (1 year return period).

[3]:
```python
rf.frequency
```

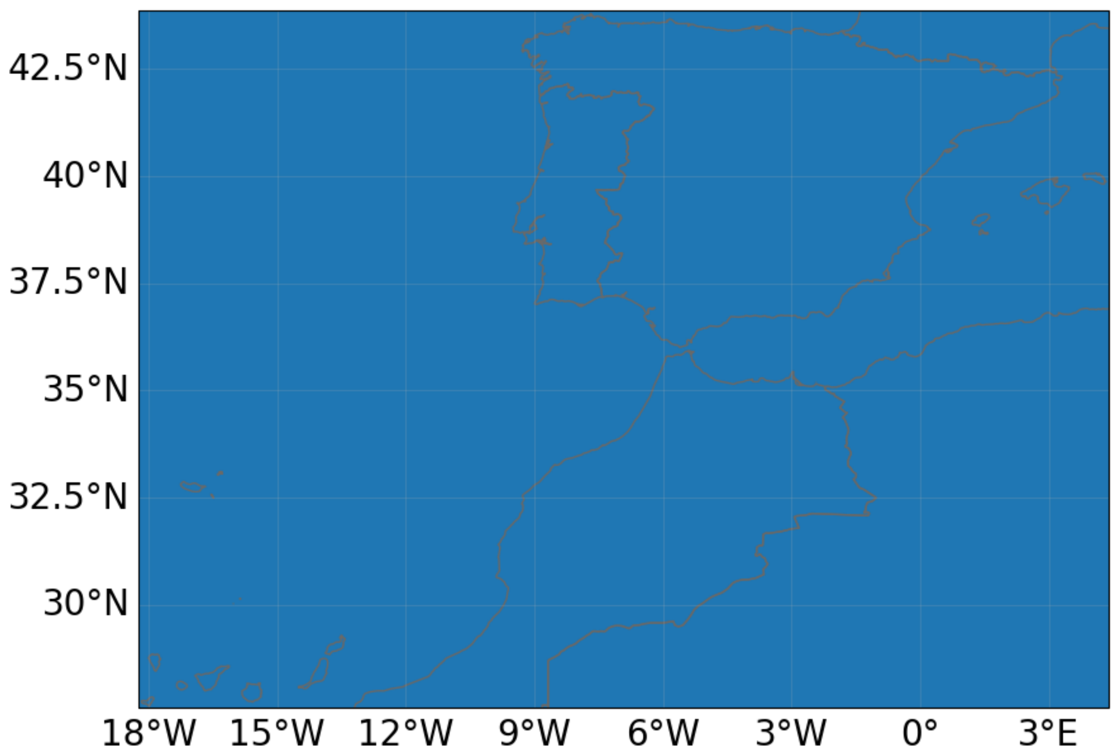
[3]: array([0.09090909, 0.09090909, 0.09090909, 0.09090909, 0.09090909,
       0.09090909, 0.09090909, 0.09090909, 0.09090909, 0.09090909,
       0.09090909])

[4]:
```
rf.frequency_unit
```

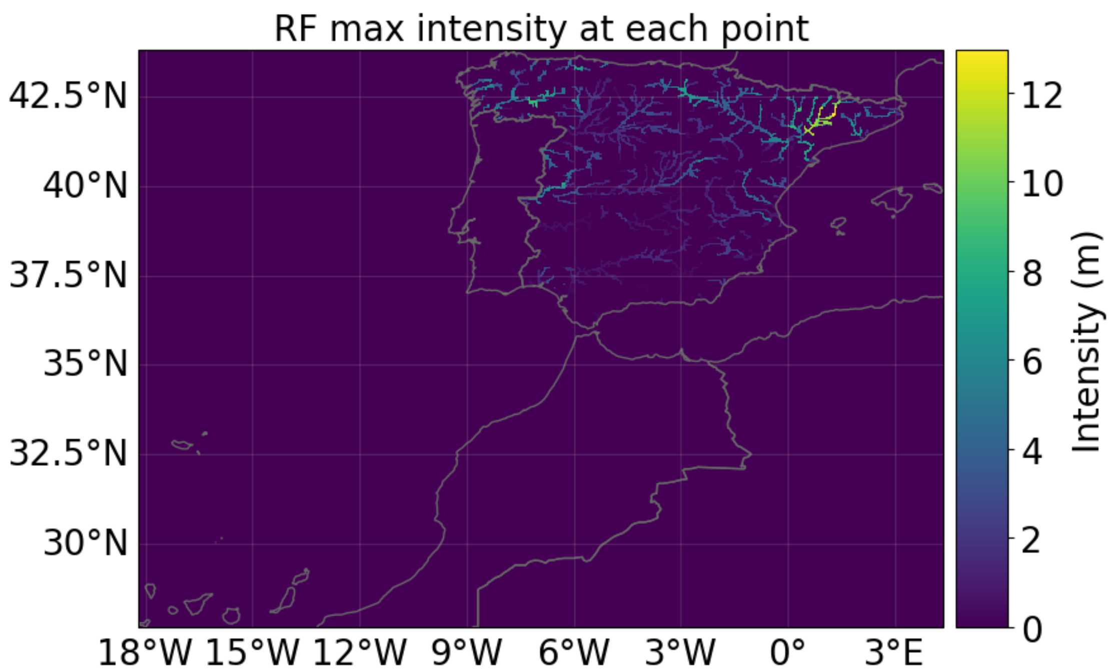
[4]: '1/year'

[5]:
```
# Note: Points outside the selected countries are masked in further analysis.
# plot centroids:
rf.centroids.plot()
# get resolution
print('resolution:', rf.centroids.meta['transform'][0])
```

resolution: 0.041666666666666667



[6]:
```
# plotting intensity (Flood depth in m)
rf.plot_intensity(event=0, smooth = False);
```



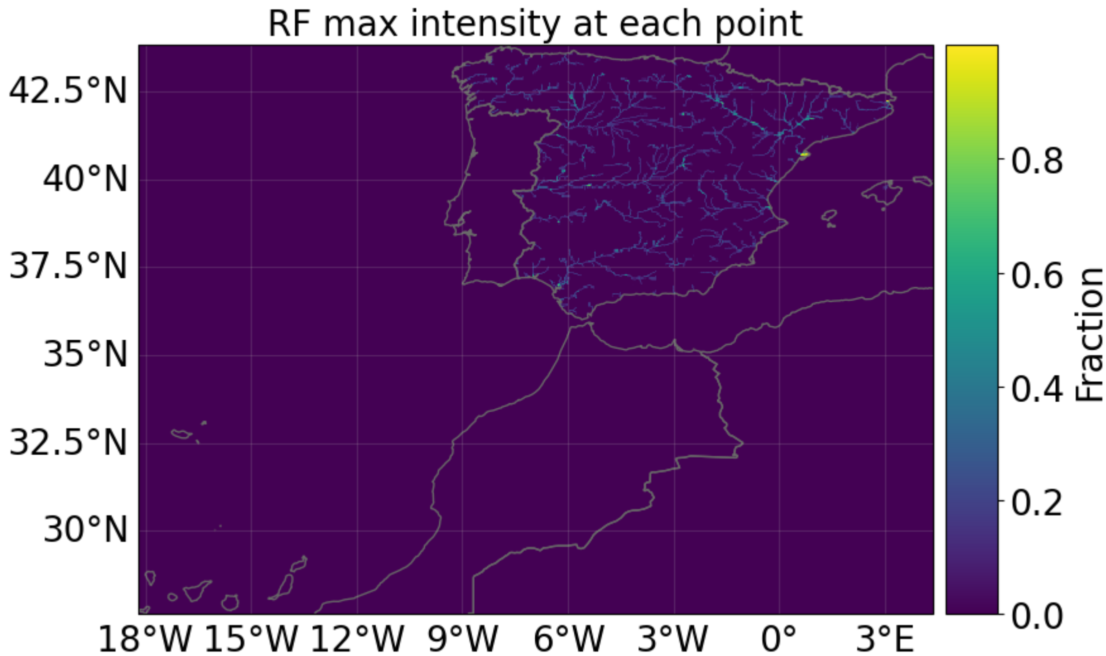
1.8 Part 3: Calculating Flooded Area

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

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

[7]:
```
rf.plot_fraction(event=0, smooth = False)
# calculating flooded area
rf.set_flooded_area()
print("Total flooded area for year " + str(years_[0]) + " in Spain:")
print(str(rf.fla_annual[0]) + " m2")

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

```
Total flooded area for year 2030 in Spain:
9675730177.295242 m2
Total flooded area at first event in Spain:
9675730177.295242 m2
```



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

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

[8]:
```
array([1110868.02198607, 7776076.15390228,   35694.49497231, …,
       1330806.54442225,   83175.40902639,  918050.91685855])
```

1.9 Part 4: 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.

[9]:
```
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()
```

[9]:
```
       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
```

[10]:
```python
exp_df.rename(columns = {'capacity_mw':'value'}, inplace =True)
exp_df.reset_index(inplace=True)
```

1.9.1 Remove Outliers in the data

Surprisingly latitude outliers are found for Spain case.

[11]:
```python
exp_df = exp_df[exp_df.latitude > 25]
```

1.9.2 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).

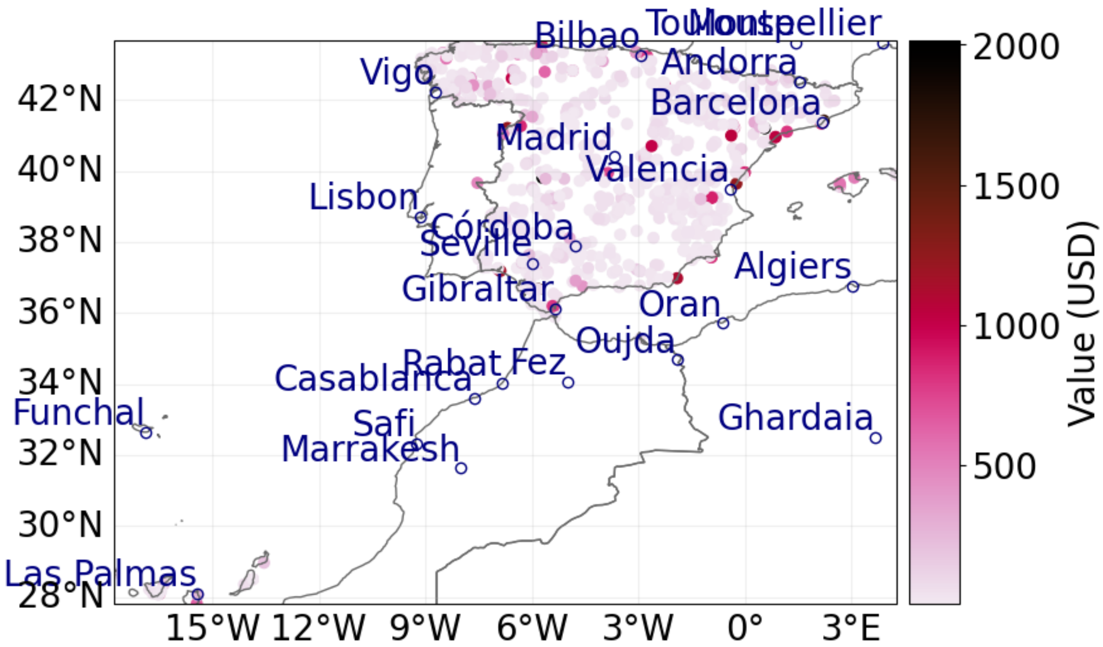
[12]:
```python
inmp_RF = 3
exp_df['impf_RF'] = np.ones(exp_df.shape[0], int) * inmp_RF
```

[13]:
```python
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]
```

[14]:
```
from matplotlib import colors
norm=colors.LogNorm(vmin=1.0e2, vmax=1.0e10)
exp.plot_scatter();
```



1.10 Part 5: Setting JRC damage functions

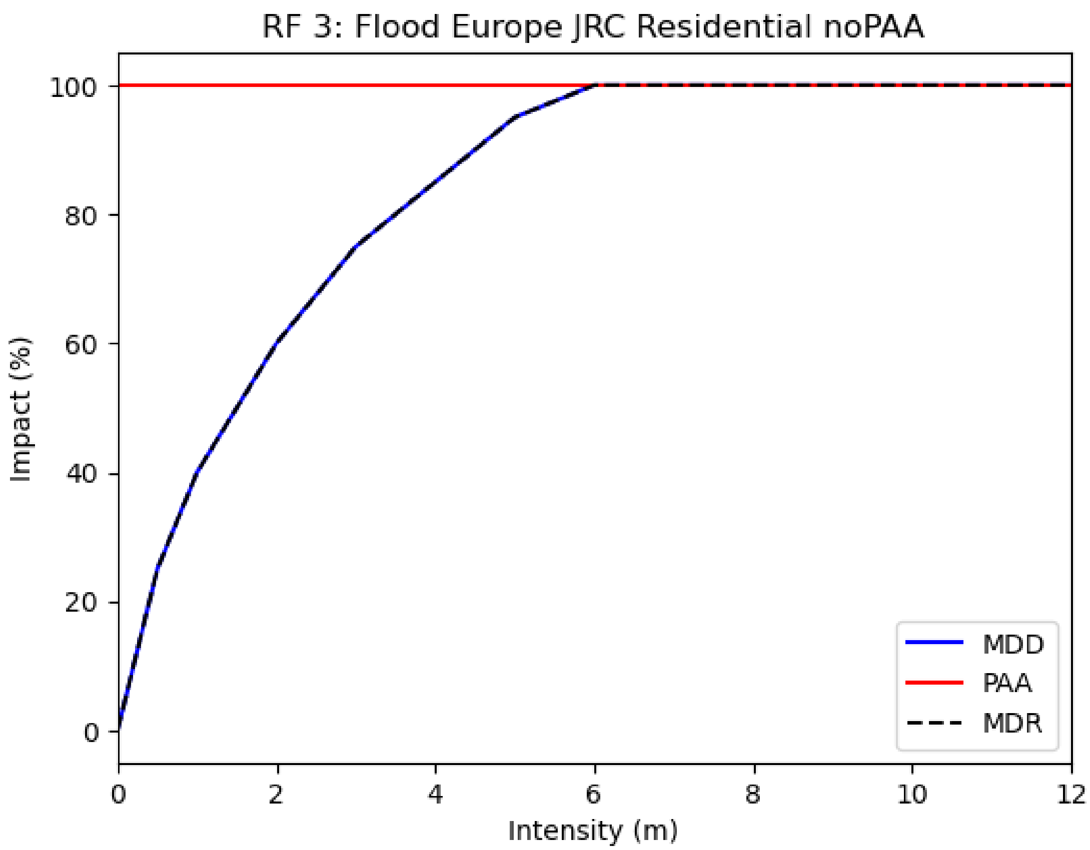
In CLIMADA we currently calculate damage by translating flood-depth into a damage factors. Damage assessments implemented in CLIMADA base on the residential damage functions basing on an empirical estimate published in the JRC report. Individual damage functions are available for six continents:

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

For further information on depth-damage functions see here. Also see ImpactFunc class of Climada.

[15]:
```
# 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()
```

[15]:
```
<AxesSubplot:title={'center':'RF 3: Flood Europe JRC Residential noPAA'},
xlabel='Intensity (m)', ylabel='Impact (%)'>
```



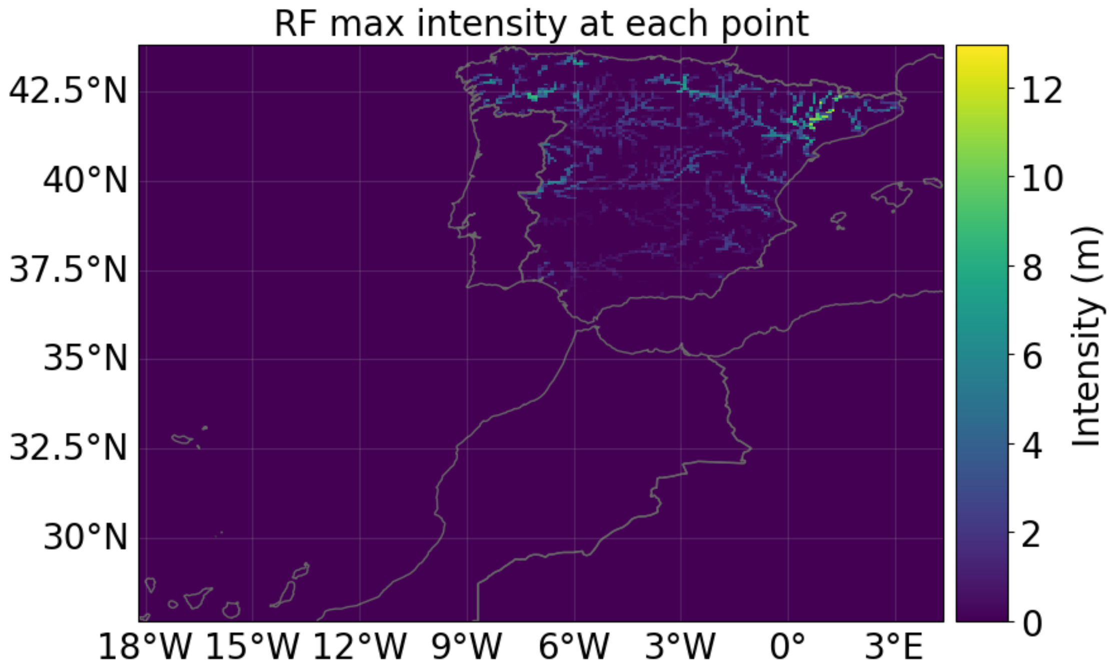
The plots illustrate how flood-depth is translated into a damage factor (0%-100%). The damage factor is then multiplied with the exposed asset in each grid cell to derive a local damage.

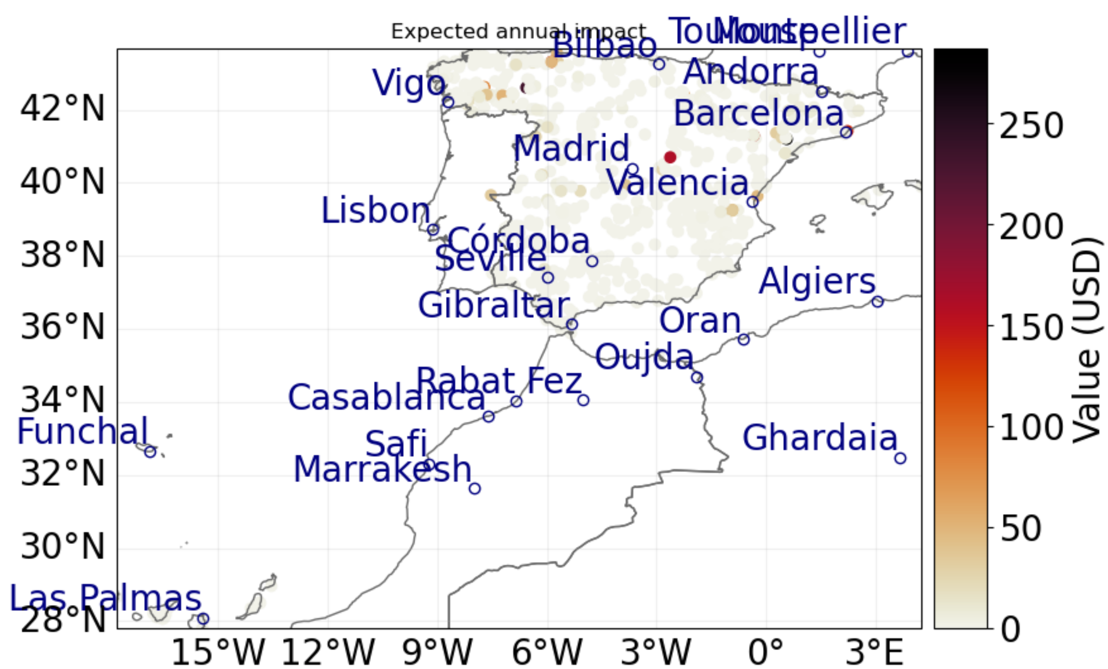
Plot leyend:
mdd:= Mean damage (impact) degree for each intensity (numbers in [0,1]).
paa:= Percentage of affected assets (exposures) for each intensity (numbers in [0,1]).
mdr:= The mean damage ratio (mdr=mdd*paa).

1.11 Part 6: Deriving flood economic-impact

[16]:
```
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 14:56:33,976 - climada.engine.impact - WARNING - The use of
Impact().calc() is deprecated. Use ImpactCalc().impact() instead.
```





[17]:
```python
from climada.engine import Impact
imp=Impact()
imp.calc(exp, impf_set,rf,save_mat=False)
```

```
2023-05-24 14:56:40,857 - climada.engine.impact - WARNING - The use of
Impact().calc() is deprecated. Use ImpactCalc().impact() instead.
```

1.12 Part 7: 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.

[18]:
```python
print('Total loss at each event:')
imp.at_event
```

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

[18]:
```
array([2312.69247553, 2968.65226467, 3083.82024697, 2249.64933126,
       2570.56985619, 2259.29901882, 2427.35426562, 2367.06837902,
       2565.63135751, 3077.29655163, 3262.59891522])
```

[19]:
```python
print('Average Agregated Impact:')
imp.aai_agg
```

```
Average Agregated Impact:
```

[19]:
```
2649.512060220551
```

[20]:
```python
# Check AAI computing is correct
sum(imp.at_event*rf.frequency) == imp.aai_agg
```

[20]:
```
False
```

[21]:
```python
print('Expected Annual Loss:')
EAL_df = pd.DataFrame(imp.eai_exp, columns = ['EAL (€)'])
EAL_df.head(5)
```

```
Expected Annual Loss:
```

[21]:

	EAL (€)
0	48.190241
1	37.278127
2	0.000000
3	0.000000
4	0.000000

1.12.1 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.

[22]:
```python
from climada.engine.impact_calc import ImpactCalc
impcalc = ImpactCalc(exp, impf_set, rf)
im = impcalc.impact(save_mat=False, assign_centroids=True)
```

[23]:
```python
exp_gdf = impcalc.exposures.gdf
exp_gdf.head(5)
```

[23]:
```
   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      3551
1     50750
2     29724
3     25864
4     25864
```

1.12.2 Build Hazard DataFrame

For that we join fraction, intensity and centroid coordinates.

[24]:
```python
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)
```

[24]:
```
    I1   I2   I3   I4   I5   I6   I7   I8   I9  I10  …  FR4  FR5  FR6  FR7  \
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0

   FR8  FR9  FR10  FR11      lat        lon
0  0.0  0.0   0.0   0.0  43.8125 -18.187500
1  0.0  0.0   0.0   0.0  43.8125 -18.145833
```

```
2  0.0  0.0   0.0   0.0  43.8125 -18.104167
3  0.0  0.0   0.0   0.0  43.8125 -18.062500
4  0.0  0.0   0.0   0.0  43.8125 -18.020833

[5 rows x 24 columns]
```

1.12.3 Merge Hazard DataFrame and Exposure DataFrame on centroid reference

[25]:

```python
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)
```

[25]:

```
   Centroid ID        I1        I2        I3        I4        I5        I6  \
0         3551  1.055622  1.336143  1.810507  0.938985  1.463692  1.340045
1        50750  0.835546  1.424117  0.801128  0.846222  0.792113  0.796046
2        29724  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
3        25864  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
4        25864  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

         I7        I8        I9  …       FR6      FR7       FR8      FR9  \
0  0.914419  1.165091  0.940650  …  0.113208  0.08805  0.113208  0.08805
1  0.906532  1.068029  0.775983  …  0.132800  0.13280  0.132800  0.13280
2  0.000000  0.000000  0.000000  …  0.000000  0.00000  0.000000  0.00000
3  0.000000  0.000000  0.000000  …  0.000000  0.00000  0.000000  0.00000
4  0.000000  0.000000  0.000000  …  0.000000  0.00000  0.000000  0.00000

       FR10      FR11        lat       lon  asset_value    EAL (€)
0  0.141509  0.141509  43.562500 -5.729167      877.660  48.190241
1  0.132800  0.132800  39.937500 -3.854167      758.740  37.278127
2  0.000000  0.000000  41.562500  0.812500       15.000   0.000000
3  0.000000  0.000000  41.854167 -1.937500       18.000   0.000000
4  0.000000  0.000000  41.854167 -1.937500       16.334   0.000000

[5 rows x 27 columns]
```

1.12.4 Save Results

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

[26]:

```python
haz_df_
```

[26]:

```
    Centroid ID        I1        I2        I3        I4        I5        I6  \
0          3551  1.055622  1.336143  1.810507  0.938985  1.463692  1.340045
1         50750  0.835546  1.424117  0.801128  0.846222  0.792113  0.796046
2         29724  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
3         25864  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
4         25864  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
..          …         …         …         …         …         …         …
820       28505  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
821       51763  1.972173  3.341146  4.804051  1.863211  5.799756  2.813602
822       24807  0.000000  1.697076  0.988051  0.408080  0.573607  0.000000
823       24807  0.000000  1.697076  0.988051  0.408080  0.573607  0.000000
824       63709  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

           I7        I8        I9  …       FR6      FR7       FR8      FR9  \
0    0.914419  1.165091  0.940650  …  0.113208  0.08805  0.113208  0.08805
1    0.906532  1.068029  0.775983  …  0.132800  0.13280  0.132800  0.13280
2    0.000000  0.000000  0.000000  …  0.000000  0.00000  0.000000  0.00000
3    0.000000  0.000000  0.000000  …  0.000000  0.00000  0.000000  0.00000
4    0.000000  0.000000  0.000000  …  0.000000  0.00000  0.000000  0.00000
..        …         …         …  …       …        …        …        …
820  0.000000  0.000000  0.000000  …  0.000000  0.00000  0.000000  0.00000
821  2.194903  1.906710  2.476631  …  0.160000  0.16000  0.160000  0.16000
822  0.025261  3.313138  0.029203  …  0.000000  0.00640  0.006400  0.00640
823  0.025261  3.313138  0.029203  …  0.000000  0.00640  0.006400  0.00640
824  0.000000  0.000000  0.000000  …  0.000000  0.00000  0.000000  0.00000

         FR10      FR11        lat       lon  asset_value   EAL (€)
0    0.141509  0.141509  43.562500 -5.729167      877.660  48.190241
1    0.132800  0.132800  39.937500 -3.854167      758.740  37.278127
2    0.000000  0.000000  41.562500  0.812500       15.000   0.000000
3    0.000000  0.000000  41.854167 -1.937500       18.000   0.000000
4    0.000000  0.000000  41.854167 -1.937500       16.334   0.000000
..        …         …          …        …          …          …
820  0.000000  0.000000  41.645833 -4.812500        9.000   0.000000
821  0.160000  0.160000  39.854167 -6.812500        7.600   0.850946
822  0.006400  0.006400  41.937500 -0.812500        9.000   0.015219
823  0.006400  0.006400  41.937500 -0.812500        9.900   0.016741
824  0.000000  0.000000  38.937500 -5.895833        7.000   0.000000

[825 rows x 27 columns]
```

[27]:

```
haz_df_.to_excel('isimip2b.xlsx')
```