

# detect\_storms\_in\_model

Documentation

Marine De Carlo

2023

# Architecture of the folder

The **src** folder contains:

- detect\_storms\_in\_model.py
- linkStorms\_models.py
- detection\_code/
  - \_\_init\_\_.py
  - params\_detect.py
  - storms\_functions\_detect.py
  - storms\_functions\_tracking.py
  - storms\_functions\_io.py
  - storms\_functions\_geo.py

# Needed packages in environment

Packages to be installed:

- Matplotlib
- Pandas
- Numpy
- Scipy
- Xarray
- Netcdf4
- H5netcdf

To use the environment in jupyter :

- `'<whatever_snake> install ipykernel jupyter'`
- `python -m ipykernel install --user --name=<ENV_name> --display-name=<ENV_name_to_be_displayed>`

# Params\_detect.py

Params\_detect.py contains your parameters, both paths and detection criterion.

Path parameters:

- Nb\_CPU : nb of cpu that you want to allow for multiprocessing if any (limitation max to avoid memory issue)
- isWW3 (depending on how is the data saved =1 is monthly, =0 is daily)
- PATH : where are the input files stored (without year or month info)
- FORMAT\_IN\* : remaining of the path = part containing the temporal info + filename
- PATH\_SAVE\_detect, PATH\_SAVE\_tracking : Paths where to save the results of the code
- FORMAT\_OUT\_detect\*, FORMAT\_OUT\_tracking\* : file generic name for saving results
- filedist2coast: path to nc file containing the distance to coast grid.

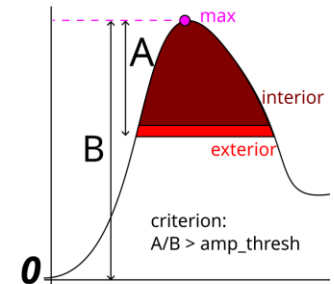
\*write generic name with YYYY and MM (and DD for input files when isWW3 ==0)

3 params to define Hs levels used for detection (see [slide 9](#)):

- swh\_crit\_max, dswh\_crit, min\_swh  
=> swh\_levels = np.arange(min\_swh, swh\_crit\_max + dswh\_crit, dswh\_crit)

Parameters for detection criterion:

- amp\_thresh : minimum value of the normalized prominence of the peak for a region to be detected as storm  
*default : 1./5*
- min\_area : minimum area for a region to be detected as storm, in km2  
*default : 500*
- area\_forgotten\_ratio : minimum ratio of area for a region to become a "forgotten" storm and be saved as a storm (see [slide 10](#))  
*default : 0.2*



Parameter for tracking:

- threshold\_dist : distance max for a storm between 2 timesteps, in km  
*default : 400*

# How to run

To run the code :

- 1) Activate your python environment
- 2) Go to the src folder
- 3) Check the params\_detect.py file  
(update with your data, folders etc)
- 4) Type the instruction (e.g.)

```
python detect_storms_in_model.py -s 0 -y 2023 -Y 2023 -m 1 -M 2
```

Amongst the options the '-s' (step) one is mandatory. Here are the different subroutine run for each step (for the difference between internal tracking and transition tracking, see [slide 12](#))

Step Number	0	1	2	111	110	11
Detection	✓			✓	✓	
Internal tracking		✓		✓	✓	✓
Transition tracking			✓	✓		✓

# How to run detect\_storms\_in\_model.py

```
python detect_storms_in_model.py -h
```

```
usage: detect_track_storms [-h] [-y YEAR] [-m MONTH] [-Y FINAL_YEAR] [-M FINAL_MONTH] [-s {0,1,2,11,111,110}] [-r] [-R]
                        [--multiprocessing]
```

options:

-h, --help show this help message and exit

-y YEAR, --year YEAR Chose the year for processing

-m MONTH, --month MONTH

Chose the month for processing

-Y FINAL\_YEAR, --final\_year FINAL\_YEAR

Chose the final year to take into account

-M FINAL\_MONTH, --final\_month FINAL\_MONTH

Chose the final month to take into account

-s {0,1,2,11,111,110}, --step {0,1,2,11,111,110}

Choose the step for applying the calculation process. 0: detect\_only | 1: file\_internal\_track\_only | 2: transition\_track\_only | 111 : all\_steps | 110: detect and file internal track steps (no transition between files track) | 11 : all\_tracking\_steps

-r, --reprocess Force a reprocessing for the detection step : existing files will be deleted and re computed

-R, --reprocess\_tracking

Force a reprocessing for the tracking steps : existing files will be deleted and re computed

--multiprocessing Use multiprocessing to run the code

# Function 'run\_detection' (step 0)

Loop over months (1 file for WW3 or concatenation of days for ERA5),

For each month, there is a loop over time steps: either using the python package multiprocessing or using a classical for-loop (depending on the '--multiprocessing' option of the code)

For each time step the function 'get\_storm\_by\_timestep' is applied.

The results are then appended, concatenated and saved into 1 dataset per month.

=> The Path and name format for saving these dataset are defined in params\_detect.py as 'PATH\_SAVE\_detect' and 'FORMAT\_OUT\_detect'.

The final filename is `os.path.join(PATH_SAVE_detect, FORMAT_OUT_detect)`

# Function 'get\_storm\_by\_timestep'

Inputs :

- **ds0** : xarray DataSet with coordinates 'longitude', 'latitude' and 'time'
- **levels** for Hs screening: np.array() in decreasing order (see [slide 9](#)). *Defined in params\_detect.py*
- **amp\_thresh** : relative amplitude threshold ratio criterion to select region as storm. *Defined in params\_detect.py*  
(linked to condition  $(\max(\text{interior}) - \min(\text{exterior})) / \max(\text{interior}) \geq \text{amp\_thresh}$ )
- **min\_area** : criterion for minimum size of a storm. *Defined in params\_detect.py*
- **area\_forgotten\_ratio**: criterion for saving a storm close to another one. *Defined in params\_detect.py*
- **plot\_output=False** : `#!/` returns the handles for the results figure
- **plot\_example=False**

Params have been removed but can still be of interest (*Defined in params\_detect.py*)

- **Npix\_min** : for criterion on number of pixels per selection (old version of `eddy_area_within_limits`)
- **cte.d\_thresh\_min, cte.d\_thresh\_max** => condition `is_large_enough` (criterion on 'shape').



# Function 'get\_storm\_by\_timestep'











Outputs :

- `_results` is a xarray dataset => the 'storms\_by\_t' coordinate is the number of storm detected by timestep the 'areastorm' is given in km<sup>2</sup>.
- If option **plot\_output** is True: the handle to the figure is also returned









xarray.Dataset

► Dimensions: (x: 12)

▼ Coordinates:

time	()	datetime64[ns]	2023-01-01		
latitude	(x)	float32	40.5 -48.5 -61.5 ... -61.5 63.5		
longitude	(x)	float32	-161.5 114.5 -139.5 ... 140.5 -11.0		
regions	(x)	float64	0.0 1.0 2.0 3.0 ... 9.0 10.0 11.0		
storms_by_t	(x)	int64	12 12 12 12 12 12 12 12 12 12 12 12		

▼ Data variables:

lon_max	(x)	float64	-161.5 114.5 -139.5 ... 140.5 -11.0		
lat_max	(x)	float32	40.5 -48.5 -61.5 ... -61.5 63.5		
hs_max	(x)	float32	9.398 7.966 7.474 ... 4.918 6.038		
areastorm	(x)	float32	2.659e+06 1.222e+06 ... 1.152e+06		

► Indexes: (0)

► Attributes: (0)

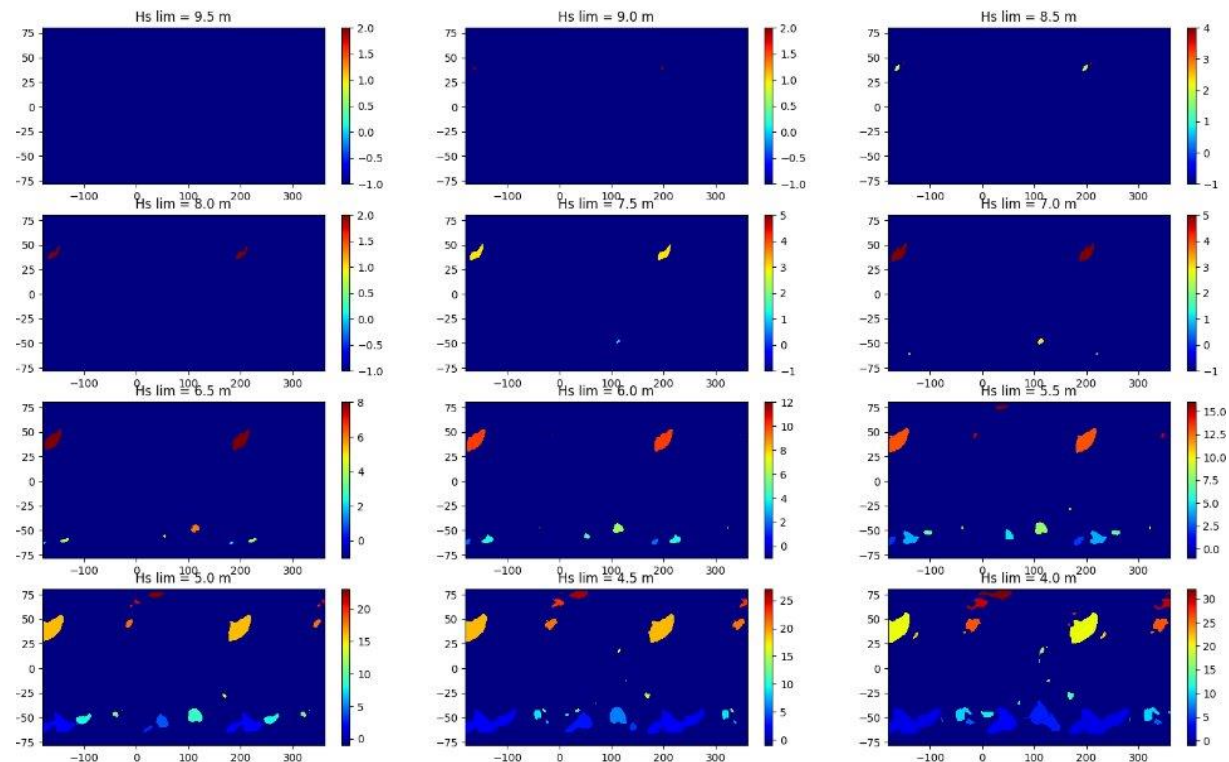
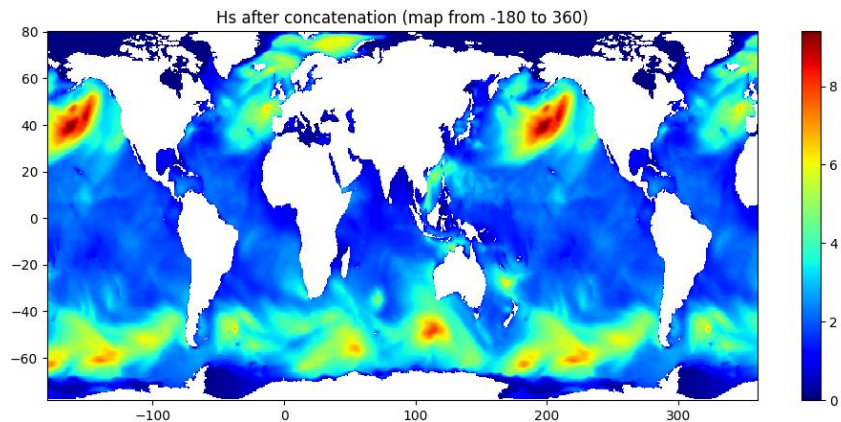
# Function 'get\_storm\_by\_timestep'

For each level in **levels** (in decreasing order)

- 'labellisation' of regions above threshold is done
- each region is studied: if criterions are matched  
=> storm is saved !

The Hs map is duplicated  
from lon = [-180;180] to lon=[-  
180;360]

=> to avoid detection issues  
around 180°



# Function 'get\_storm\_by\_timestep'

Criteria for saving storms:

- eddy\_area\_within\_limits : the area of the region must be  $> \text{min\_area}$
- has\_internal\_max: the maximum of the region must be properly inside and not on its edge
- is\_tall\_storm: the difference between the max of the region and the mean of its edge should be higher than  $\text{amp\_thresh} * \text{max}(\text{region})$

Special criterions exists to deal with problematic situations such as :

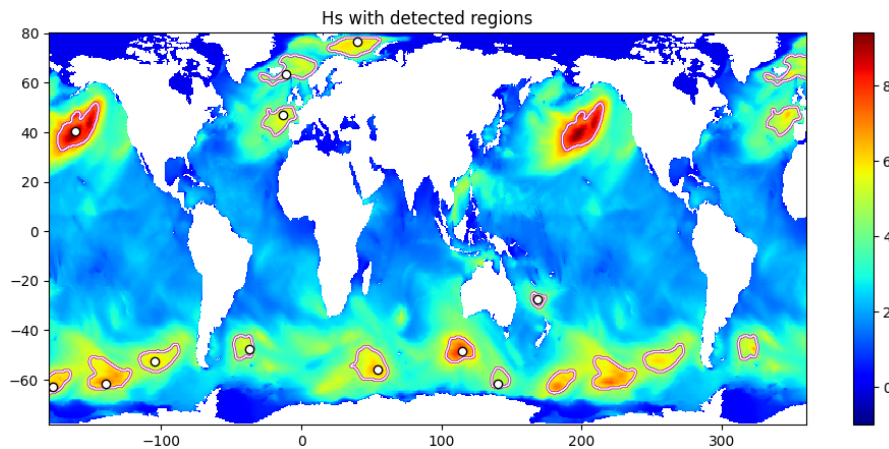
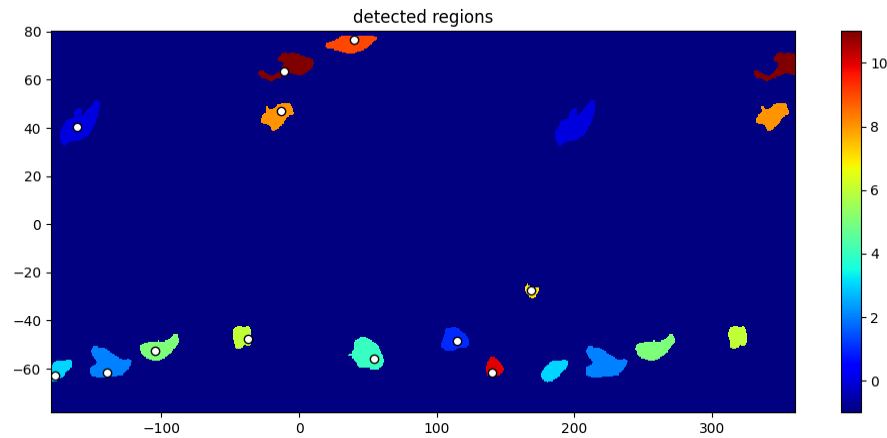
1. At previous level, both regions red and orange have been detected
2. However, at previous level, only the orange regions has been saved,
3. At current level, the detection gives the black region

=> criterion to see if the red region is a 'forgotten\_storm' or not ? OR  
Are there 2 storms or just one ?

The criterion compares the param **area\_forgotten\_ratio** to the area ratio between the 'forgotten' region (here the red region) and the larger of the saved regions area (here the orange region) inside the new region (here the black region).



# Function 'get\_storm\_by\_timestep'



The function gets a map of "detected" or "saved" regions

The *get\_storm\_info\_from\_savemap* function is applied to extract the main information:

- Number of storms detect
- Index of the storm (called 'regions')
- Hs max in the region
- Area of the region in km<sup>2</sup>
- 'lon\_max', 'lat\_max': coordinates of Hs max (same as dataset coordinates longitude, latitude)

# Tracking functions (steps 1 and 2)

For efficiency, 2 types of tracking are done:

- The first one is called **internal tracking** : the tracking is performed from one timestep to the next *inside* one month file.
  - Can be performed using multiprocessing
  - For each file, the storms number goes from 0 to N

=> **Corresponds to step 1 (file\_internal\_track\_only) and included in steps 111, 110 and 11.**
- The second type is called **transition tracking** as it makes the transition between files, tracking storms between the first date of one month and the last date of the previous one, and rename the storms
  - Must be performed sequentially (best with the entire sequence)
  - For each file, the storms number goes from  $N_i$  to  $N_{i+1}$

=> **Corresponds to step 2 (transition\_track\_only) and included in steps 111 and 11.**

# Tracking functions

For both tracking types, the comparison is directly made between time step T and T-1.

A for-loop over all storms from time step T is performed:

For each storm  $st$  in time step T, the function ' $one\_storm\_vs\_old\_storms$ ' is applied.

This function:

- 1) Computes the distance (haversine function) between the storm  $st$  and all storms ( $st\_old_i$ ) from T-1
- 2) The couples ( $st, st\_old_i$ ) with distance < threshold\_dist are considered potential\_points.
- 3) If there is land between  $st$  and  $st\_old_i \Rightarrow$  the couple is removed from potential\_points
- 4) From the remaining potential\_points, the couple ( $st, st\_old_i$ ) with the minimal distance is selected, if there is no remaining potential\_points  $st$  is flagged as *not\_linked*.

If various storms  $st$  are linked to the same  $st\_old_i \Rightarrow$  only the  $st$  with the min distance is kept, the others are flagged as *not\_linked*.

For each storm  $st$  in timestep T,  
if there is a selected couple ( $st, st\_old_i$ )  $\Rightarrow$  assign to  $st$  the number of  $st\_old_i$   
else (if not\_linked)  $\Rightarrow$  the  $st$  is considered a new storm and assigned a new number

