py-eddy-tracker Documentation

Release v3.3.0

A. Delepoulle & E. Mason

Dec 18, 2020

INSTALLATION

CHAPTER

ONE

HOW DO I GET SET UP ?

Source are available on github https://github.com/AntSimi/py-eddy-tracker

Use python3. To avoid problems with installation, use of the virtualenv Python virtual environment is recommended or conda.

Then use pip to install all dependencies (numpy, scipy, matplotlib, netCDF4, . . .), e.g.:

```
pip install numpy scipy netCDF4 matplotlib opencv-python pyyaml pint polygon3
```

Then run the following to install the eddy tracker:

```
python setup.py install
```

Several executables are available in your PATH:

```
GridFiltering # Allow to apply a high frequency filter on a NetCDF grid
EddyId # Provide identification of eddies for one grid
EddySubSetter # Allow to apply sub setting on eddies dataset
EddyTracking # Allow to track Identification dataset
```

CHAPTER

TWO

PY EDDY TRACKER TOOLBOX

All figures in this gallery, used an experimental dataset, compute with this dataset : cmems_product.

CHAPTER

THREE

EDDY DETECTION

3.1 Display contour & circle

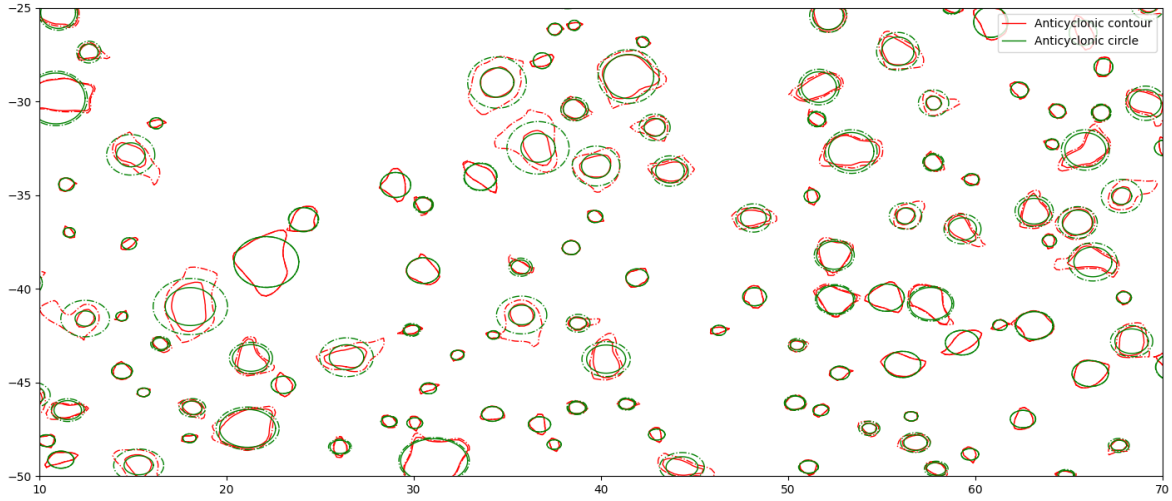
```
from matplotlib import pyplot as plt

from py_eddy_tracker import data
from py_eddy_tracker.observations.observation import EddiesObservations
```

Load detection files

```
a = EddiesObservations.load_file(data.get_path("Anticyclonic_20190223.nc"))
```

Plot the speed and effective (dashed) contours

```
fig = plt.figure(figsize=(15, 8))
ax = fig.add_axes((0.05, 0.05, 0.9, 0.9))
ax.set_aspect("equal")
ax.set_xlim(10, 70)
ax.set_ylim(-50, -25)
a.display(ax, label="Anticyclonic contour", color="r", lw=1)

# Replace contours by circles using center and radius (effective is dashed)
a.circle_contour()
a.display(ax, label="Anticyclonic circle", color="g", lw=1)
ax.legend(loc="upper right")
```



Total running time of the script: (0 minutes 1.568 seconds)

3.2 Display identification

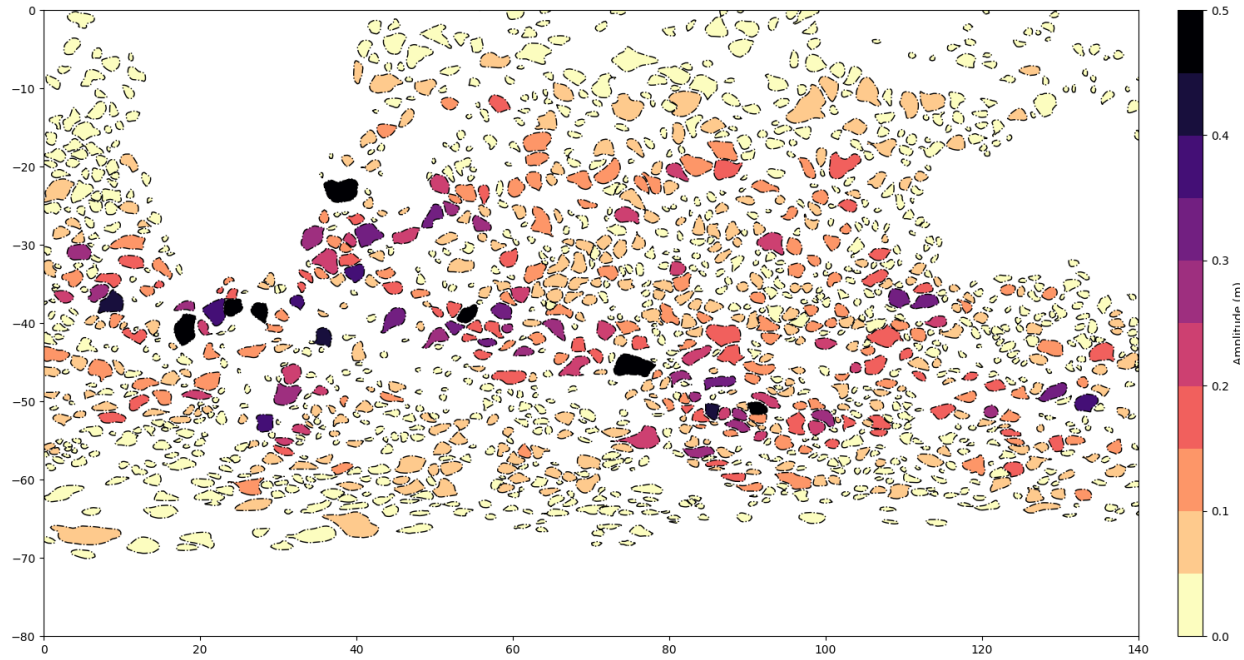
```
from matplotlib import pyplot as plt

from py_eddy_tracker import data
from py_eddy_tracker.observations.observation import EddiesObservations
```

Load detection files

```
a = EddiesObservations.load_file(data.get_path("Anticyclonic_20190223.nc"))
c = EddiesObservations.load_file(data.get_path("Cyclonic_20190223.nc"))
```

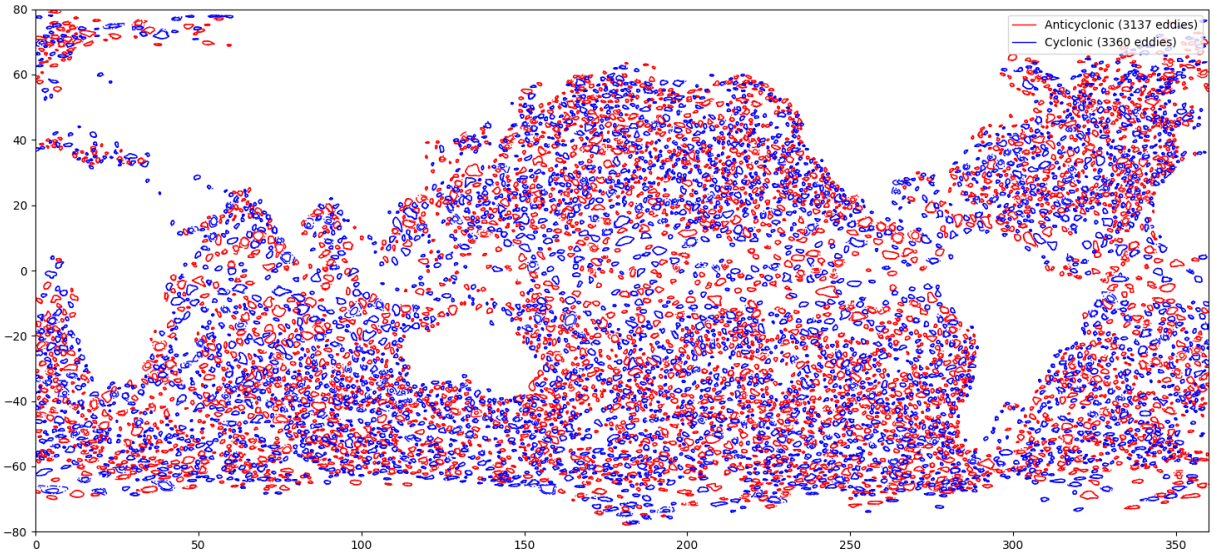
Fill effective contour with amplitude

```
fig = plt.figure(figsize=(15, 8))
ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
ax.set_aspect("equal")
ax.set_xlim(0, 140)
ax.set_ylim(-80, 0)
kwargs = dict(extern_only=True, color="k", lw=1)
a.display(ax, **kwargs), c.display(ax, **kwargs)
a.filled(ax, "amplitude", cmap="magma_r", vmin=0, vmax=0.5)
m = c.filled(ax, "amplitude", cmap="magma_r", vmin=0, vmax=0.5)
colorbar = plt.colorbar(m, cax=ax.figure.add_axes([0.95, 0.03, 0.02, 0.94]))
colorbar.set_label("Amplitude (m)")
```



Draw speed contours

```
fig = plt.figure(figsize=(15, 8))
ax = fig.add_axes([0.03, 0.03, 0.94, 0.94])
ax.set_aspect("equal")
ax.set_xlim(0, 360)
ax.set_ylim(-80, 80)
a.display(ax, label="Anticyclonic ({nb_obs} eddies)", color="r", lw=1)
c.display(ax, label="Cyclonic ({nb_obs} eddies)", color="b", lw=1)
ax.legend(loc="upper right")
```



Get general informations

```
print(a)
```

Out:

```
| 3137 observations from 25255 to 25255 (1 days, ~3137 obs/day)
|   Speed area      : 32.98 Mkm²/day
|   Effective area  : 45.65 Mkm²/day
----Distribution in Amplitude:
|   Amplitude bounds (cm)        0.00      1.00      2.00      3.00      4.00      5.
↪00     10.00    500.00
|   Percent of eddies         :      19.35     22.73     15.40     10.30      6.18
↪ 15.91     10.14
----Distribution in Radius:
|   Speed radius (km)            0.00     15.00     30.00     45.00     60.00     75.
↪00    100.00    200.00   2000.00
|   Percent of eddies         :       0.00      9.47     34.56     24.55     13.29
↪ 11.67      6.34      0.13
|   Effective radius (km)        0.00     15.00     30.00     45.00     60.00     75.
↪00    100.00    200.00   2000.00
|   Percent of eddies         :       0.00      7.52     26.62     20.88     15.40
↪ 15.94     13.32      0.32
----Distribution in Latitude
    Latitude bounds           -90.00    -60.00    -15.00     15.00     60.00     90.
↪00
    Percent of eddies         :       7.62     46.86     12.81     30.06      2.65
    Percent of speed area     :       4.69     41.94     26.90     25.30      1.17
    Percent of effective area :       4.74     43.40     25.53     25.11      1.21
    Mean speed radius (km)    :      43.94     52.75     81.69     51.01     37.91
    Mean effective radius (km):      52.14     62.43     94.14     59.44     44.81
    Mean amplitude (cm)       :       3.53      5.30      2.19      4.32      3.12
```

```
print(c)
```

Out:

```
| 3360 observations from 25255 to 25255 (1 days, ~3360 obs/day)
|   Speed area      : 32.89 Mkm²/day
|   Effective area  : 46.42 Mkm²/day
----Distribution in Amplitude:
|   Amplitude bounds (cm)        0.00      1.00      2.00      3.00      4.00      5.
↪00     10.00    500.00
|   Percent of eddies         :      18.81     24.02     14.11     10.89      5.98
↪ 16.19     10.00
----Distribution in Radius:
|   Speed radius (km)            0.00     15.00     30.00     45.00     60.00     75.
↪00    100.00    200.00   2000.00
|   Percent of eddies         :       0.03     10.15     35.03     25.15     14.40
↪ 10.09      5.12      0.03
|   Effective radius (km)        0.00     15.00     30.00     45.00     60.00     75.
↪00    100.00    200.00   2000.00
|   Percent of eddies         :       0.03      7.98     26.88     21.61     15.92
↪ 15.09     12.14      0.36
----Distribution in Latitude
    Latitude bounds           -90.00    -60.00    -15.00     15.00     60.00     90.
↪00
    Percent of eddies         :       7.92     46.96     13.12     29.61      2.38
    Percent of speed area     :       4.80     41.08     27.30     25.87      0.93
```

(continues on next page)

(continued from previous page)

```
    Percent of effective area :       4.83     42.35     25.36     26.55      0.92
    Mean speed radius (km)    :      42.23     50.71     78.76     50.80     34.64
    Mean effective radius (km):      49.25     60.50     89.91     59.96     40.20
    Mean amplitude (cm)       :       3.19      5.71      2.19      4.24      2.42
```

Total running time of the script: (0 minutes 2.586 seconds)

3.3 Radius vs area

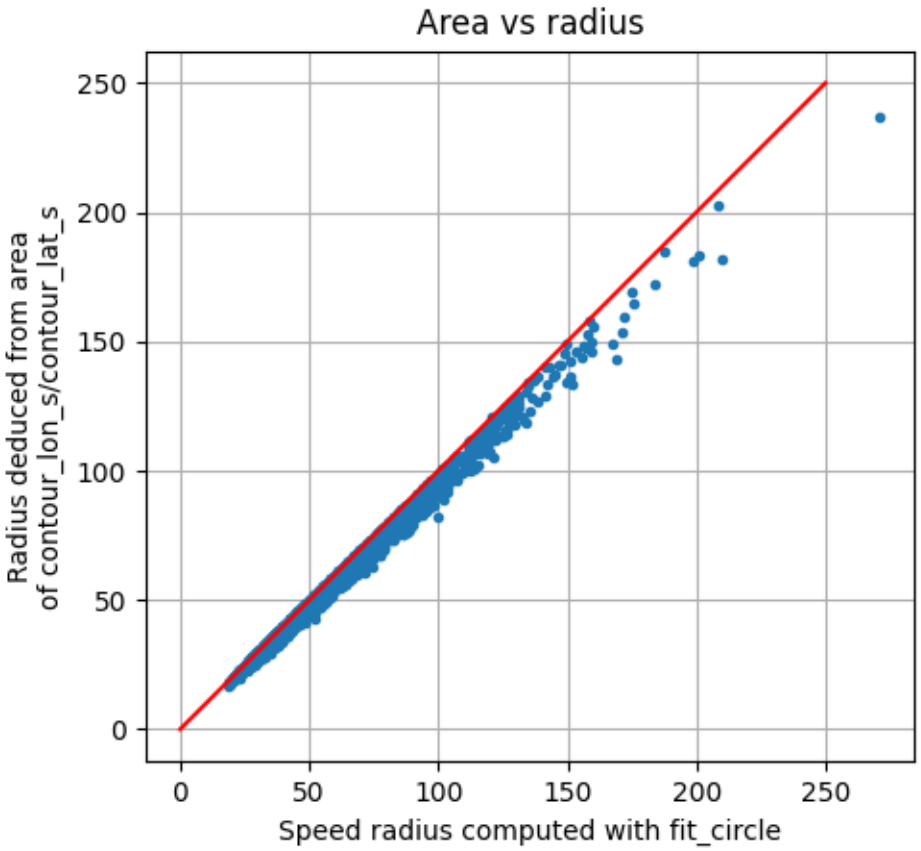
```
from matplotlib import pyplot as plt
from numpy import array, pi

from py_eddy_tracker import data
from py_eddy_tracker.generic import coordinates_to_local
from py_eddy_tracker.observations.observation import EddiesObservations
from py_eddy_tracker.poly import poly_area
```

Load detection files

```
a = EddiesObservations.load_file(data.get_path("Anticyclonic_20190223.nc"))
areas = list()
# For each contour area will be compute in local reference
for i in a:
    x, y = coordinates_to_local(
        i["contour_lon_s"], i["contour_lat_s"], i["lon"], i["lat"]
    )
    areas.append(poly_area(x, y))
areas = array(areas)
```

Radius provided by eddy detection is computed with *fit_circle()* method. This radius will be compared with an equivalent radius deduced from polygon area.

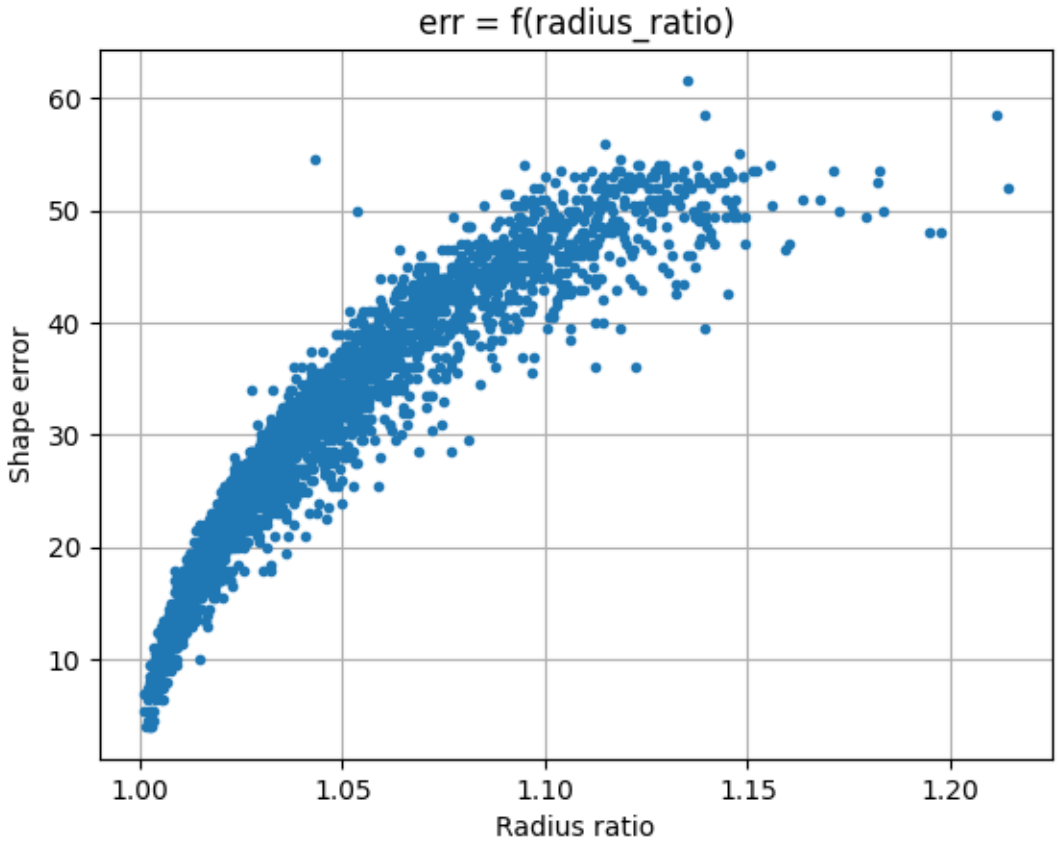
```
ax = plt.subplot(111)
ax.set_aspect("equal")
ax.grid()
ax.set_xlabel("Speed radius computed with fit_circle")
ax.set_ylabel("Radius deduced from area\nof contour_lon_s/contour_lat_s")
ax.set_title("Area vs radius")
ax.plot(a["radius_s"] / 1000.0, (areas / pi) ** 0.5 / 1000.0, ".")
ax.plot((0, 250), (0, 250), "r")
```



Fit circle give a radius bigger than polygon area

When error is tiny, radius are very close.

```
ax = plt.subplot(111)
ax.grid()
ax.set_xlabel("Radius ratio")
ax.set_ylabel("Shape error")
ax.set_title("err = f(radius_ratio)")
ax.plot(a["radius_s"] / (areas / pi) ** 0.5, a["shape_error_s"], ".")
```



Total running time of the script: (0 minutes 3.475 seconds)

3.4 Shape error gallery

Gallery of contours with shape error

```
from matplotlib import pyplot as plt
from numpy import arange, cos, linspace, radians, sin

from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker.eddy_feature import Contours
from py_eddy_tracker.generic import local_to_coordinates
```

Method to built circle from center coordinates

```
def build_circle(x0, y0, r):
    angle = radians(linspace(0, 360, 50))
    x_norm, y_norm = cos(angle), sin(angle)
    return local_to_coordinates(x_norm * r, y_norm * r, x0, y0)
```

We iterate over closed contours and sort with regards of shape error

```
g = RegularGridDataset(
    data.get_path("dt_med_allsat_phy_l4_20160515_20190101.nc"), "longitude", "latitude
↪"
)
c = Contours(g.x_c, g.y_c, g.grid("adt") * 100, arange(-50, 50, 0.2))
contours = dict()
for coll in c.iter():
    for current_contour in coll.get_paths():
        _, _, _, aerr = current_contour.fit_circle()
        i = int(aerr // 4) + 1
        if i not in contours:
            contours[i] = list()
        contours[i].append(current_contour)
```

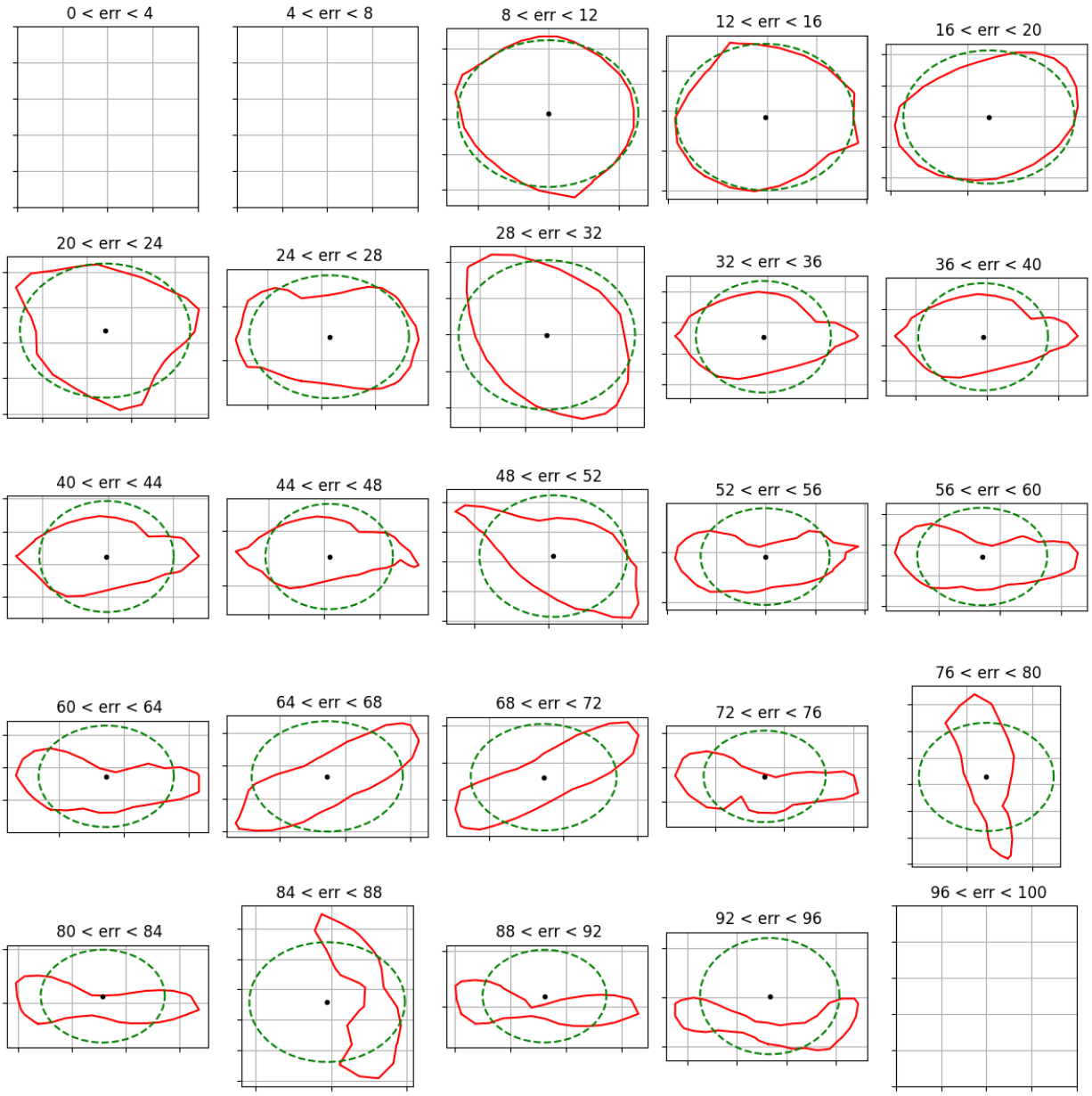
Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

3.4.1 Shape error gallery

For each contour display, we display circle fitted, we work at different latitude circle could have distorsion

```
fig = plt.figure(figsize=(12, 12))
for i in range(1, 26):
    e_min, e_max = (i - 1) * 4, i * 4
    ax = plt.subplot(5, 5, i, title=f" {e_min} < err < {e_max}")
    ax.xaxis.set_ticklabels([])
    ax.yaxis.set_ticklabels([])
    ax.set_aspect("equal")
    ax.grid()
    if i in contours:
        for contour in contours[i]:
            x, y = contour.lon, contour.lat
            x0, y0, radius, _ = contour.fit_circle()
            if x.shape[0] > 30 and 30000 < radius < 70000:
                # Plot only first contour found
                m = ax.plot(x, y, "r")[0]
                ax.plot(*build_circle(x0, y0, radius), "g--")
                ax.plot(x0, y0, "k.")
                break
plt.tight_layout()
```



Total running time of the script: (0 minutes 9.358 seconds)

3.5 Get mean of grid in each eddies

```
from matplotlib import pyplot as plt

from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker.observations.observation import EddiesObservations
```

```
def start_axes(title):
    fig = plt.figure(figsize=(13, 5))
```

(continues on next page)

(continued from previous page)

```
    ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
    ax.set_aspect("equal")
    ax.set_title(title)
    return ax


def update_axes(ax, mappable=None):
    ax.grid()
    ax.legend()
    if mappable:
        plt.colorbar(mappable, cax=ax.figure.add_axes([0.95, 0.05, 0.01, 0.9]))
```

Load detection files and data to interp

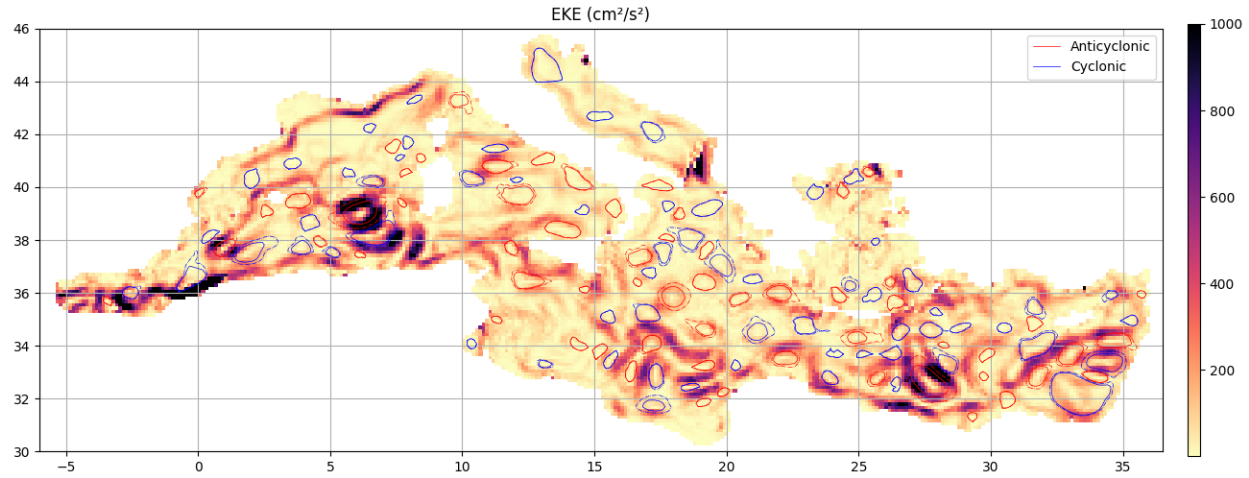
```
a = EddiesObservations.load_file(data.get_path("Anticyclonic_20160515.nc"))
c = EddiesObservations.load_file(data.get_path("Cyclonic_20160515.nc"))

aviso_map = RegularGridDataset(
    data.get_path("dt_med_allsat_phy_l4_20160515_20190101.nc"), "longitude", "latitude
↪"
)
aviso_map.add_uv("adt")
```

Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

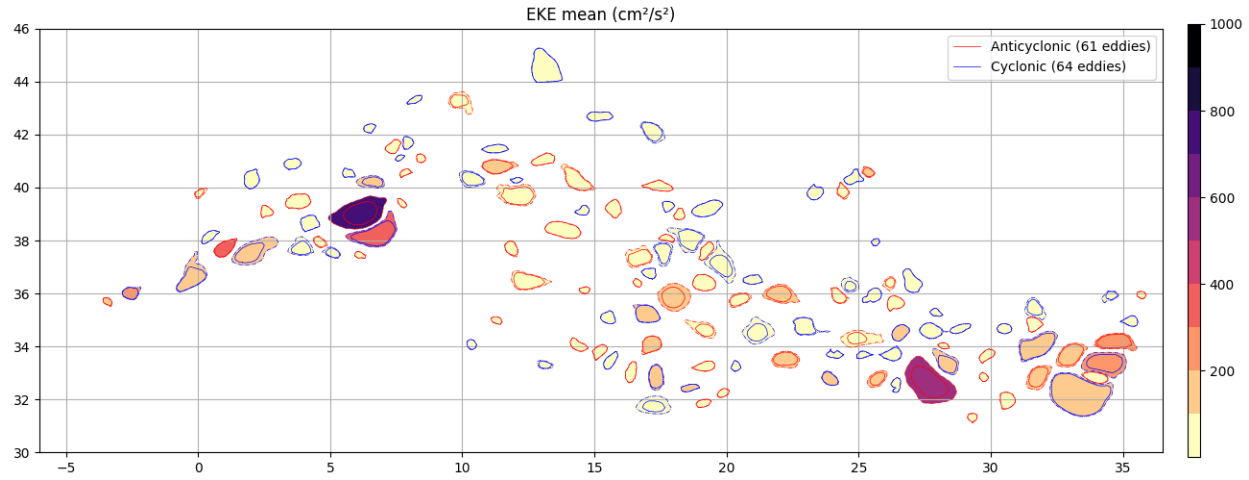
Compute and store eke in cm^2/s^2

```
aviso_map.add_grid(
    "eke", (aviso_map.grid("u") ** 2 + aviso_map.grid("v") ** 2) * 0.5 * (100 ** 2)
)

eke_kwargs = dict(vmin=1, vmax=1000, cmap="magma_r")

ax = start_axes("EKE (cm$^2$/s$^2$)")
m = aviso_map.display(ax, "eke", **eke_kwargs)
a.display(ax, color="r", linewidth=0.5, label="Anticyclonic", ref=-10)
c.display(ax, color="b", linewidth=0.5, label="Cyclonic", ref=-10)
update_axes(ax, m)
```



Get mean of eke in each effective contour

```
ax = start_axes("EKE mean (cm$^2$/s$^2$)")
a.display(ax, color="r", linewidth=0.5, label="Anticyclonic ({nb_obs} eddies)", ref=-
↪10)
c.display(ax, color="b", linewidth=0.5, label="Cyclonic ({nb_obs} eddies)", ref=-10)
eke = a.interp_grid(aviso_map, "eke", method="mean", intern=False)
a.filled(ax, eke, ref=-10, **eke_kwargs)
eke = c.interp_grid(aviso_map, "eke", method="mean", intern=False)
m = c.filled(ax, eke, ref=-10, **eke_kwargs)
update_axes(ax, m)
```



Total running time of the script: (0 minutes 5.625 seconds)

3.6 Eddy detection : Med

Script will detect eddies on adt field, and compute u,v with method add_uv(which could use, only if equator is avoid)

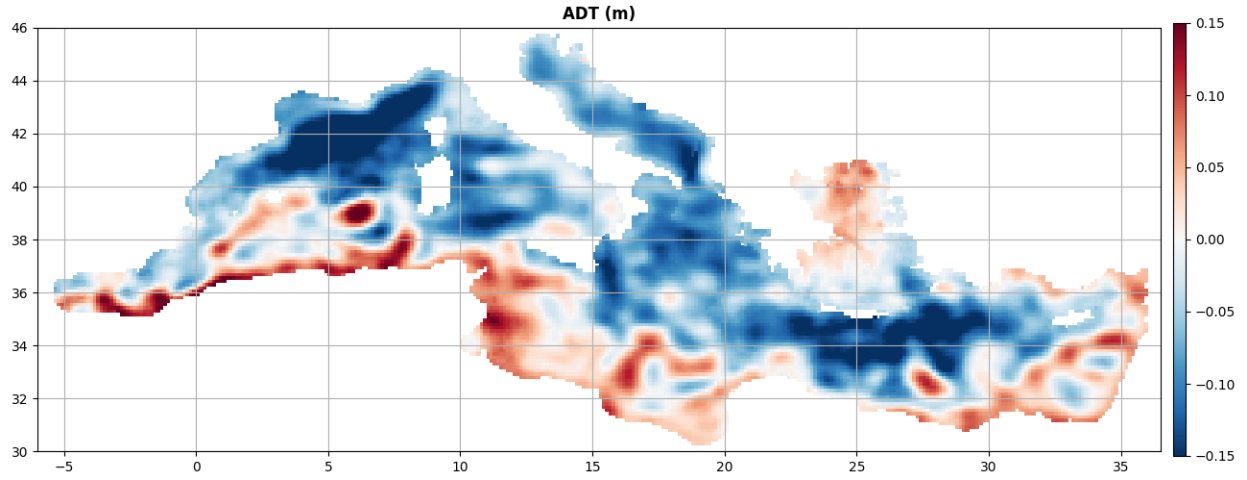
Figures will show different step to detect eddies.

```
from datetime import datetime

from matplotlib import pyplot as plt
from numpy import arange

from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset
```

```
def start_axes(title):
    fig = plt.figure(figsize=(13, 5))
    ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
    ax.set_aspect("equal")
    ax.set_title(title, weight="bold")
    return ax


def update_axes(ax, mappable=None):
    ax.grid()
    if mappable:
        plt.colorbar(mappable, cax=ax.figure.add_axes([0.94, 0.05, 0.01, 0.9]))
```

Load Input grid, ADT is used to detect eddies

```
g = RegularGridDataset(
    data.get_path("dt_med_allsat_phy_l4_20160515_20190101.nc"), "longitude", "latitude
↪"
)

ax = start_axes("ADT (m)")
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15, cmap="RdBu_r")
update_axes(ax, m)
```



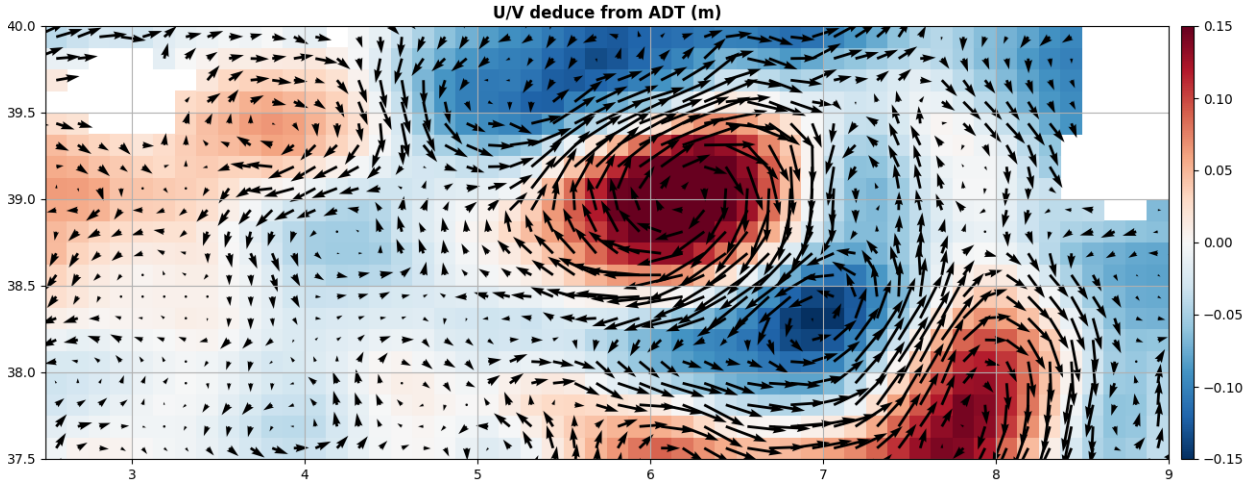
Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

3.6.1 Get geostrophic speed u,v

U/V are deduced from ADT, this algortihm is not ok near the equator (~+- 2°)

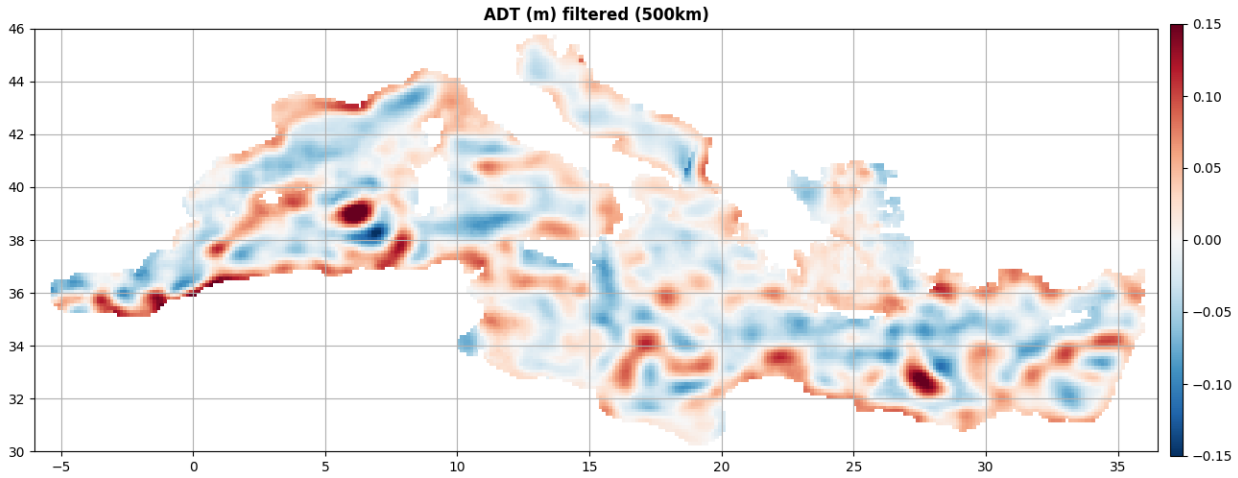
```
g.add_uv("adt")
ax = start_axes("U/V deduce from ADT (m)")
ax.set_xlim(2.5, 9), ax.set_ylim(37.5, 40)
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15, cmap="RdBu_r")
u, v = g.grid("u").T, g.grid("v").T
ax.quiver(g.x_c, g.y_c, u, v, scale=10)
update_axes(ax, m)
```



3.6.2 Pre-processings

Apply a high-pass filter to remove the large scale and highlight the mesoscale

```
g.bessel_high_filter("adt", 500)
ax = start_axes("ADT (m) filtered (500km)")
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15, cmap="RdBu_r")
update_axes(ax, m)
```



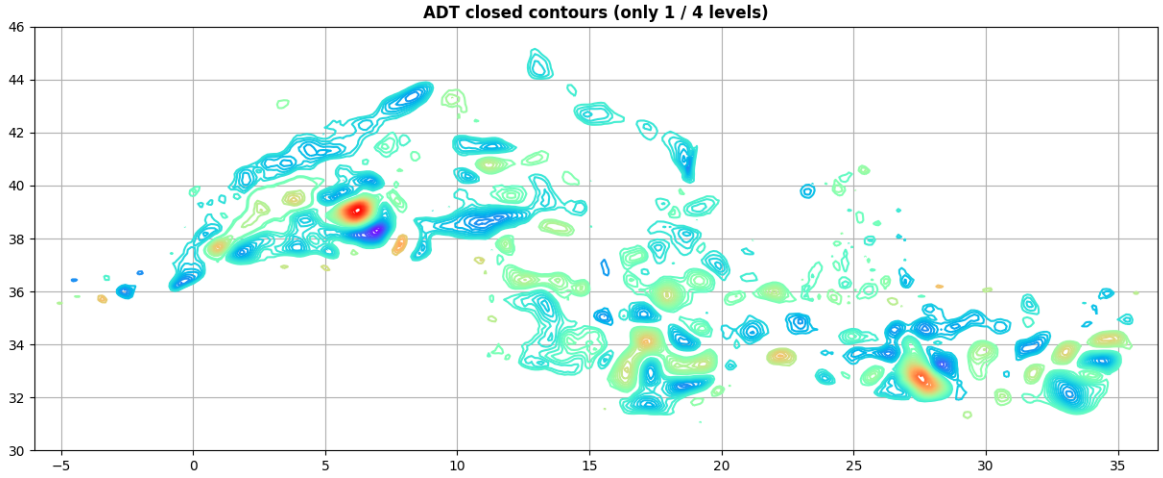
3.6.3 Identification

Run the identification step with slices of 2 mm

```
date = datetime(2016, 5, 15)
a, c = g.eddy_identification("adt", "u", "v", date, 0.002, shape_error=55)
```

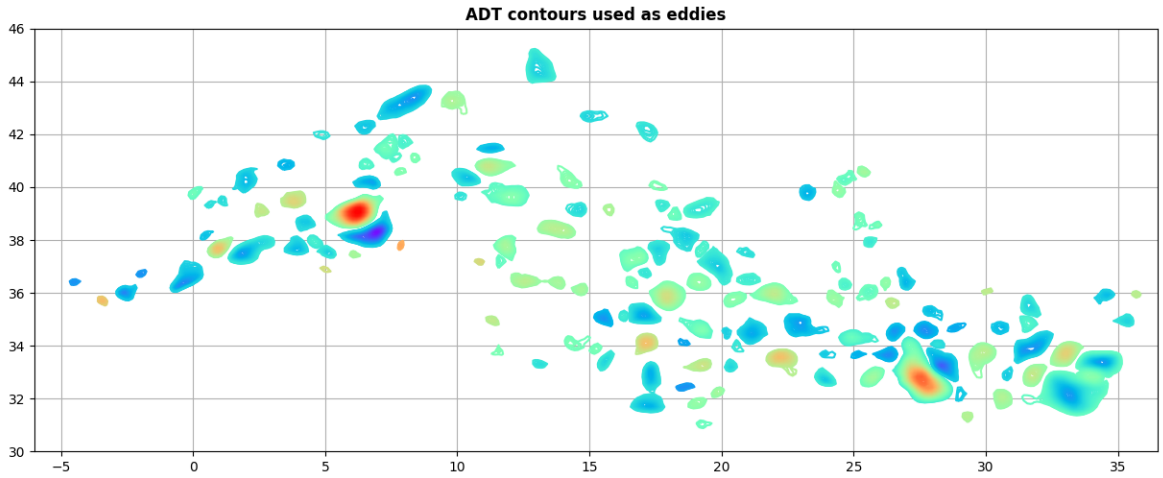
Display of all closed contours found in the grid (only 1 contour every 4)

```
ax = start_axes("ADT closed contours (only 1 / 4 levels)")
g.contours.display(ax, step=4)
update_axes(ax)
```



Contours included in eddies

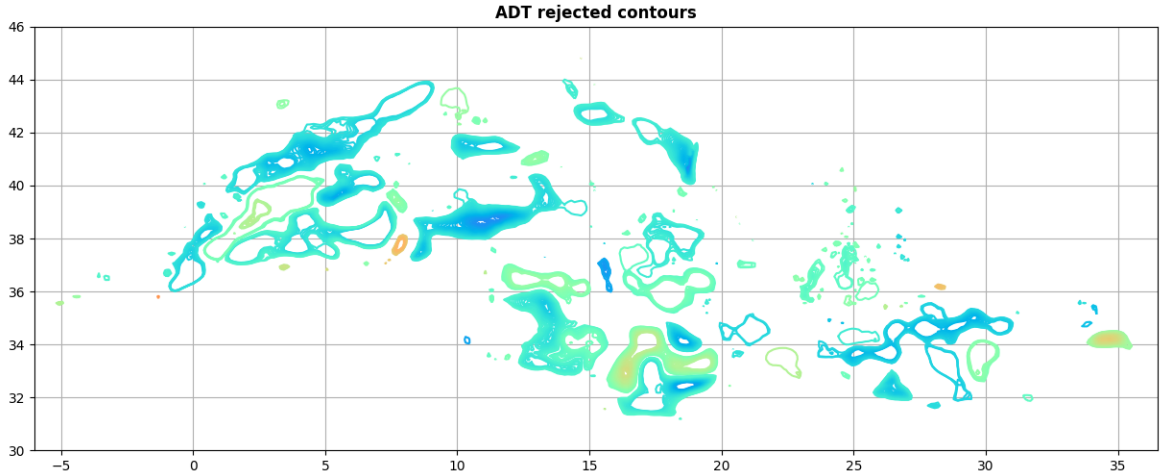
```
ax = start_axes("ADT contours used as eddies")
g.contours.display(ax, only_used=True)
update_axes(ax)
```



3.6.4 Post analysis

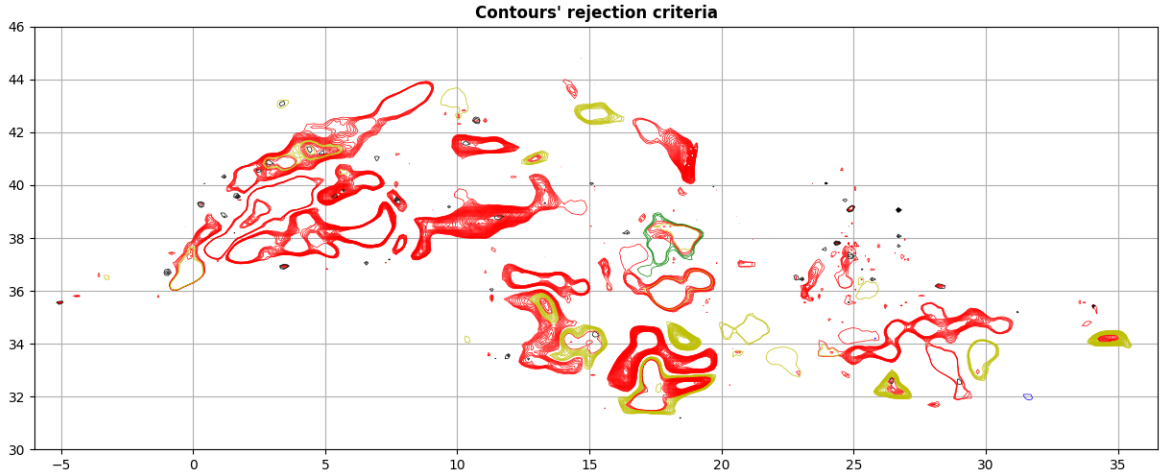
Contours can be rejected for several reasons (shape error to high, several extremum in contour, . . .)

```
ax = start_axes("ADT rejected contours")
g.contours.display(ax, only_unused=True)
update_axes(ax)
```



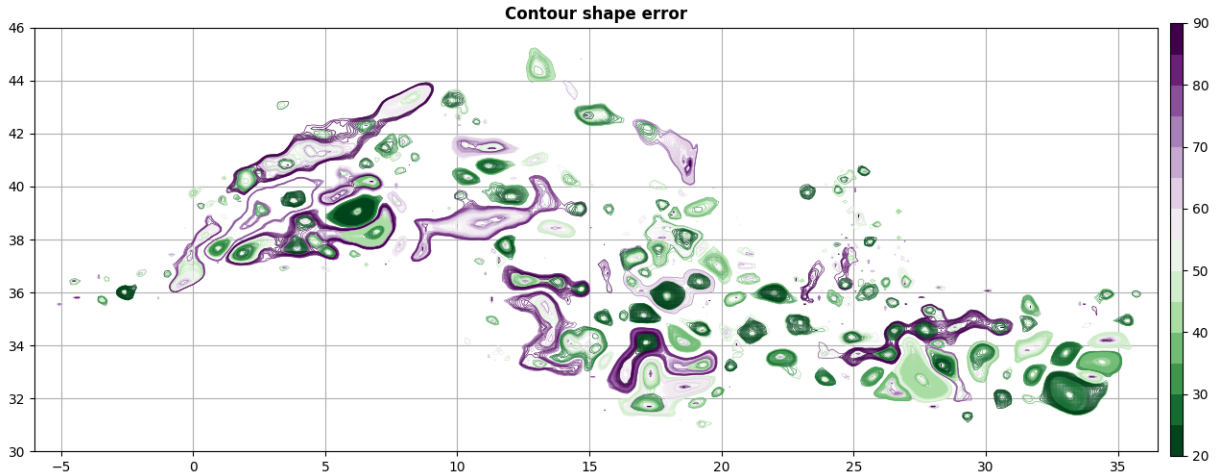
Creteria for rejecting a contour 0. - Accepted (green) 1. - Rejection for shape error (red) 2. - Masked value within contour (blue) 3. - Under or over the pixel limit bounds (black) 4. - Amplitude criterion (yellow)

```
ax = start_axes("Contours' rejection criteria")
g.contours.display(ax, only_unused=True, lw=0.5, display_criterion=True)
update_axes(ax)
```



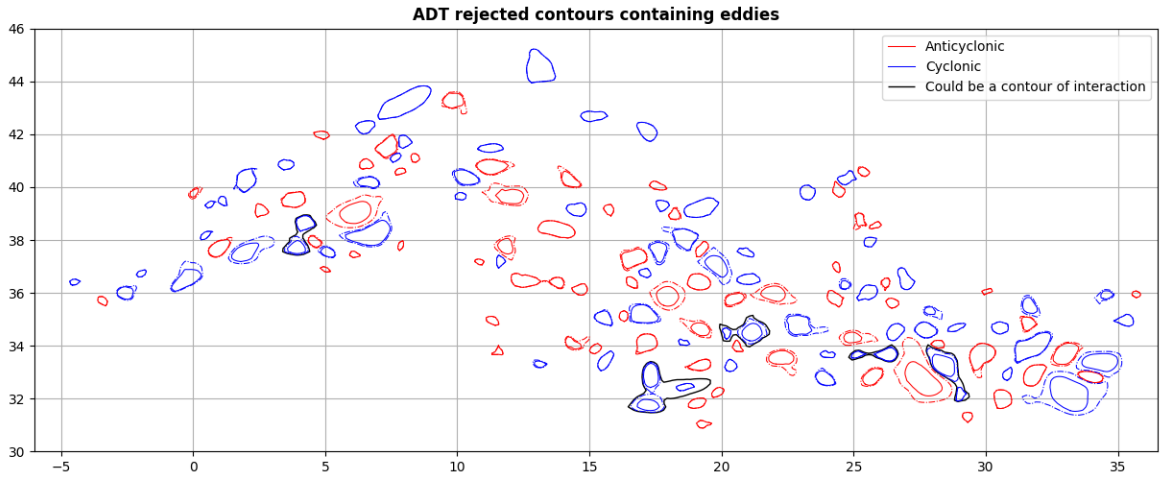
Display the shape error of each tested contour, the limit of shape error is set to 55 %

```
ax = start_axes("Contour shape error")
m = g.contours.display(
    ax, lw=0.5, field="shape_error", bins=arange(20, 90.1, 5), cmap="PRGn_r"
)
update_axes(ax, m)
```



Some closed contours contains several eddies (aka, more than one extremum)

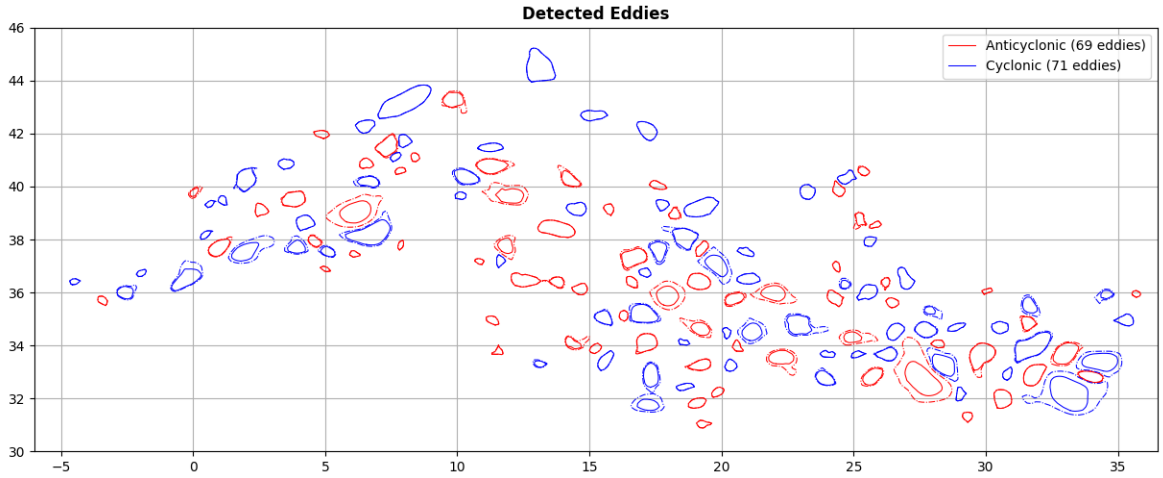
```
ax = start_axes("ADT rejected contours containing eddies")
g.contours.label_contour_unused_which_contain_eddies(a)
g.contours.label_contour_unused_which_contain_eddies(c)
g.contours.display(
    ax,
    only_contain_eddies=True,
    color="k",
    lw=1,
    label="Could be a contour of interaction",
)
a.display(ax, color="r", linewidth=0.75, label="Anticyclonic", ref=-10)
c.display(ax, color="b", linewidth=0.75, label="Cyclonic", ref=-10)
ax.legend()
update_axes(ax)
```



3.6.5 Output

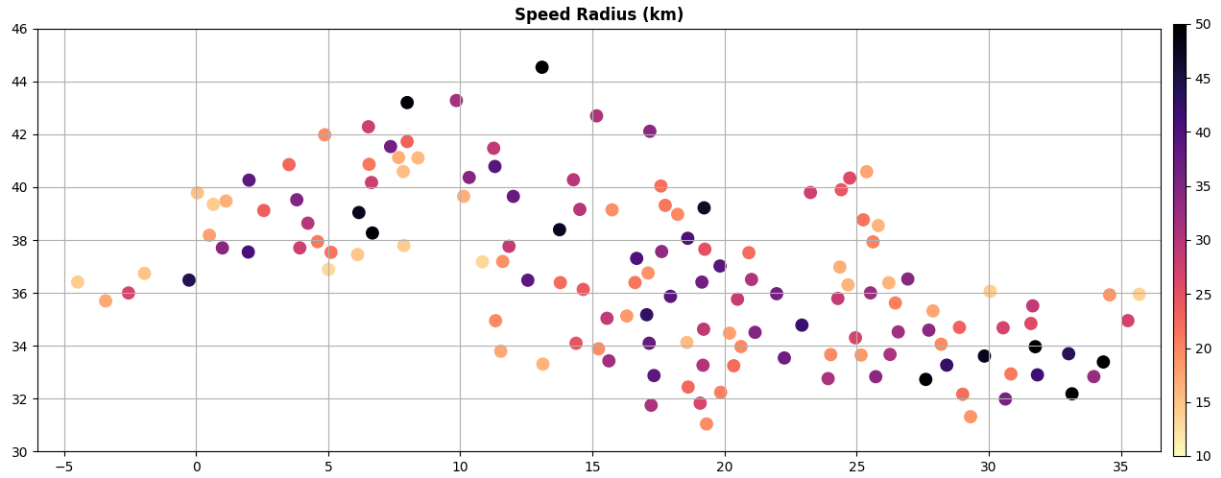
When displaying the detected eddies, dashed lines are for effective contour, solide lines for the contour of the maximum mean speed. See figure 1 of https://doi.org/10.1175/JTECH-D-14-00019.1

```
ax = start_axes("Detected Eddies")
a.display(
    ax, color="r", linewidth=0.75, label="Anticyclonic ({nb_obs} eddies)", ref=-10
)
c.display(ax, color="b", linewidth=0.75, label="Cyclonic ({nb_obs} eddies)", ref=-10)
ax.legend()
update_axes(ax)
```



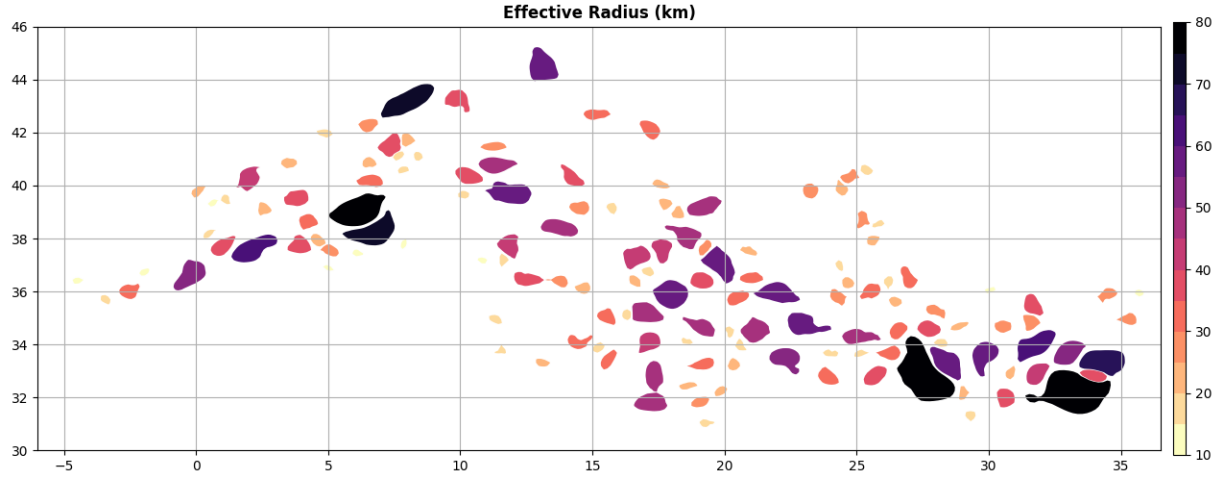
Display the speed radius of the detected eddies

```
ax = start_axes("Speed Radius (km)")
a.scatter(ax, "radius_s", vmin=10, vmax=50, s=80, ref=-10, cmap="magma_r", factor=0.
↪001)
m = c.scatter(
    ax, "radius_s", vmin=10, vmax=50, s=80, ref=-10, cmap="magma_r", factor=0.001
)
update_axes(ax, m)
```



Filling the effective radius contours with the effective radius values

```
ax = start_axes("Effective Radius (km)")
kwargs = dict(vmin=10, vmax=80, cmap="magma_r", factor=0.001, lut=14, ref=-10)
a.filled(ax, "effective_radius", **kwargs)
m = c.filled(
    ax, "radius_e", vmin=10, vmax=80, cmap="magma_r", factor=0.001, lut=14, ref=-10
)
update_axes(ax, m)
```



Total running time of the script: (0 minutes 19.600 seconds)

3.7 Eddy detection : Gulf stream

Script will detect eddies on adt field, and compute u,v with method add_uv(which could use, only if equator is avoid)

Figures will show different step to detect eddies.

```
from datetime import datetime

from matplotlib import pyplot as plt
from numpy import arange
```

(continues on next page)

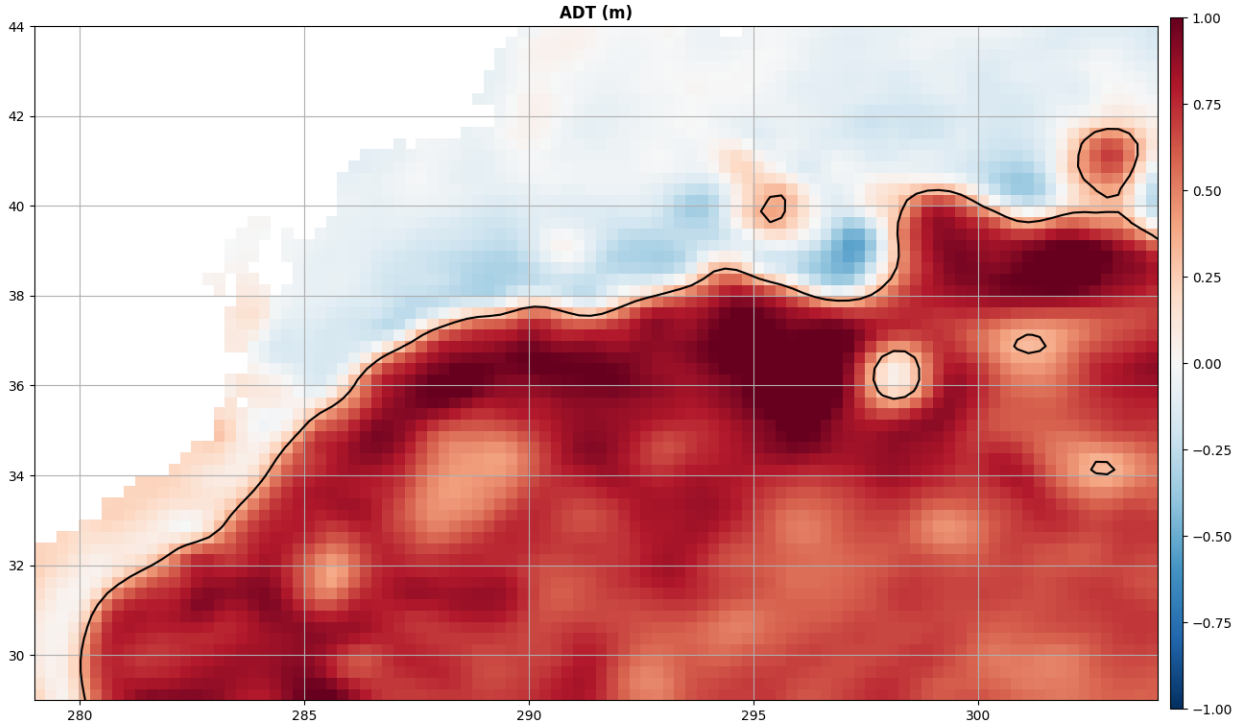
(continued from previous page)

```
from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker.eddy_feature import Contours
```

```
def start_axes(title):
    fig = plt.figure(figsize=(13, 8))
    ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
    ax.set_xlim(279, 304), ax.set_ylim(29, 44)
    ax.set_aspect("equal")
    ax.set_title(title, weight="bold")
    return ax


def update_axes(ax, mappable=None):
    ax.grid()
    if mappable:
        plt.colorbar(mappable, cax=ax.figure.add_axes([0.94, 0.05, 0.01, 0.9]))
```

Load Input grid, ADT is used to detect eddies

```
margin = 30
g = RegularGridDataset(
    data.get_path("nrt_global_allsat_phy_l4_20190223_20190226.nc"),
    "longitude",
    "latitude",
    # Manual area subset
    indexs=dict(
        longitude=slice(1116 - margin, 1216 + margin),
        latitude=slice(476 - margin, 536 + margin),
    ),
)

ax = start_axes("ADT (m)")
m = g.display(ax, "adt", vmin=-1, vmax=1, cmap="RdBu_r")
# Draw line on the gulf stream front
great_current = Contours(g.x_c, g.y_c, g.grid("adt"), levels=(0.35,), keep_
↪unclose=True)
great_current.display(ax, color="k")
update_axes(ax, m)
```



Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/nrt_global_allsat_phy_l4_20190223_20190226.nc
```

3.7.1 Get geostrophic speed u,v

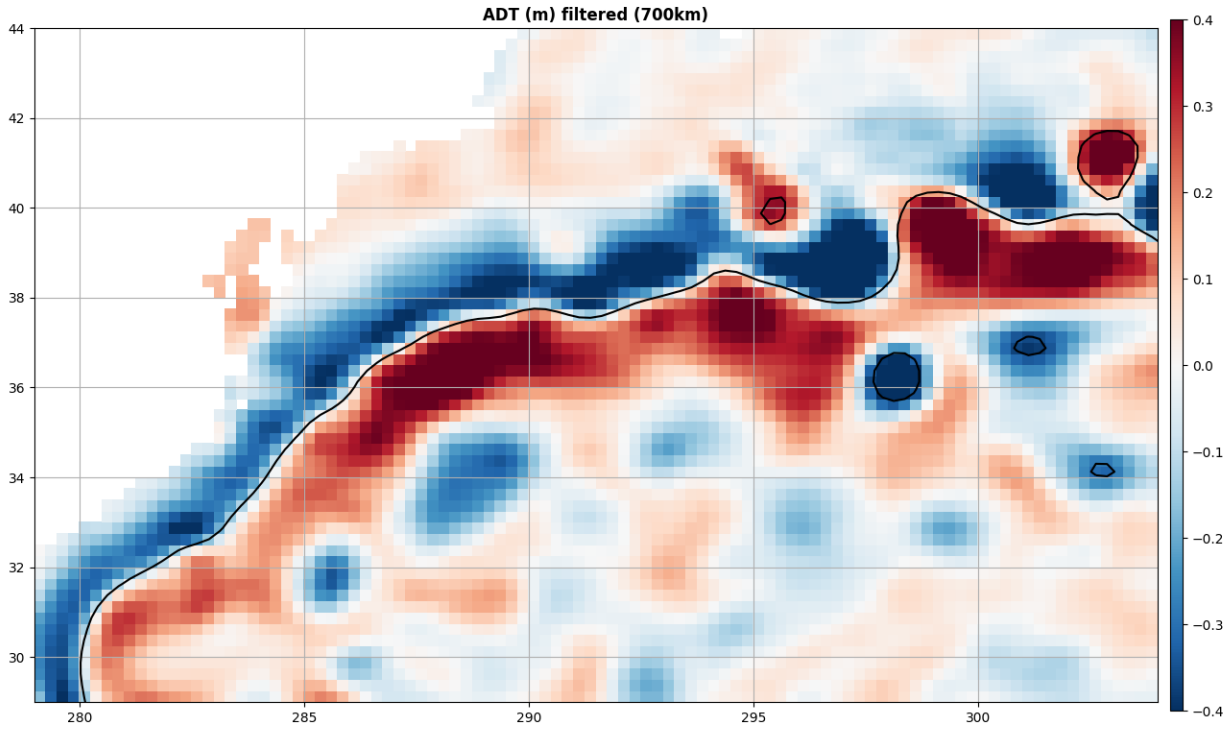
U/V are deduced from ADT, this algortihm is not ok near the equator (~+- 2°)

```
g.add_uv("adt")
```

3.7.2 Pre-processings

Apply a high-pass filter to remove the large scale and highlight the mesoscale

```
g.bessel_high_filter("adt", 700)
ax = start_axes("ADT (m) filtered (700km)")
m = g.display(ax, "adt", vmin=-0.4, vmax=0.4, cmap="RdBu_r")
great_current.display(ax, color="k")
update_axes(ax, m)
```



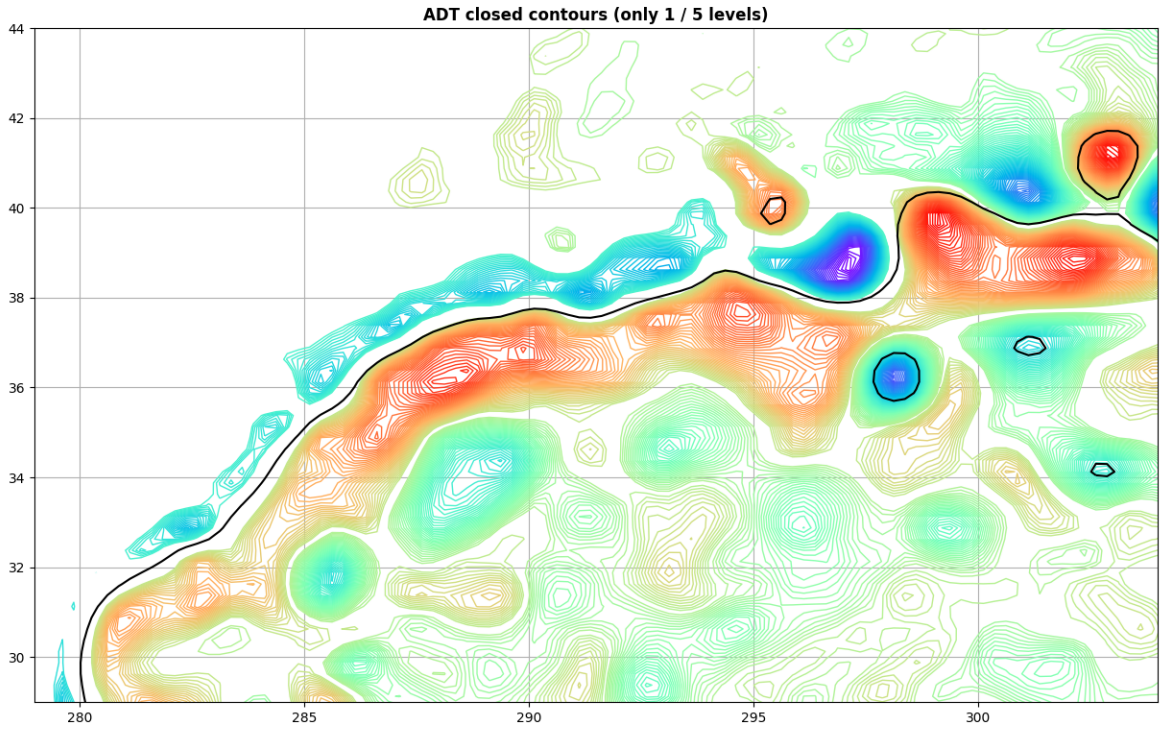
3.7.3 Identification

Run the identification step with slices of 2 mm

```
date = datetime(2016, 5, 15)
a, c = g.eddy_identification("adt", "u", "v", date, 0.002, shape_error=55)
```

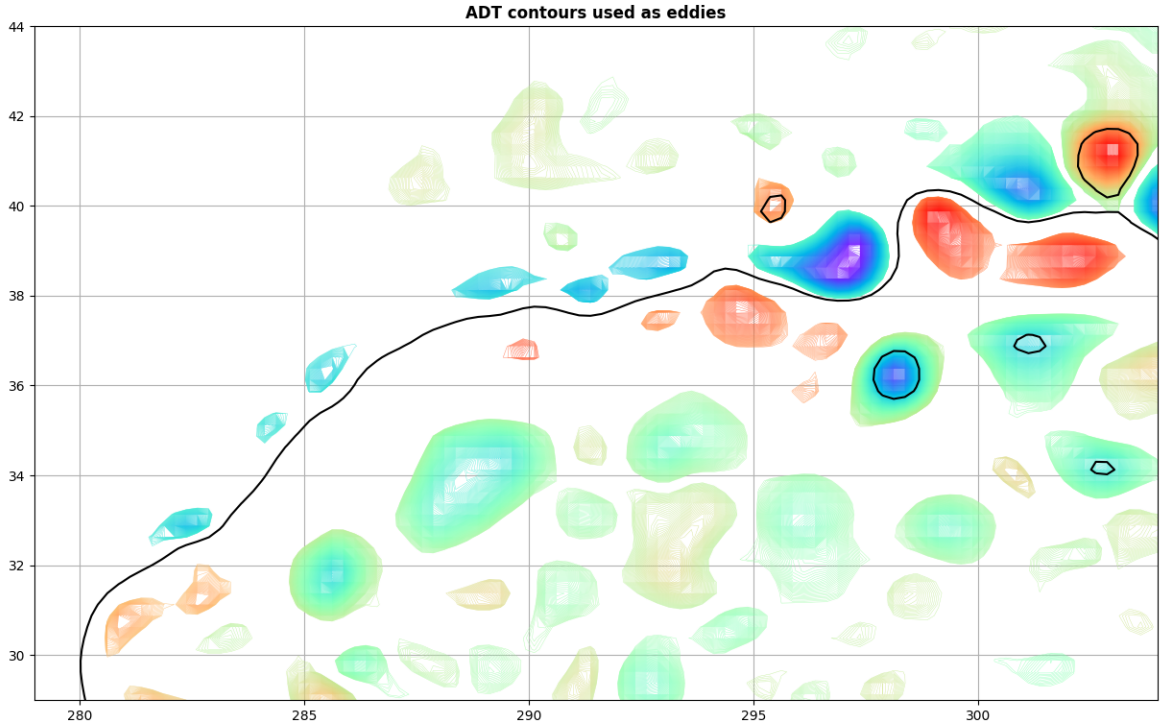
Display of all closed contours found in the grid (only 1 contour every 5)

```
ax = start_axes("ADT closed contours (only 1 / 5 levels)")
g.contours.display(ax, step=5, lw=1)
great_current.display(ax, color="k")
update_axes(ax)
```



Contours included in eddies

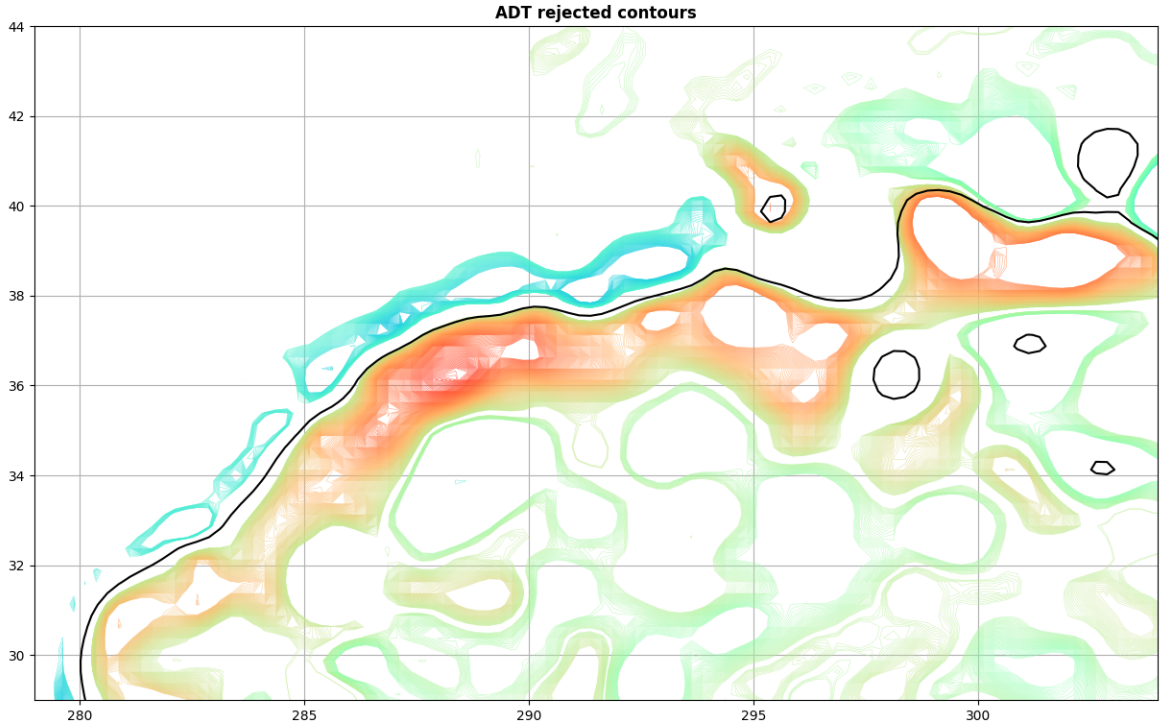
```
ax = start_axes("ADT contours used as eddies")
g.contours.display(ax, only_used=True, lw=0.25)
great_current.display(ax, color="k")
update_axes(ax)
```



3.7.4 Post analysis

Contours can be rejected for several reasons (shape error to high, several extremum in contour, . . .)

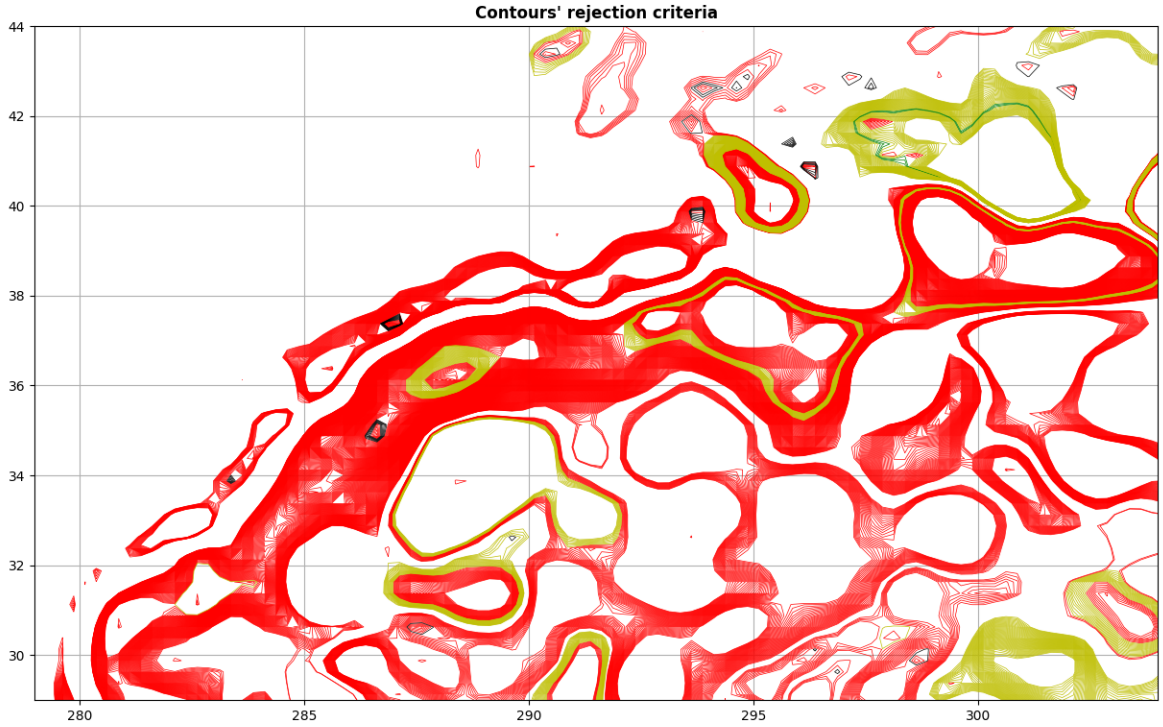
```
ax = start_axes("ADT rejected contours")
g.contours.display(ax, only_unused=True, lw=0.25)
great_current.display(ax, color="k")
update_axes(ax)
```



Criteria for rejecting a contour :

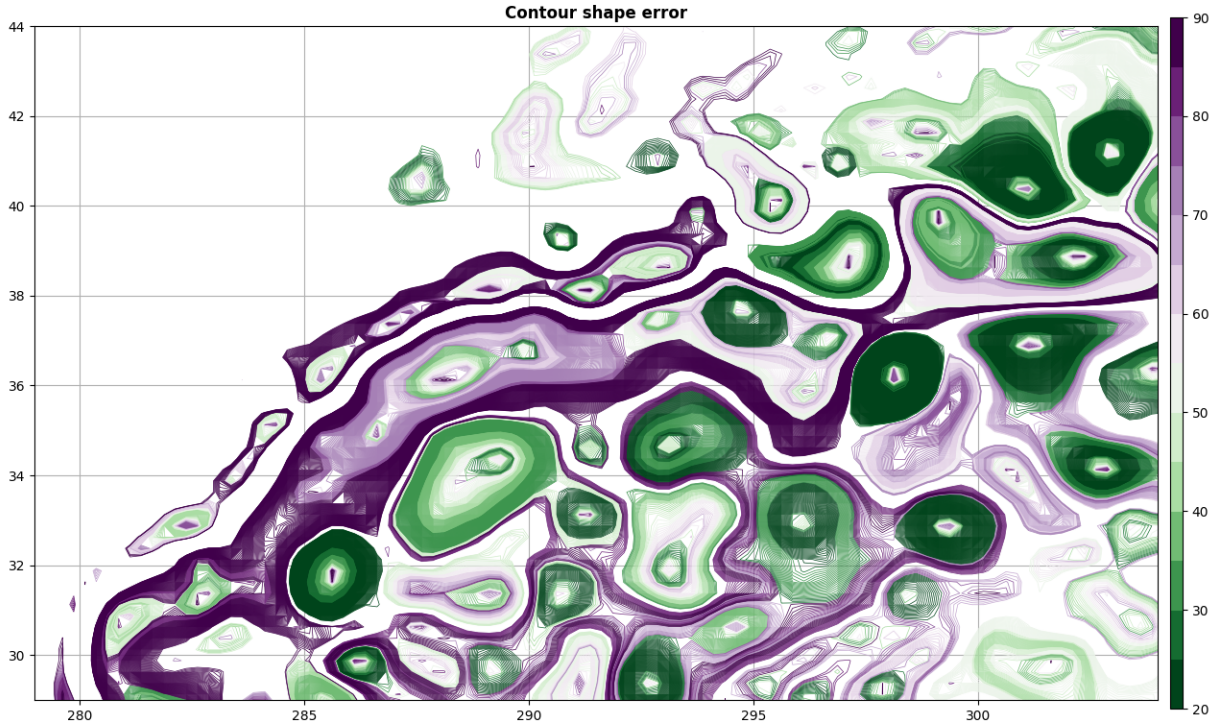
0. Accepted (green)
1. Rejection for shape error (red)
2. Masked value within contour (blue)
3. Under or over the pixel limit bounds (black)
4. Amplitude criterion (yellow)

```
ax = start_axes("Contours' rejection criteria")
g.contours.display(ax, only_unused=True, lw=0.5, display_criterion=True)
update_axes(ax)
```



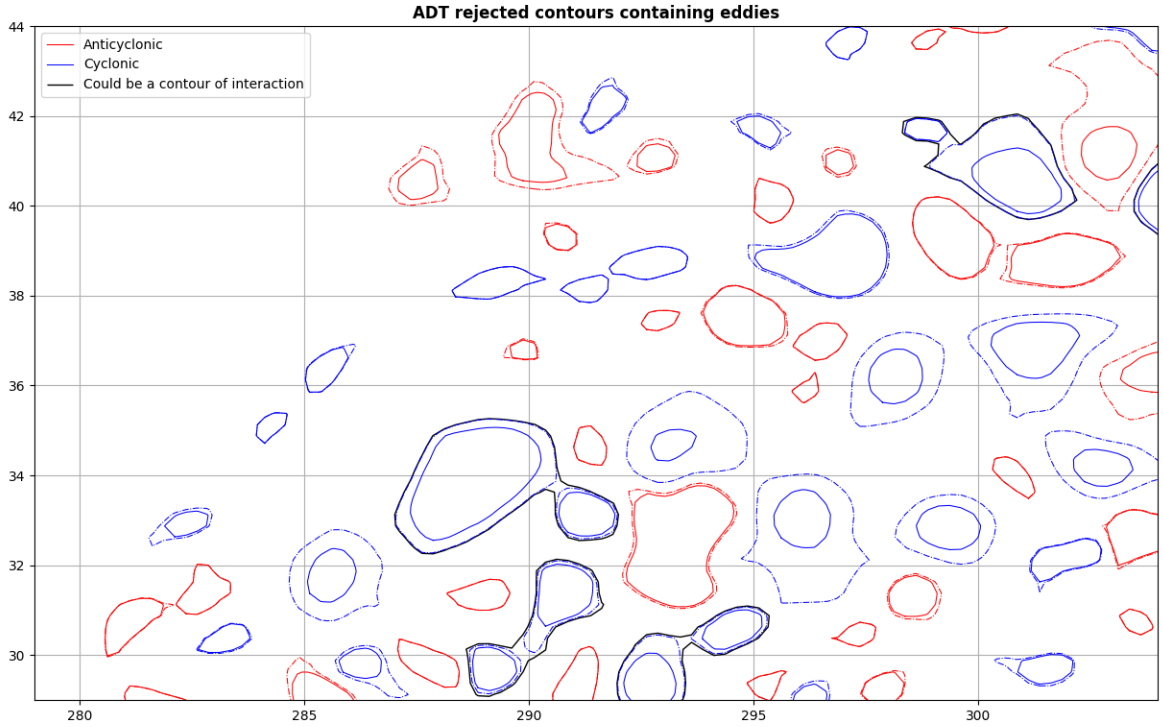
Display the shape error of each tested contour, the limit of shape error is set to 55 %

```
ax = start_axes("Contour shape error")
m = g.contours.display(
    ax, lw=0.5, field="shape_error", bins=arange(20, 90.1, 5), cmap="PRGn_r"
)
update_axes(ax, m)
```



Some closed contours contains several eddies (aka, more than one extremum)

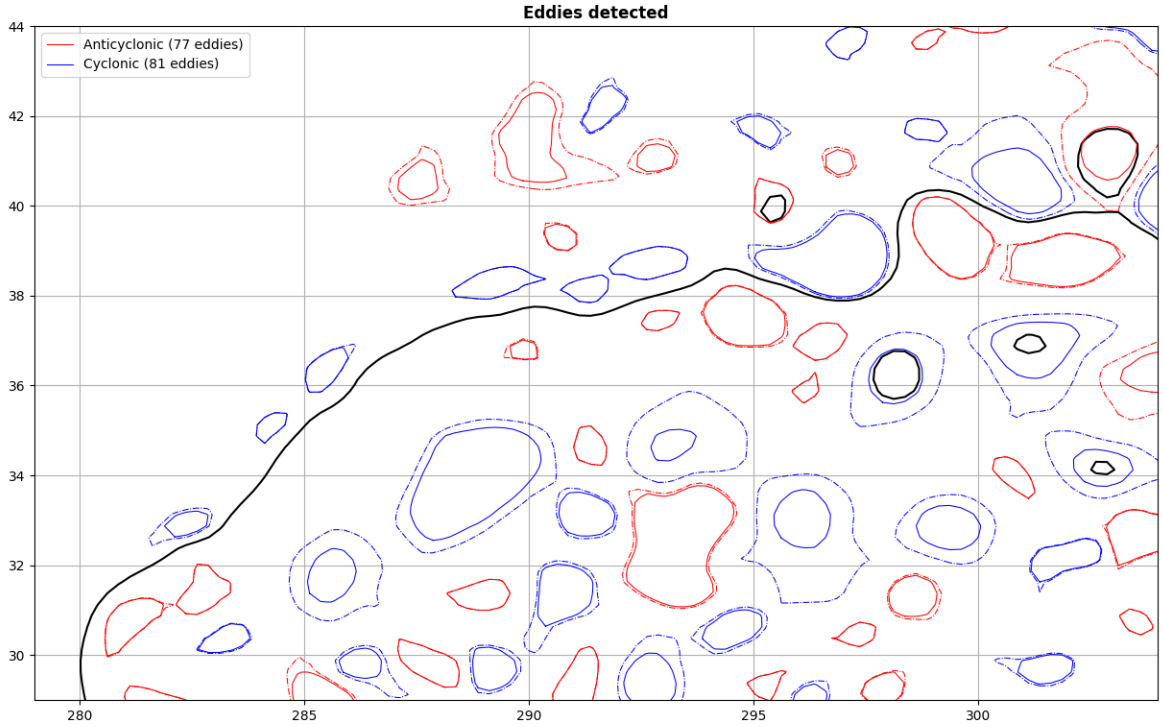
```
ax = start_axes("ADT rejected contours containing eddies")
g.contours.label_contour_unused_which_contain_eddies(a)
g.contours.label_contour_unused_which_contain_eddies(c)
g.contours.display(
    ax,
    only_contain_eddies=True,
    color="k",
    lw=1,
    label="Could be a contour of interaction",
)
a.display(ax, color="r", linewidth=0.75, label="Anticyclonic", ref=-10)
c.display(ax, color="b", linewidth=0.75, label="Cyclonic", ref=-10)
ax.legend()
update_axes(ax)
```



3.7.5 Output

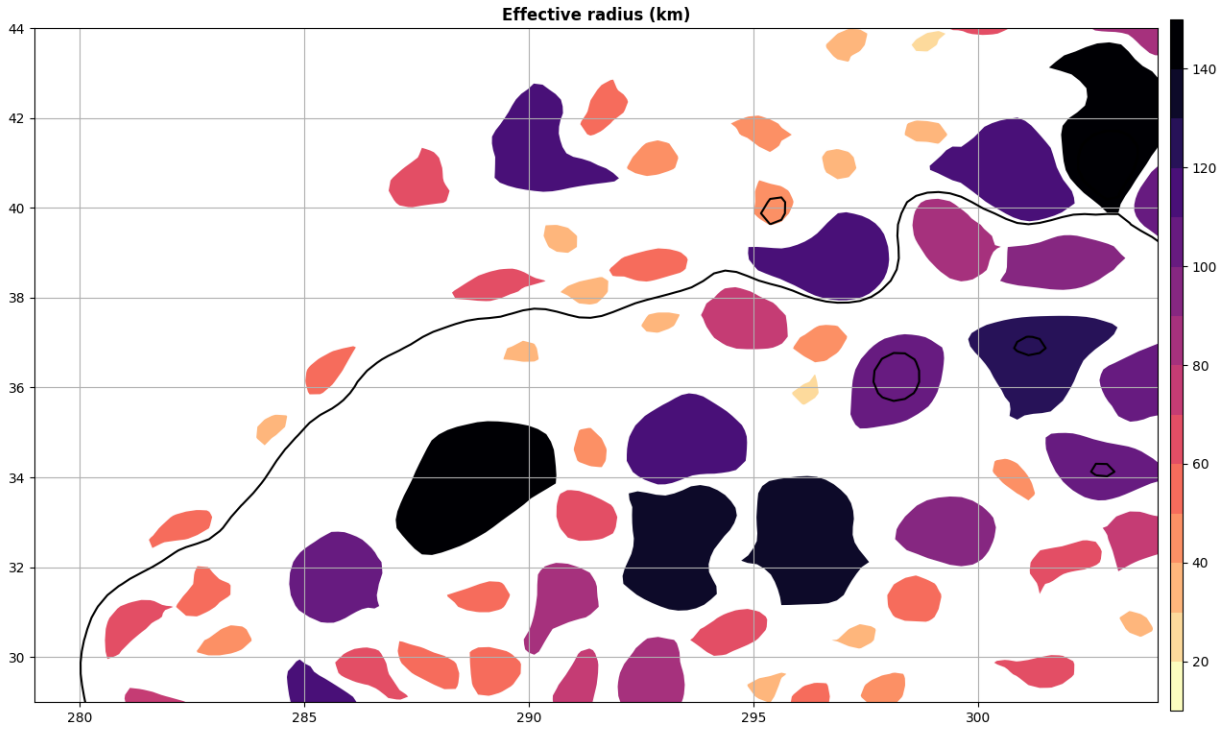
When displaying the detected eddies, dashed lines are for effective contour, solide lines for the contour of the maximum mean speed. See figure 1 of https://doi.org/10.1175/JTECH-D-14-00019.1

```
ax = start_axes("Eddies detected")
a.display(
    ax, color="r", linewidth=0.75, label="Anticyclonic ({nb_obs} eddies)", ref=-10
)
c.display(ax, color="b", linewidth=0.75, label="Cyclonic ({nb_obs} eddies)", ref=-10)
ax.legend()
great_current.display(ax, color="k")
update_axes(ax)
```



Display the effective radius of the detected eddies

```
ax = start_axes("Effective radius (km)")
a.filled(ax, "radius_e", vmin=10, vmax=150, cmap="magma_r", factor=0.001, lut=14)
m = c.filled(ax, "radius_e", vmin=10, vmax=150, cmap="magma_r", factor=0.001, lut=14)
great_current.display(ax, color="k")
update_axes(ax, m)
```



Total running time of the script: (0 minutes 10.348 seconds)

3.8 Eddy detection and filter

```
from datetime import datetime

from matplotlib import pyplot as plt
from numpy import arange

from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset
```

```
def start_axes(title):
    fig = plt.figure(figsize=(13, 5))
    ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
    ax.set_aspect("equal")
    ax.set_title(title, weight="bold")
    return ax


def update_axes(ax, mappable=None):
    ax.grid()
    if mappable:
        plt.colorbar(mappable, cax=ax.figure.add_axes([0.94, 0.05, 0.01, 0.9]))
```

Load Input grid, ADT is used to detect eddies. Add a new filed to store the high-pass filtered ADT

```
g = RegularGridDataset(
    data.get_path("dt_med_allsat_phy_l4_20160515_20190101.nc"), "longitude", "latitude
↪"
)
g.add_uv("adt")
g.copy("adt", "adt_high")
wavelength = 800
g.bessel_high_filter("adt_high", wavelength)
date = datetime(2016, 5, 15)
```

Out:

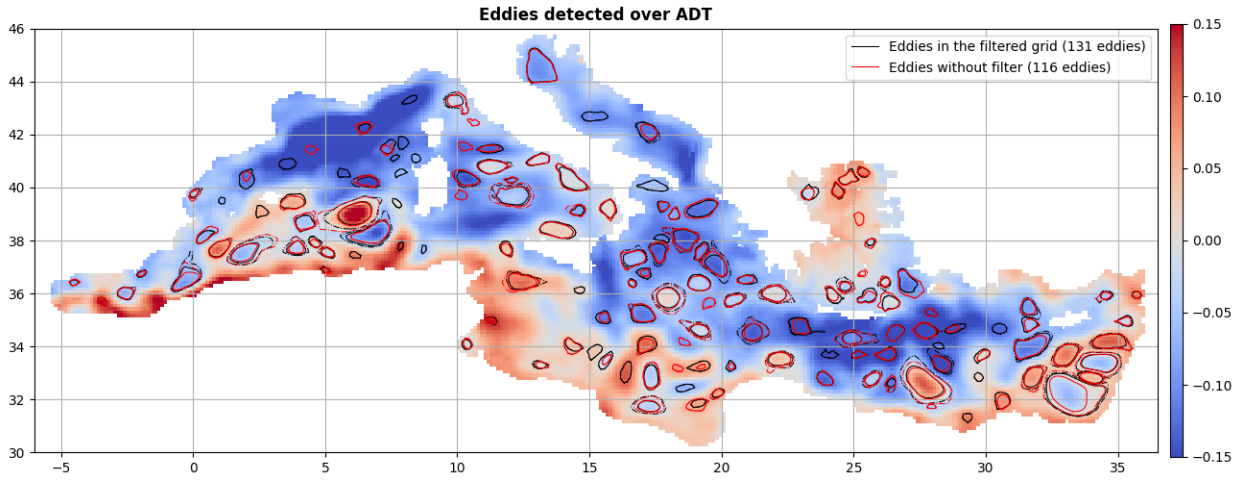
```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

Run the detection for the total grid and the filtered grid

```
a_filtered, c_filtered = g.eddy_identification("adt_high", "u", "v", date, 0.002)
merge_f = a_filtered.merge(c_filtered)
a_tot, c_tot = g.eddy_identification("adt", "u", "v", date, 0.002)
merge_t = a_tot.merge(c_tot)
```

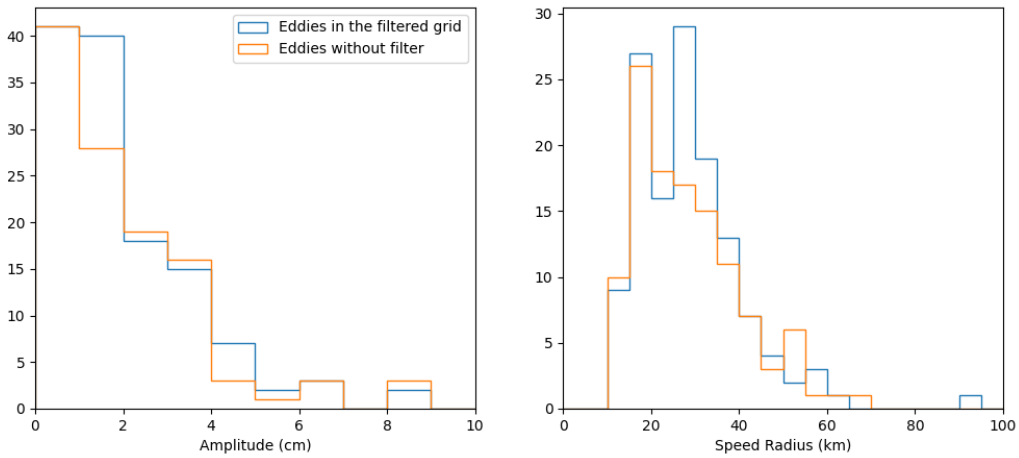
Display the two detections

```
ax = start_axes("Eddies detected over ADT")
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15)
merge_f.display(
    ax,
    lw=0.75,
    label="Eddies in the filtered grid ({nb_obs} eddies)",
    ref=-10,
    color="k",
)
merge_t.display(
    ax, lw=0.75, label="Eddies without filter ({nb_obs} eddies)", ref=-10, color="r"
)
ax.legend()
update_axes(ax, m)
```



3.8.1 Amplitude and Speed Radius distributions

```
fig = plt.figure(figsize=(12, 5))
ax_a = fig.add_subplot(121, xlabel="Amplitude (cm)")
ax_r = fig.add_subplot(122, xlabel="Speed Radius (km)")
ax_a.hist(
    merge_f.amplitude * 100,
    bins=arange(0.0005, 100, 1),
    label="Eddies in the filtered grid",
    histtype="step",
)
ax_a.hist(
    merge_t.amplitude * 100,
    bins=arange(0.0005, 100, 1),
    label="Eddies without filter",
    histtype="step",
)
ax_a.set_xlim(0, 10)
ax_r.hist(merge_f.radius_s / 1000.0, bins=arange(0, 300, 5), histtype="step")
ax_r.hist(merge_t.radius_s / 1000.0, bins=arange(0, 300, 5), histtype="step")
ax_r.set_xlim(0, 100)
ax_a.legend()
```



3.8.2 Match detection and compare

```
i_, j_, c = merge_f.match(merge_t, cmin=0.1)
```

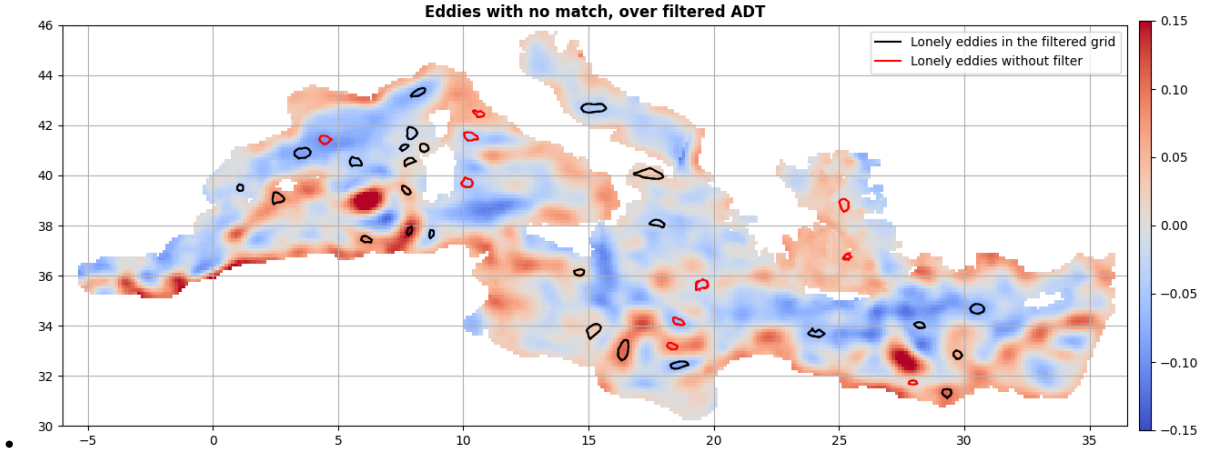
Where are the lonely eddies?

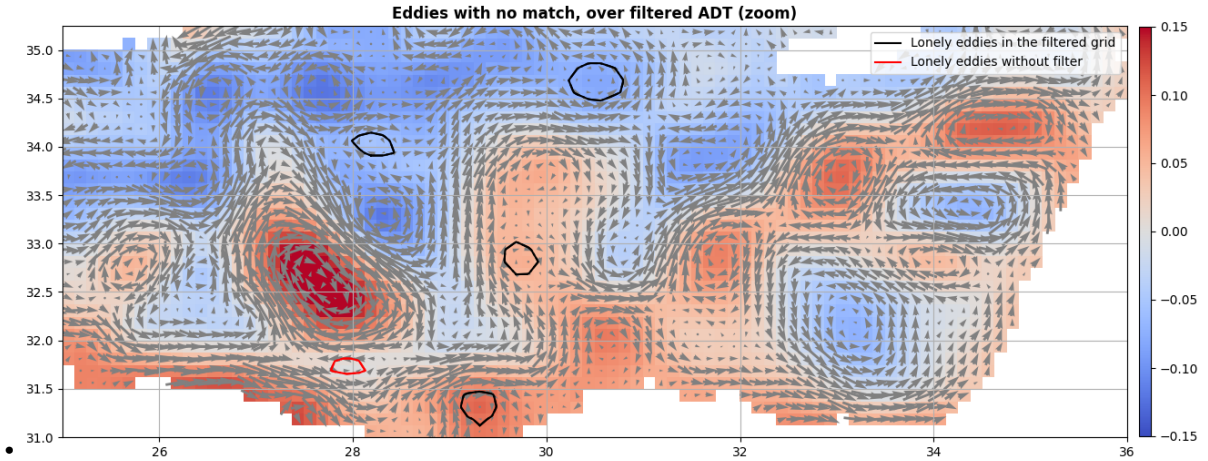
```
kwargs_f = dict(lw=1.5, label="Lonely eddies in the filtered grid", ref=-10, color="k
↪")
kwargs_t = dict(lw=1.5, label="Lonely eddies without filter", ref=-10, color="r")
ax = start_axes("Eddies with no match, over filtered ADT")
mappable = g.display(ax, "adt_high", vmin=-0.15, vmax=0.15)
merge_f.index(i_, reverse=True).display(ax, **kwargs_f)
merge_t.index(j_, reverse=True).display(ax, **kwargs_t)
ax.legend()
```

(continues on next page)

(continued from previous page)

```
update_axes(ax, mappable)

ax = start_axes("Eddies with no match, over filtered ADT (zoom)")
ax.set_xlim(25, 36), ax.set_ylim(31, 35.25)
mappable = g.display(ax, "adt_high", vmin=-0.15, vmax=0.15)
u, v = g.grid("u").T, g.grid("v").T
ax.quiver(g.x_c, g.y_c, u, v, scale=10, pivot="mid", color="gray")
merge_f.index(i_, reverse=True).display(ax, **kwargs_f)
merge_t.index(j_, reverse=True).display(ax, **kwargs_t)
ax.legend()
update_axes(ax, mappable)
```





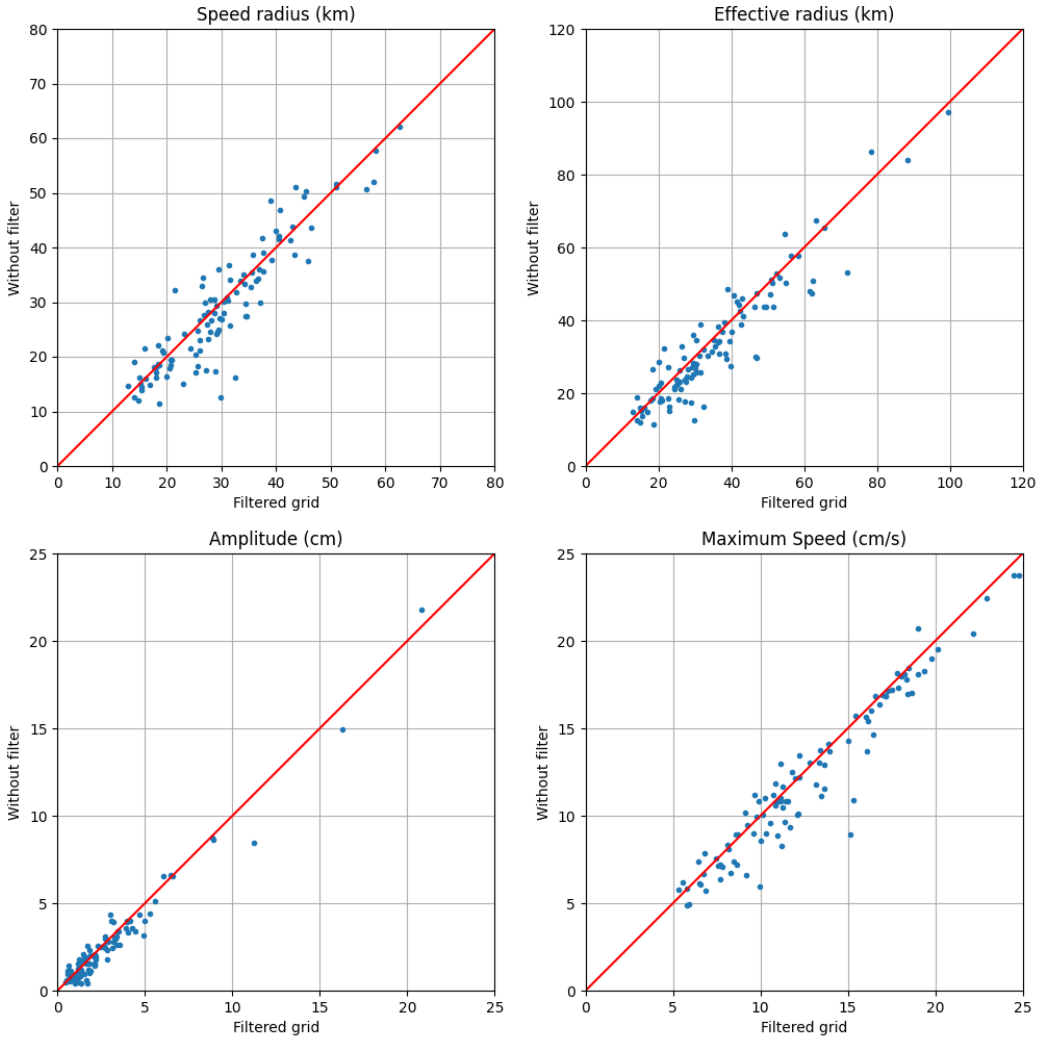

```
fig = plt.figure(figsize=(12, 12))
fig.suptitle(f"Scatter plot ({i_.shape[0]} matches)", weight="bold")

for i, (label, field, factor, stop) in enumerate(
    (
        ("Speed radius (km)", "radius_s", 0.001, 80),
        ("Effective radius (km)", "radius_e", 0.001, 120),
        ("Amplitude (cm)", "amplitude", 100, 25),
        ("Maximum Speed (cm/s)", "speed_average", 100, 25),
    )
):
    ax = fig.add_subplot(
```

(continues on next page)

(continued from previous page)

```
        2, 2, i + 1, xlabel="Filtered grid", ylabel="Without filter", title=label
    )
    ax.plot(merge_f[field][i_] * factor, merge_t[field][j_] * factor, ".")
    ax.set_aspect("equal"), ax.grid()
    ax.plot((0, 1000), (0, 1000), "r")
    ax.set_xlim(0, stop), ax.set_ylim(0, stop)
```

Scatter plot (106 matches)



Total running time of the script: (0 minutes 8.243 seconds)

3.9 Eddy detection on SLA and ADT

```
from datetime import datetime

from matplotlib import pyplot as plt

from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset
```

```
def start_axes(title):
    fig = plt.figure(figsize=(13, 5))
    ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
    ax.set_aspect("equal")
    ax.set_title(title)
    return ax


def update_axes(ax, mappable=None):
    ax.grid()
    if mappable:
        plt.colorbar(mappable, cax=ax.figure.add_axes([0.95, 0.05, 0.01, 0.9]))
```

Load Input grid, ADT will be used to detect eddies

```
g = RegularGridDataset(
    data.get_path("dt_med_allsat_phy_l4_20160515_20190101.nc"), "longitude", "latitude
↪"
)
g.add_uv("adt", "ugos", "vgos")
g.add_uv("sla", "ugosa", "vgosa")
wavelength = 400
g.copy("adt", "adt_raw")
g.copy("sla", "sla_raw")
g.bessel_high_filter("adt", wavelength)
g.bessel_high_filter("sla", wavelength)
date = datetime(2016, 5, 15)
```

Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

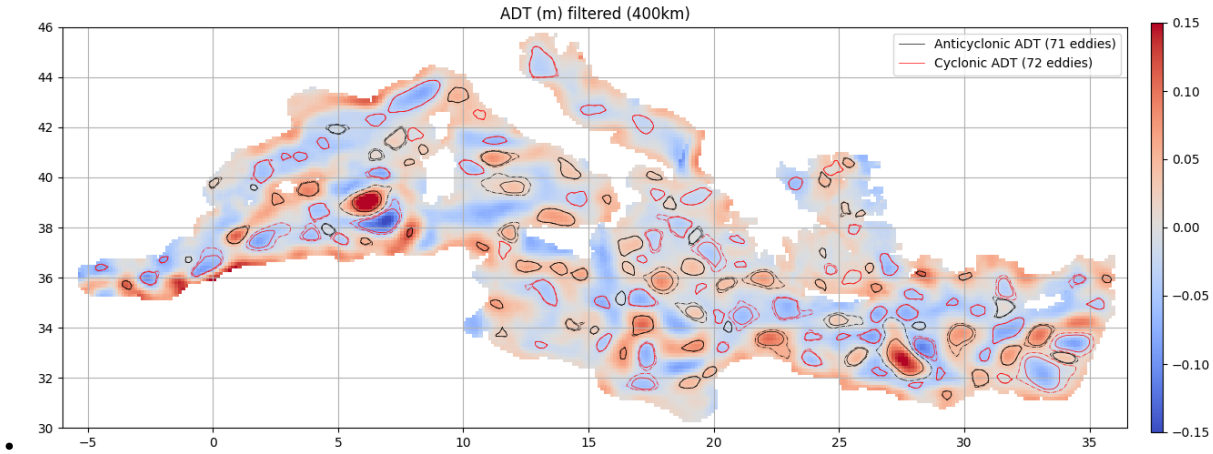
```
kwargs_a_adt = dict(
    lw=0.5, label="Anticyclonic ADT ({nb_obs} eddies)", ref=-10, color="k"
)
kwargs_c_adt = dict(lw=0.5, label="Cyclonic ADT ({nb_obs} eddies)", ref=-10, color="r
↪")
kwargs_a_sla = dict(
    lw=0.5, label="Anticyclonic SLA ({nb_obs} eddies)", ref=-10, color="g"
)
kwargs_c_sla = dict(lw=0.5, label="Cyclonic SLA ({nb_obs} eddies)", ref=-10, color="b
↪")
```

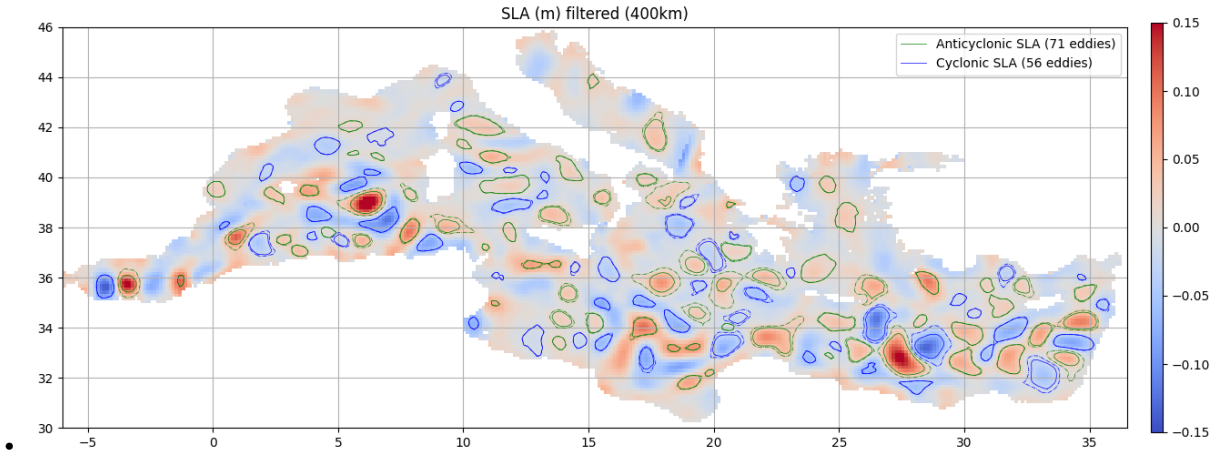
Run algorithm of detection

```
a_adt, c_adt = g.eddy_identification("adt", "ugos", "vgos", date, 0.002)
a_sla, c_sla = g.eddy_identification("sla", "ugosa", "vgosa", date, 0.002)
```

over filtered

```
ax = start_axes(f"ADT (m) filtered ({wavelength}km)")
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15)
a_adt.display(ax, **kwargs_a_adt), c_adt.display(ax, **kwargs_c_adt)
ax.legend(), update_axes(ax, m)

ax = start_axes(f"SLA (m) filtered ({wavelength}km)")
m = g.display(ax, "sla", vmin=-0.15, vmax=0.15)
a_sla.display(ax, **kwargs_a_sla), c_sla.display(ax, **kwargs_c_sla)
ax.legend(), update_axes(ax, m)
```

- 

- 

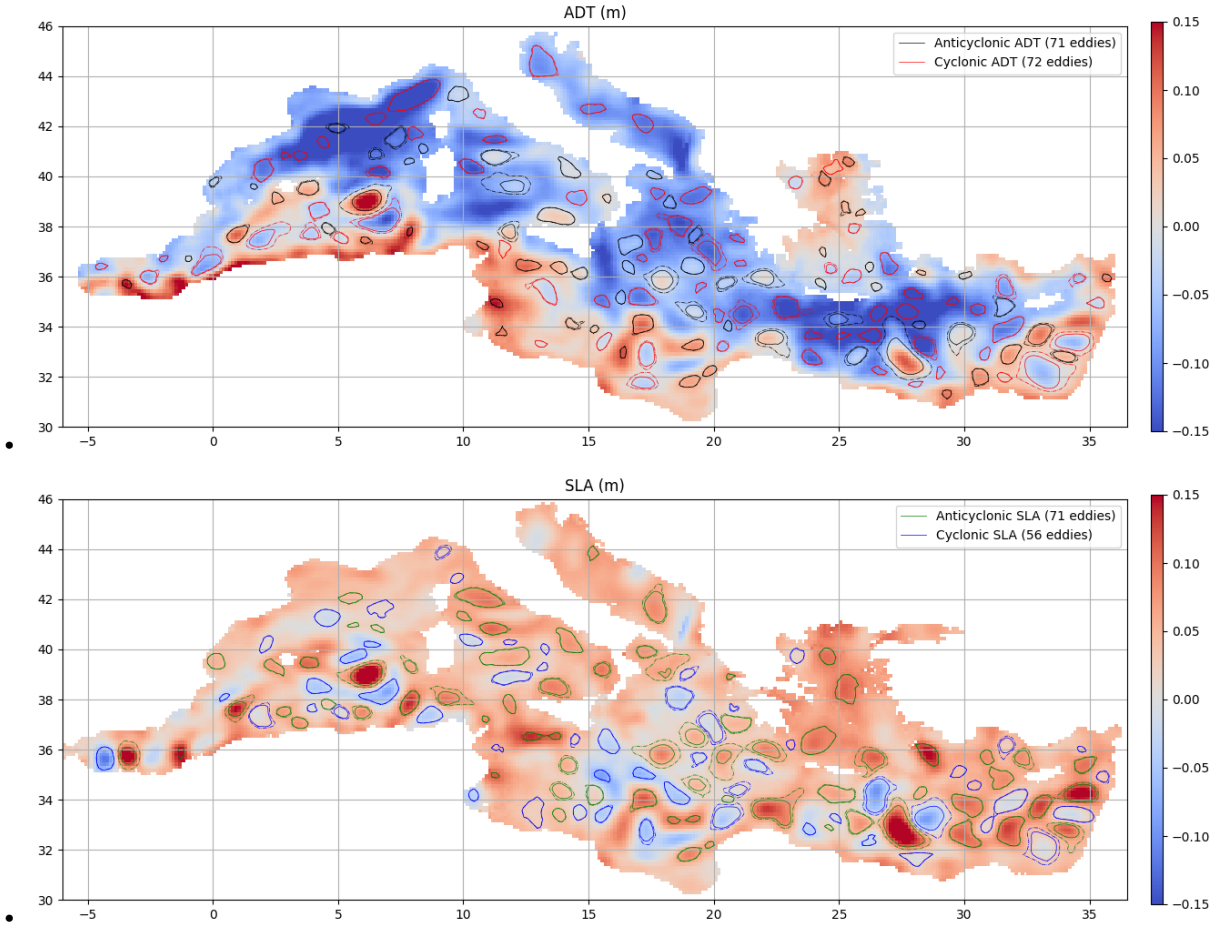
over raw

```
ax = start_axes("ADT (m)")
m = g.display(ax, "adt_raw", vmin=-0.15, vmax=0.15)
a_adt.display(ax, **kwargs_a_adt), c_adt.display(ax, **kwargs_c_adt)
ax.legend(), update_axes(ax, m)

ax = start_axes("SLA (m)")
m = g.display(ax, "sla_raw", vmin=-0.15, vmax=0.15)
a_sla.display(ax, **kwargs_a_sla), c_sla.display(ax, **kwargs_c_sla)
```

(continues on next page)

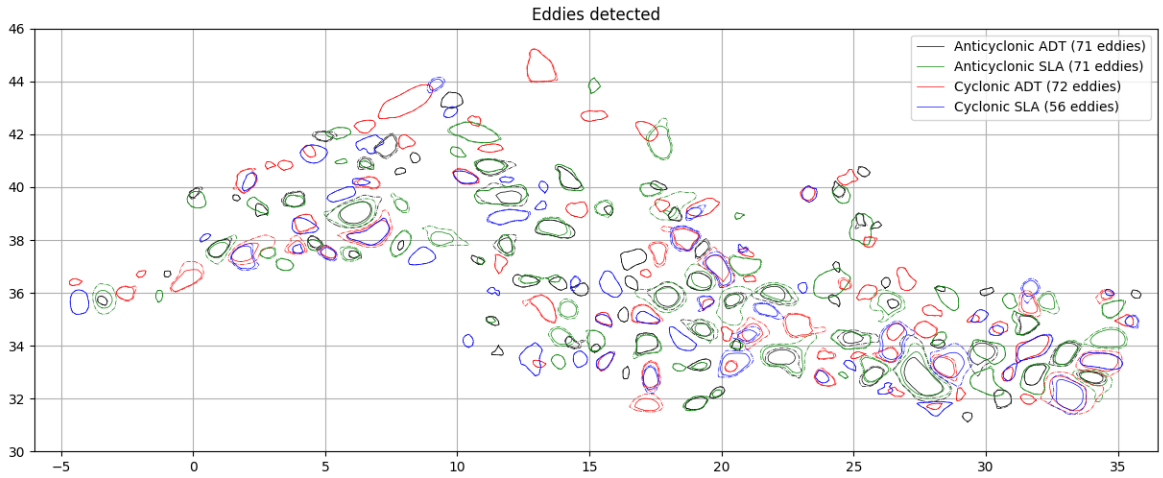
(continued from previous page)

```
ax.legend(), update_axes(ax, m)
```



Display detection

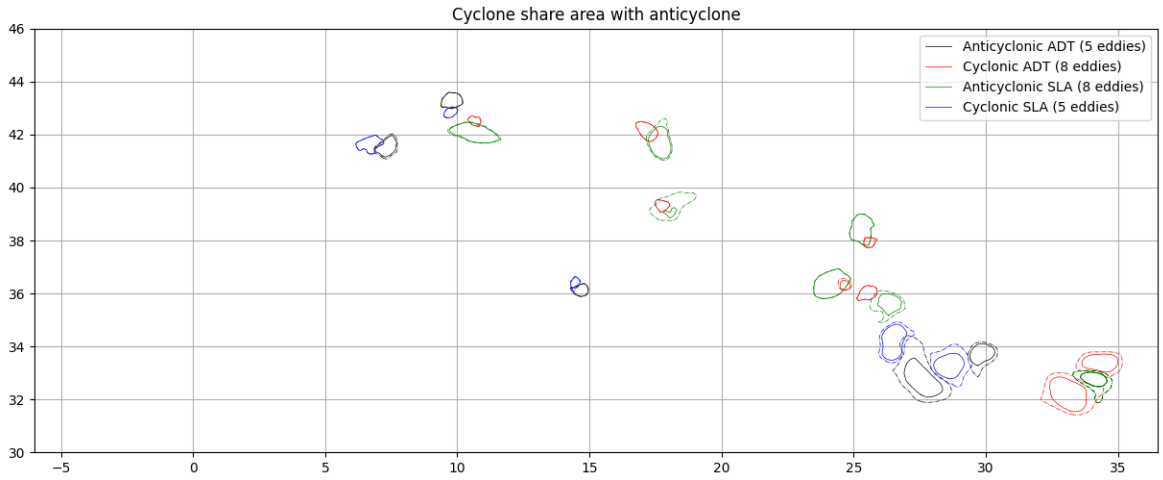
```
ax = start_axes("Eddies detected")
a_adt.display(ax, **kwargs_a_adt)
a_sla.display(ax, **kwargs_a_sla)
c_adt.display(ax, **kwargs_c_adt)
c_sla.display(ax, **kwargs_c_sla)
ax.legend()
update_axes(ax)
```



3.9.1 Match

Where cyclone meet anticyclone

```
i_c_adt, i_a_sla, c = c_adt.match(a_sla, cmin=0.01)
i_a_adt, i_c_sla, c = a_adt.match(c_sla, cmin=0.01)

ax = start_axes("Cyclone share area with anticyclone")
a_adt.index(i_a_adt).display(ax, **kwargs_a_adt)
c_adt.index(i_c_adt).display(ax, **kwargs_c_adt)
a_sla.index(i_a_sla).display(ax, **kwargs_a_sla)
c_sla.index(i_c_sla).display(ax, **kwargs_c_sla)
ax.legend()
update_axes(ax)
```

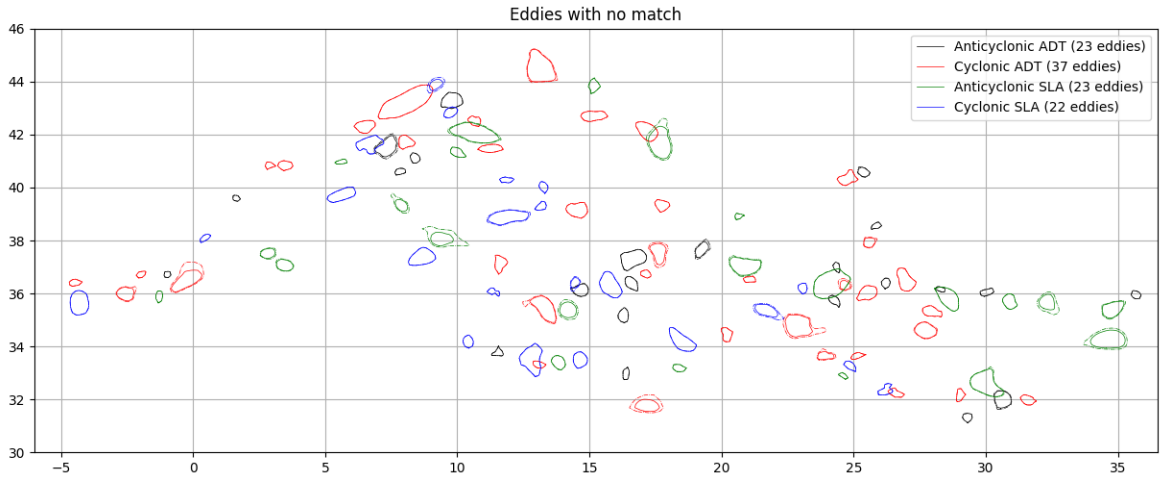


3.9.2 Scatter plot

```
i_a_adt, i_a_sla, c = a_adt.match(a_sla, cmin=0.1)
i_c_adt, i_c_sla, c = c_adt.match(c_sla, cmin=0.1)
```

where is lonely eddies

```
ax = start_axes("Eddies with no match")
a_adt.index(i_a_adt, reverse=True).display(ax, **kwargs_a_adt)
c_adt.index(i_c_adt, reverse=True).display(ax, **kwargs_c_adt)
a_sla.index(i_a_sla, reverse=True).display(ax, **kwargs_a_sla)
c_sla.index(i_c_sla, reverse=True).display(ax, **kwargs_c_sla)
ax.legend()
update_axes(ax)
```

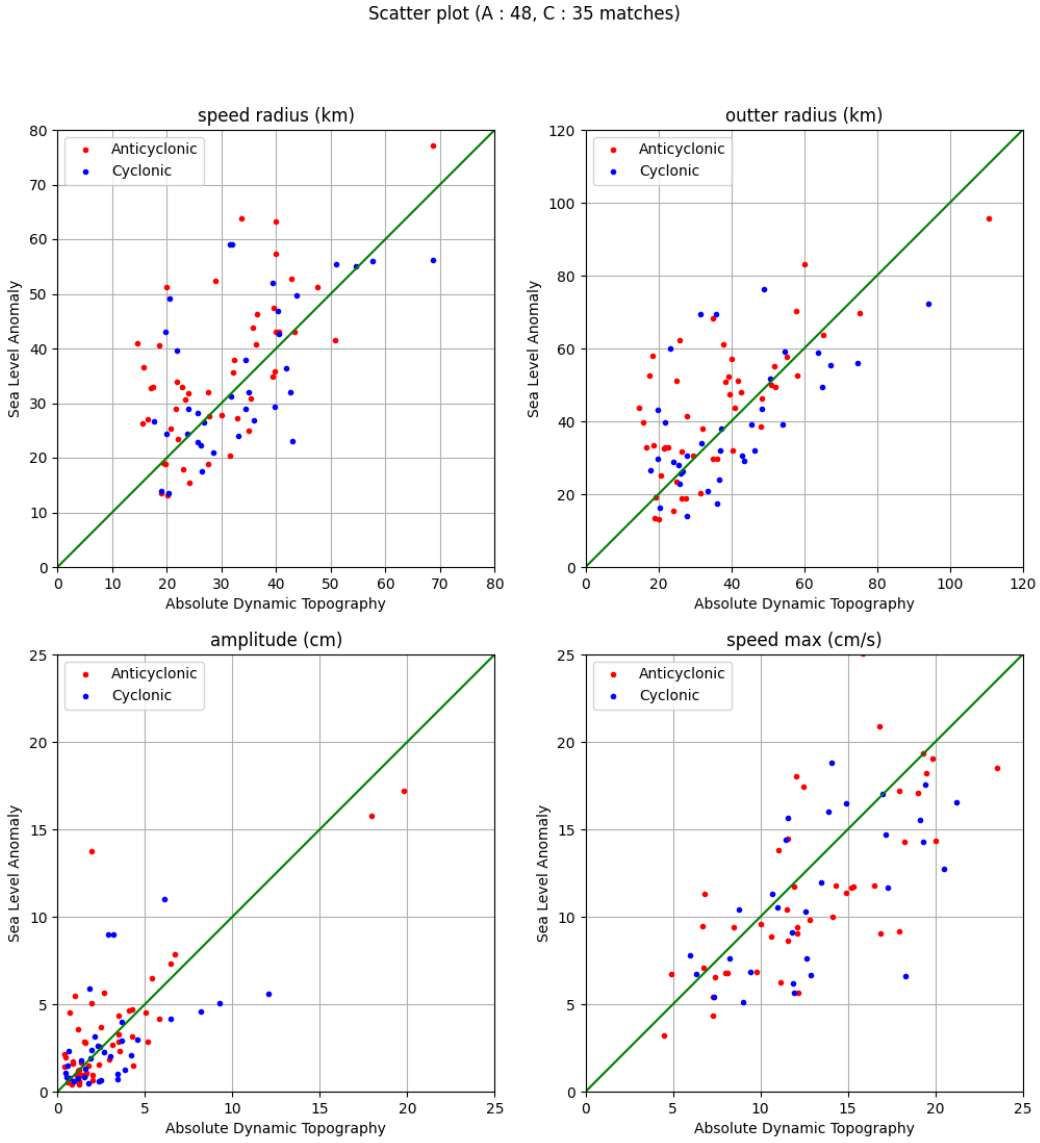



```
fig = plt.figure(figsize=(12, 12))
fig.suptitle(f"Scatter plot (A : {i_a_adt.shape[0]}, C : {i_c_adt.shape[0]} matches)")

for i, (label, field, factor, stop) in enumerate(
    (
        ("speed radius (km)", "radius_s", 0.001, 80),
        ("outter radius (km)", "radius_e", 0.001, 120),
        ("amplitude (cm)", "amplitude", 100, 25),
        ("speed max (cm/s)", "speed_average", 100, 25),
    )
):
    ax = fig.add_subplot(2, 2, i + 1, title=label)
    ax.set_xlabel("Absolute Dynamic Topography")
    ax.set_ylabel("Sea Level Anomaly")

    ax.plot(
        a_adt[field][i_a_adt] * factor,
        a_sla[field][i_a_sla] * factor,
        "r.",
        label="Anticyclonic",
    )
    ax.plot(
        c_adt[field][i_c_adt] * factor,
        c_sla[field][i_c_sla] * factor,
        "b.",
```

(continues on next page)

(continued from previous page)

```
        label="Cyclonic",
    )
    ax.set_aspect("equal"), ax.grid()
    ax.plot((0, 1000), (0, 1000), "g")
    ax.set_xlim(0, stop), ax.set_ylim(0, stop)
    ax.legend()
```



Total running time of the script: (0 minutes 6.418 seconds)

CHAPTER

FOUR

GRID MANIPULATION

4.1 Select pixel in eddies

```
from matplotlib import pyplot as plt
from matplotlib.path import Path
from numpy import ones

from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker.observations.observation import EddiesObservations
from py_eddy_tracker.poly import create_vertice
```

Load an eddy file which contains contours

```
a = EddiesObservations.load_file(data.get_path("Anticyclonic_20190223.nc"))
```

Load a grid where we want found pixels in eddies or out

```
g = RegularGridDataset(
    data.get_path("nrt_global_allsat_phy_l4_20190223_20190226.nc"),
    "longitude",
    "latitude",
)
```

Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/nrt_global_allsat_phy_l4_20190223_20190226.nc
```

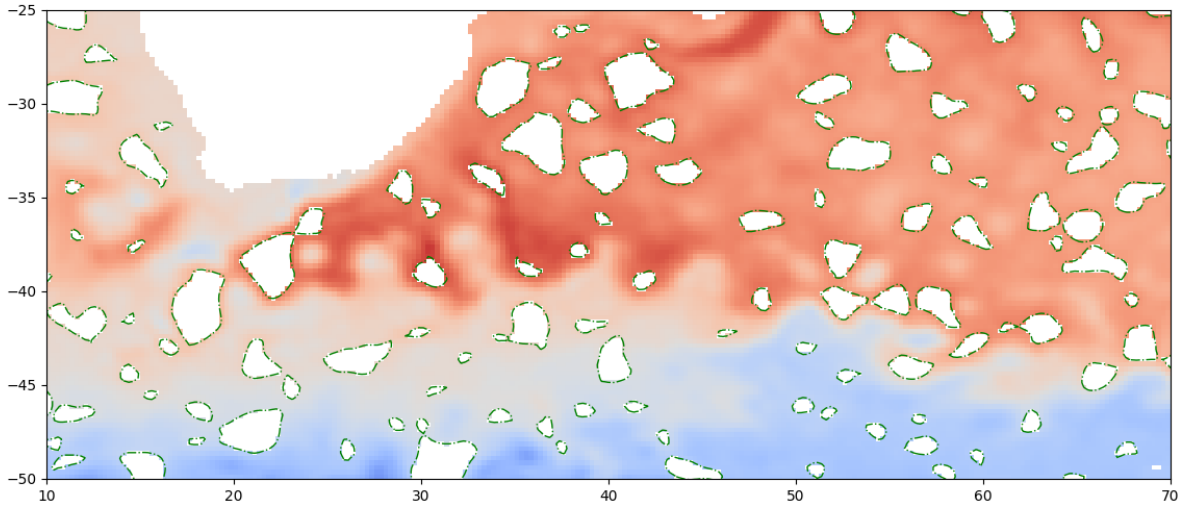
For each contours, we will get pixels indice in contour.

```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_axes((0.05, 0.05, 0.9, 0.9))
ax.set_aspect("equal")
ax.set_xlim(10, 70)
ax.set_ylim(-50, -25)
# We will used the outter contour
x_name, y_name = a.intern(False)
adt = g.grid("adt")
mask = ones(adt.shape, dtype="bool")
for eddy in a:
    i, j = Path(create_vertice(eddy[x_name], eddy[y_name])).pixels_in(g)
```

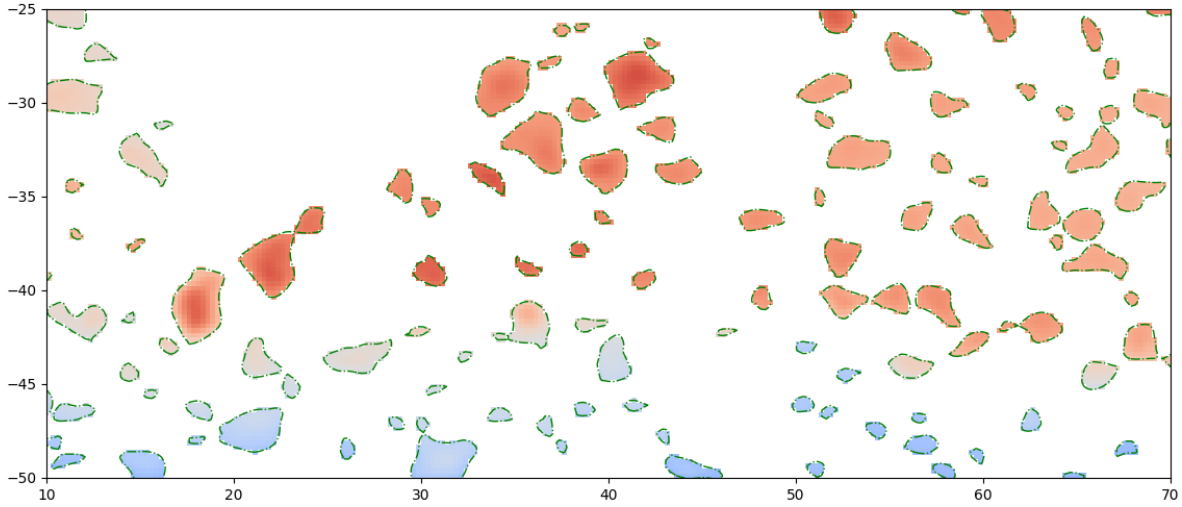
(continues on next page)

(continued from previous page)

```
    mask[i, j] = False
adt.mask[:] += ~mask
g.display(ax, "adt")
a.display(ax, label="Anticyclonic", color="g", lw=1, extern_only=True)
```




```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_axes((0.05, 0.05, 0.9, 0.9))
ax.set_aspect("equal")
ax.set_xlim(10, 70)
ax.set_ylim(-50, -25)
adt.mask[:] = mask
g.display(ax, "adt")
a.display(ax, label="Anticyclonic", color="g", lw=1, extern_only=True)
```



Total running time of the script: (0 minutes 1.355 seconds)

4.2 Grid filtering in PET

How filter work in py eddy tracker. This implementation maybe doesn't respect state art, but ...

We code a specific filter in order to filter grid with same wavelength at each pixel.

```
from matplotlib import pyplot as plt
from numpy import arange

from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset


def start_axes(title):
    fig = plt.figure(figsize=(13, 5))
    ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
    ax.set_aspect("equal")
    ax.set_title(title)
    return ax


def update_axes(ax, mappable=None):
    ax.grid()
    if mappable:
        plt.colorbar(mappable, cax=ax.figure.add_axes([0.95, 0.05, 0.01, 0.9]))
```

All information will be for regular grid

```
g = RegularGridDataset(
    data.get_path("dt_med_allsat_phy_l4_20160515_20190101.nc"), "longitude", "latitude
↪"
)
```

Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

4.2.1 Kernel

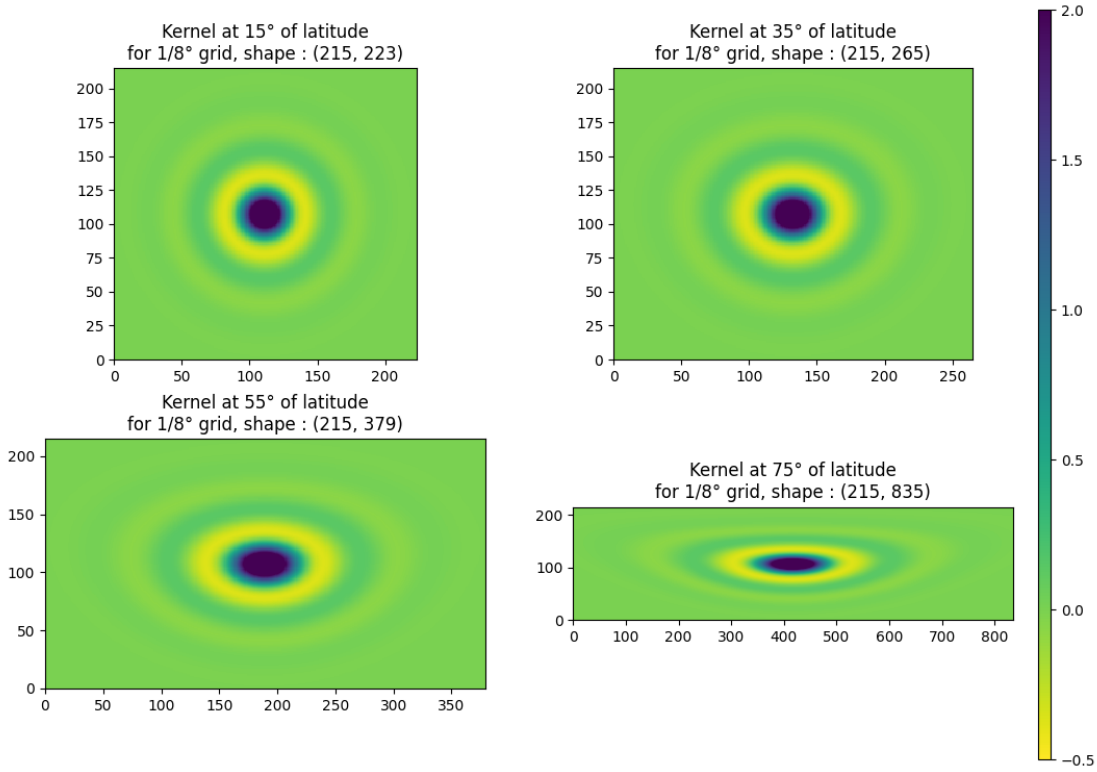
Shape of kernel will increase in x, when latitude increase

```
fig = plt.figure(figsize=(12, 8))
for i, latitude in enumerate((15, 35, 55, 75)):
    k = g.kernel_bessel(latitude, 500, order=3).T
    ax0 = fig.add_subplot(
        2,
        2,
        i + 1,
        title=f"Kernel at {latitude}° of latitude\nfor 1/8° grid, shape : {k.shape}",
```

(continues on next page)

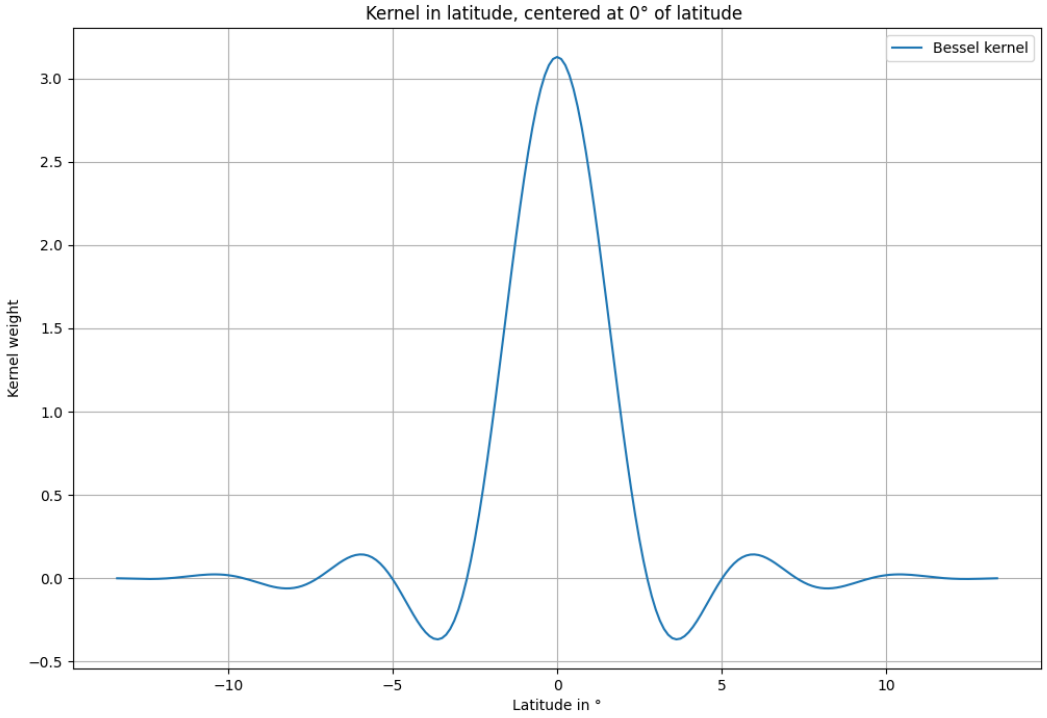
(continued from previous page)

```
        aspect="equal",
    )
    m = ax0.pcolormesh(k, vmin=-0.5, vmax=2, cmap="viridis_r")
plt.colorbar(m, cax=fig.add_axes((0.92, 0.05, 0.01, 0.9)))
```



Kernel along latitude

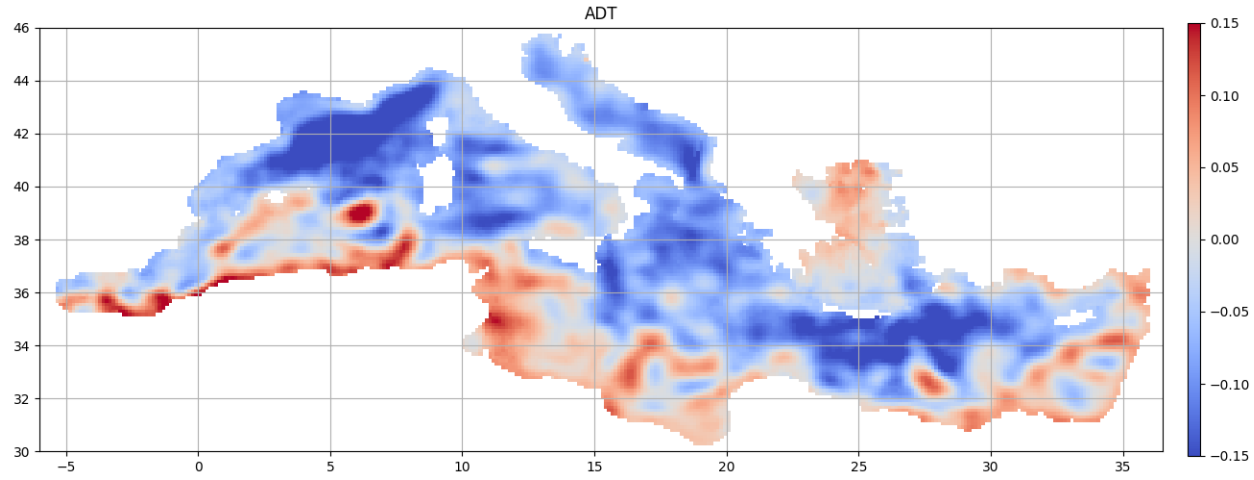
```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(
    111,
    ylabel="Kernel weight",
    xlabel="Latitude in °",
    title="Kernel in latitude, centered at 0° of latitude ",
)
k = g.kernel_bessel(0, 500, order=3)
k_lat = k[k.shape[0] // 2 + 1]
nb = k_lat.shape[0] // 2
ax.plot(
    arange(-nb * g.xstep, (nb + 0.5) * g.xstep, g.xstep), k_lat, label="Bessel kernel"
)

ax.legend()
ax.grid()
```



4.2.2 Kernel applying

Original grid

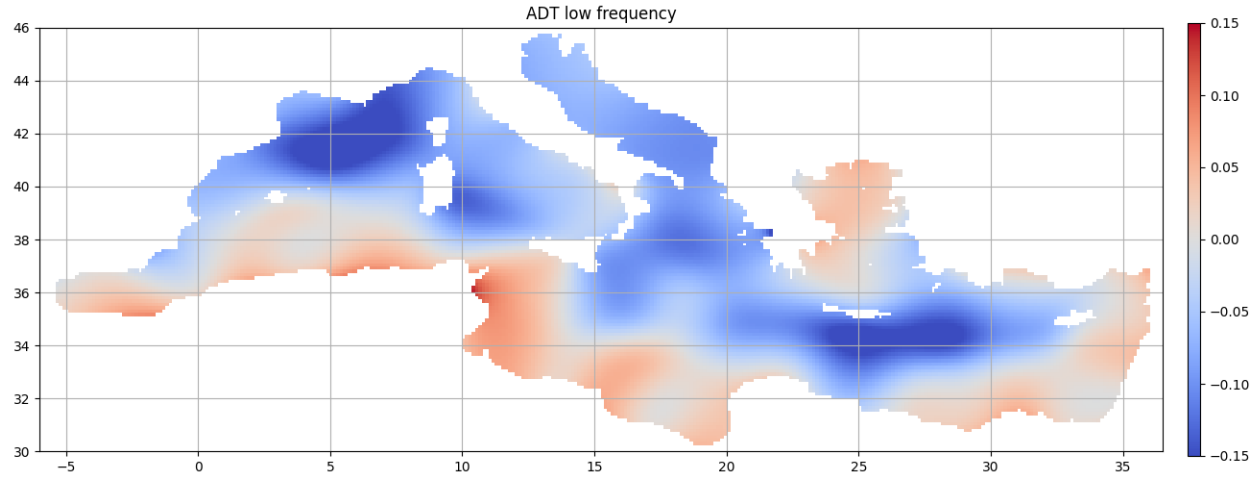
```
ax = start_axes("ADT")
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15)
update_axes(ax, m)
```



We will select wavelength of 300 km

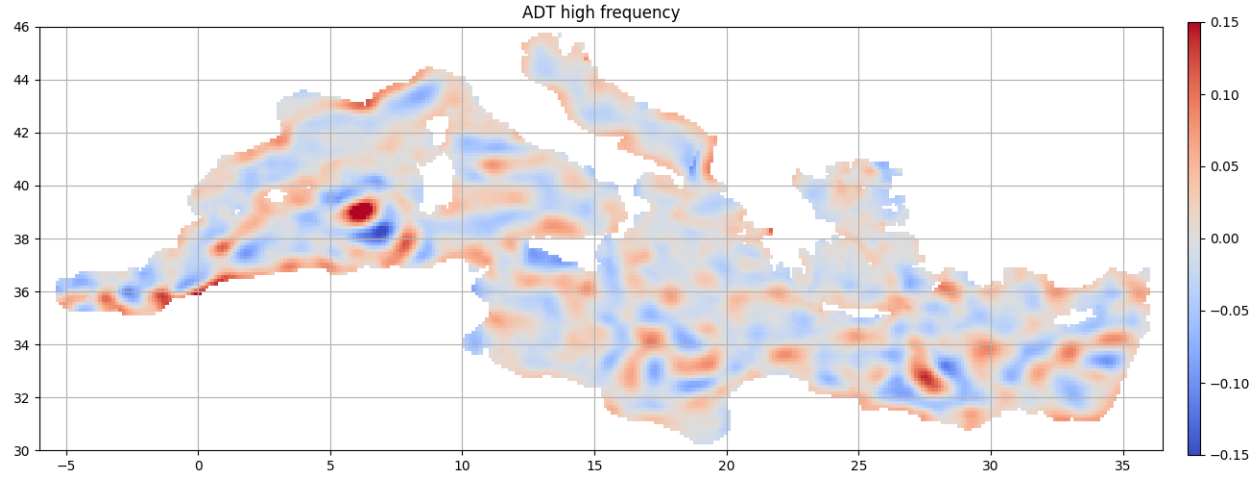
Low frequency

```
ax = start_axes("ADT low frequency")
g.copy("adt", "adt_low_300")
g.bessel_low_filter("adt_low_300", 300, order=3)
m = g.display(ax, "adt_low_300", vmin=-0.15, vmax=0.15)
update_axes(ax, m)
```



High frequency

```
ax = start_axes("ADT high frequency")
g.copy("adt", "adt_high_300")
g.bessel_high_filter("adt_high_300", 300, order=3)
m = g.display(ax, "adt_high_300", vmin=-0.15, vmax=0.15)
update_axes(ax, m)
```



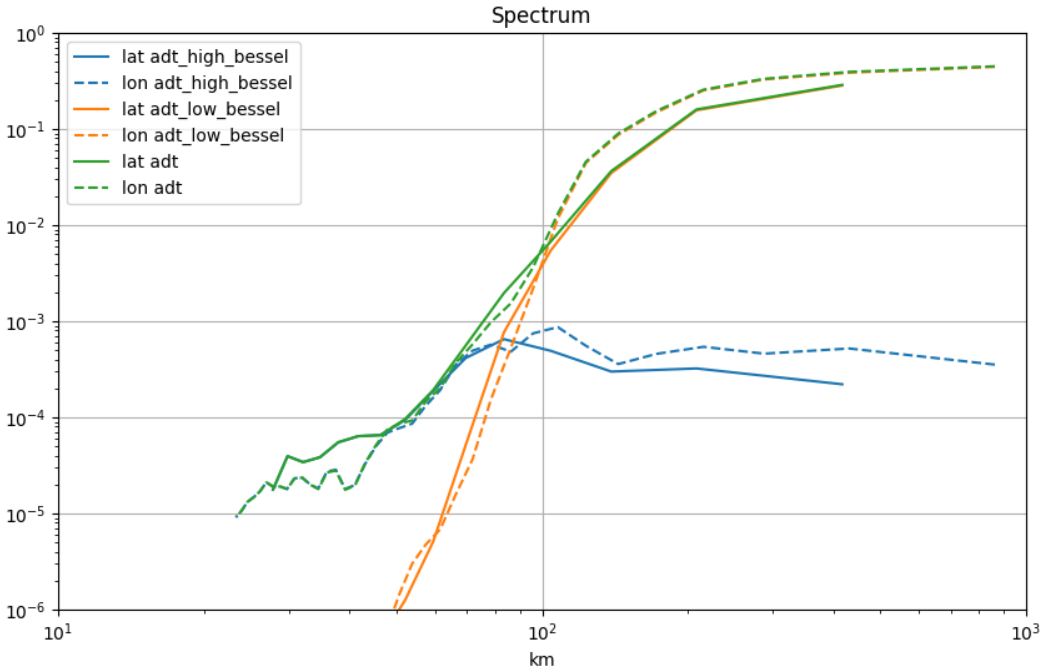
4.2.3 Clues

wavelength : 80km

```
g.copy("adt", "adt_high_bessel")
g.bessel_high_filter("adt_high_bessel", 80, order=3)
g.copy("adt", "adt_low_bessel")
g.bessel_low_filter("adt_low_bessel", 80, order=3)

area = dict(llcrnrlon=11.75, urcrnrlon=21, llcrnrlat=33, urcrnrlat=36.75)
```

Spectrum

```
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)
ax.set_title("Spectrum")
ax.set_xlabel("km")

for label in ("adt_high_bessel", "adt_low_bessel", "adt"):
    lon_spec, lat_spec = g.spectrum_lonlat(label, area=area)
    mappable = ax.loglog(*lat_spec, label=f"lat {label}")[0]
    ax.loglog(
        *lon_spec, label=f"lon {label}", color=mappable.get_color(), linestyle="--"
    )

ax.set_xlim(10, 1000)
ax.set_ylim(1e-6, 1)
ax.set_xscale("log")
ax.legend()
ax.grid()
```

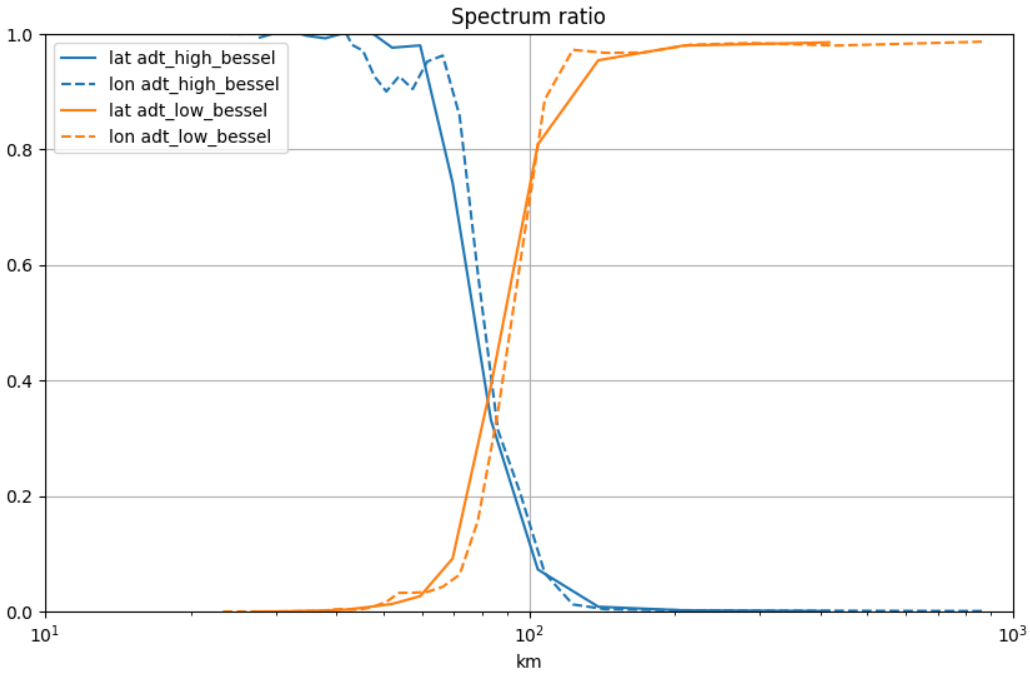


Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 30, using nperseg = 30
  .format(nperseg, input_length))
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 74, using nperseg = 74
  .format(nperseg, input_length))
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 30, using nperseg = 30
  .format(nperseg, input_length))
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 74, using nperseg = 74
  .format(nperseg, input_length))
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 30, using nperseg = 30
  .format(nperseg, input_length))
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 74, using nperseg = 74
  .format(nperseg, input_length))
```

Spectrum ratio

```
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)
ax.set_title("Spectrum ratio")
ax.set_xlabel("km")

for label in ("adt_high_bessel", "adt_low_bessel"):
    lon_spec, lat_spec = g.spectrum_lonlat(label, area=area, ref=g, ref_grid_name="adt
↪")
    mappable = ax.plot(*lat_spec, label=f"lat {label}")[0]
    ax.plot(*lon_spec, label=f"lon {label}", color=mappable.get_color(), linestyle="--
↪")

ax.set_xlim(10, 1000)
ax.set_ylim(0, 1)
ax.set_xscale("log")
ax.legend()
ax.grid()
```



Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 30, using nperseg = 30
  .format(nperseg, input_length))
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 74, using nperseg = 74
  .format(nperseg, input_length))
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 30, using nperseg = 30
  .format(nperseg, input_length))
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 74, using nperseg = 74
  .format(nperseg, input_length))
```

4.2.4 Old filter

To do ...

Total running time of the script: (0 minutes 7.194 seconds)

4.3 Get Okubo Weis

$$OW = S_n^2 + S_s^2 + \omega^2$$

with normal strain (S_n), shear strain (S_s) and vorticity (ω)

$$S_n = \frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}, S_s = \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}, \omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

```
from matplotlib import pyplot as plt
from numpy import arange, ma, where

from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker.observations.observation import EddiesObservations
```

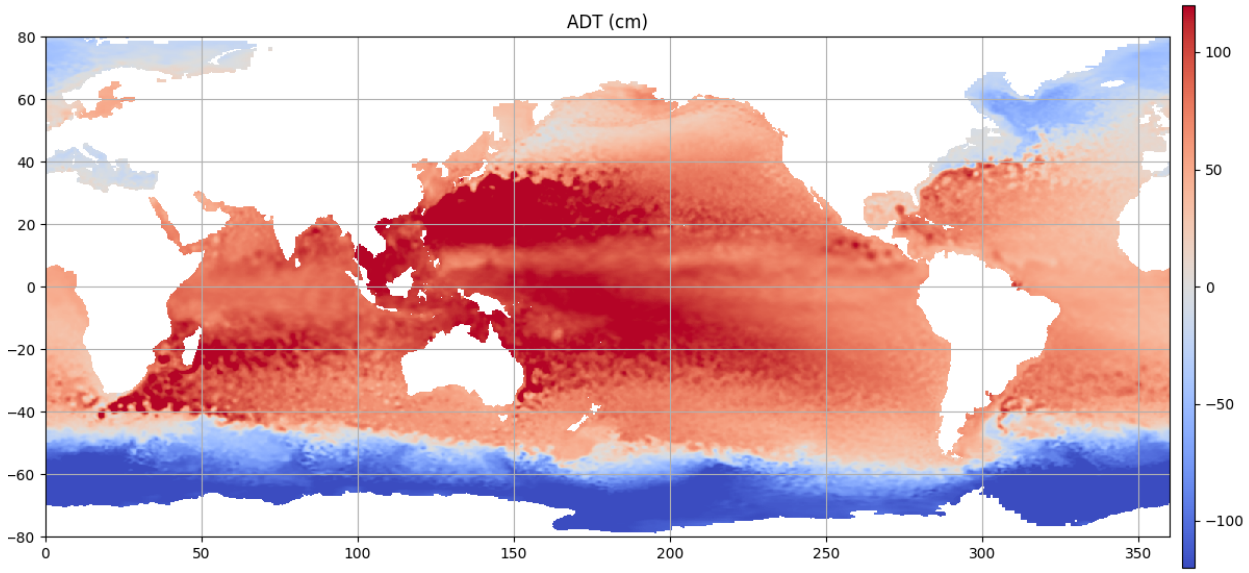
```
def start_axes(title, zoom=False):
    fig = plt.figure(figsize=(12, 6))
    axes = fig.add_axes([0.03, 0.03, 0.90, 0.94])
    axes.set_xlim(0, 360), axes.set_ylim(-80, 80)
    if zoom:
        axes.set_xlim(270, 340), axes.set_ylim(20, 50)
    axes.set_aspect("equal")
    axes.set_title(title)
    return axes


def update_axes(axes, mappable=None):
    axes.grid()
    if mappable:
        plt.colorbar(mappable, cax=axes.figure.add_axes([0.94, 0.05, 0.01, 0.9]))
```

Load detection files

```
a = EddiesObservations.load_file(data.get_path("Anticyclonic_20190223.nc"))
c = EddiesObservations.load_file(data.get_path("Cyclonic_20190223.nc"))
```

Load Input grid, ADT will be used to detect eddies

```
g = RegularGridDataset(
    data.get_path("nrt_global_allsat_phy_l4_20190223_20190226.nc"),
    "longitude",
    "latitude",
)

ax = start_axes("ADT (cm)")
m = g.display(ax, "adt", vmin=-120, vmax=120, factor=100)
update_axes(ax, m)
```

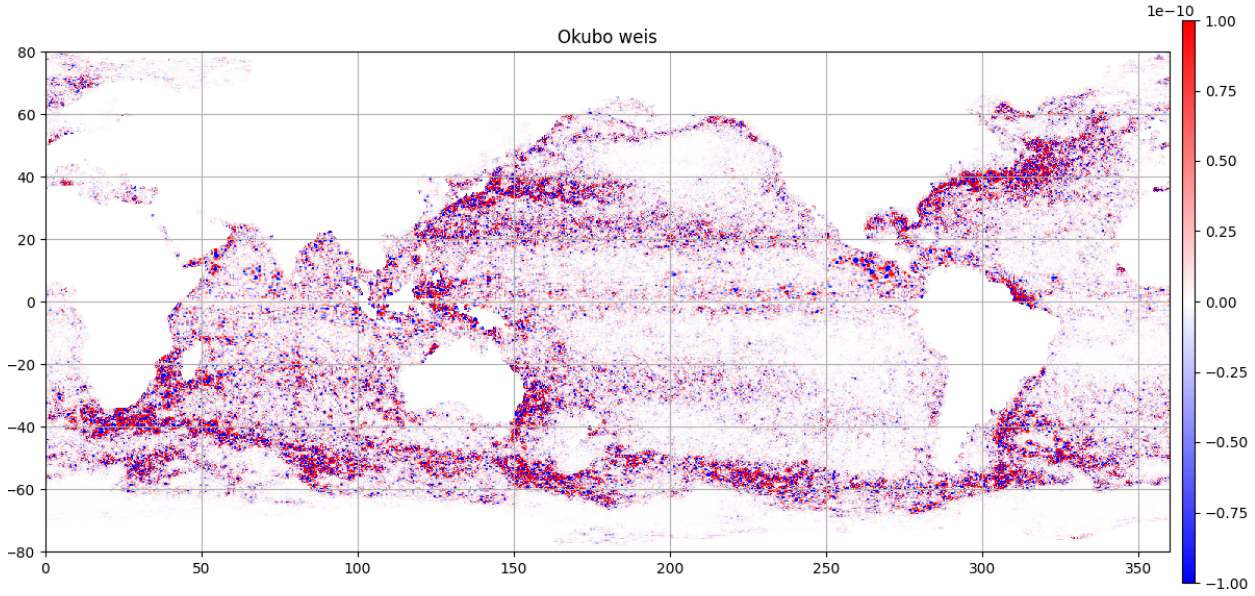


Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/nrt_global_allsat_phy_l4_20190223_20190226.nc
```

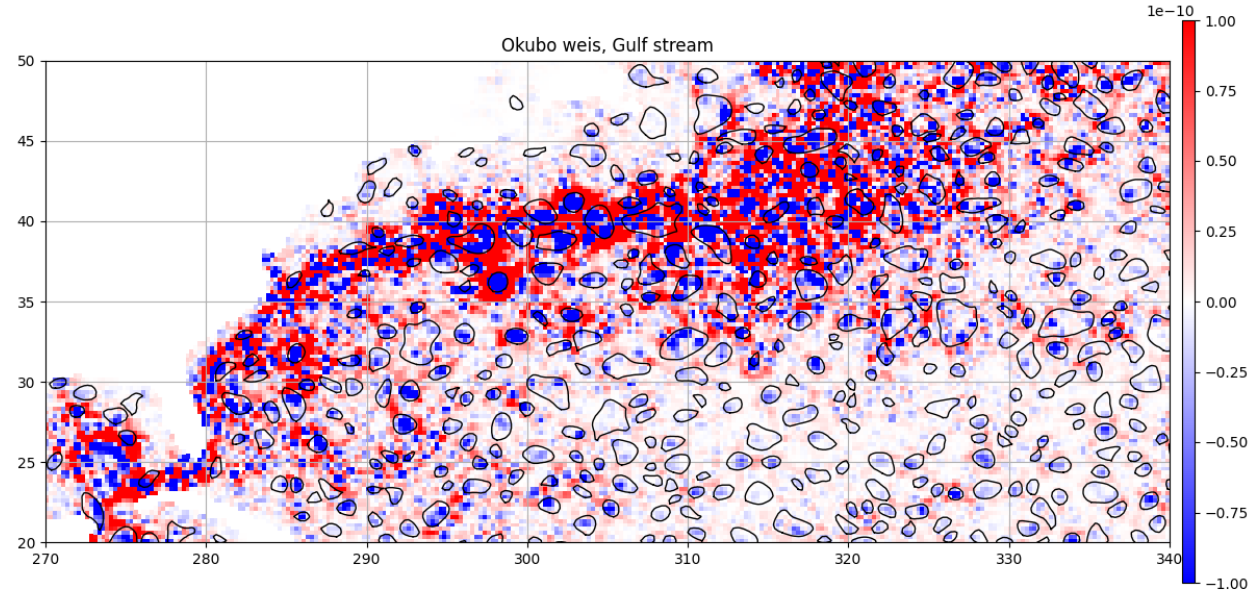
Get parameter for ow

```
u_x = g.compute_stencil(g.grid("ugos"))
u_y = g.compute_stencil(g.grid("ugos"), vertical=True)
v_x = g.compute_stencil(g.grid("vgos"))
v_y = g.compute_stencil(g.grid("vgos"), vertical=True)
ow = g.vars["ow"] = (u_x - v_y) ** 2 + (v_x + u_y) ** 2 - (v_x - u_y) ** 2

ax = start_axes("Okubo weis")
m = g.display(ax, "ow", vmin=-1e-10, vmax=1e-10, cmap="bwr")
update_axes(ax, m)
```



Gulf stream zoom

```
ax = start_axes("Okubo weis, Gulf stream", zoom=True)
m = g.display(ax, "ow", vmin=-1e-10, vmax=1e-10, cmap="bwr")
kw_ed = dict(intern_only=True, color="k", lw=1)
a.display(ax, **kw_ed), c.display(ax, **kw_ed)
update_axes(ax, m)
```



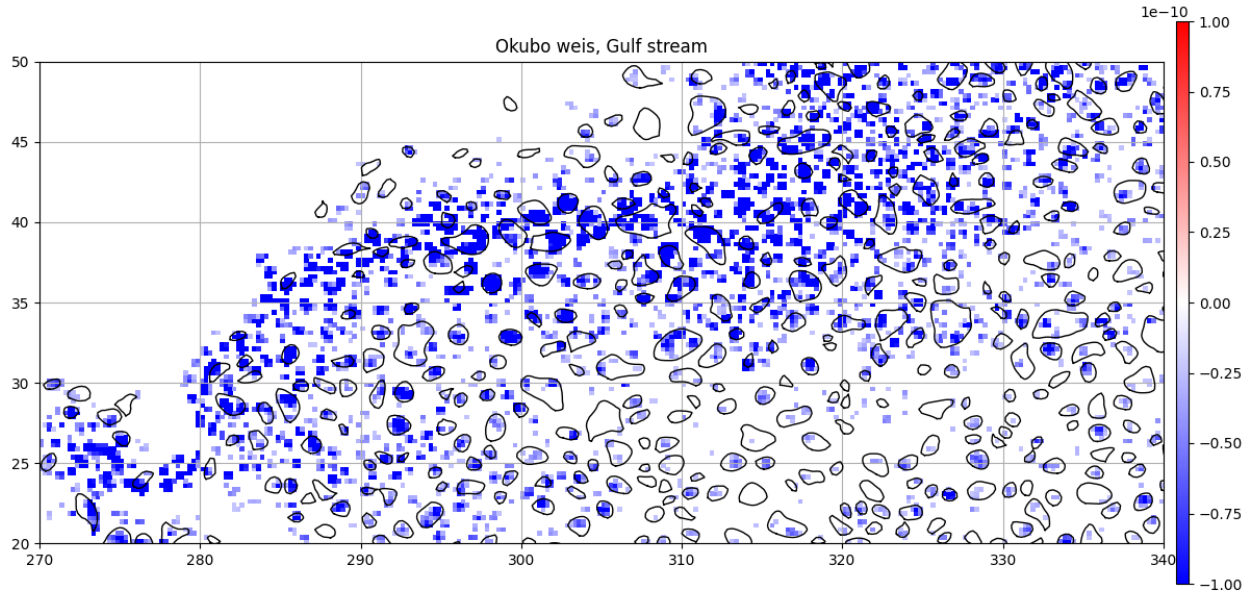
only negative OW

```
ax = start_axes("Okubo weis, Gulf stream", zoom=True)
threshold = ow.std() * -0.2
ow = ma.array(ow, mask=ow > threshold)
m = g.display(ax, ow, vmin=-1e-10, vmax=1e-10, cmap="bwr")
```

(continues on next page)

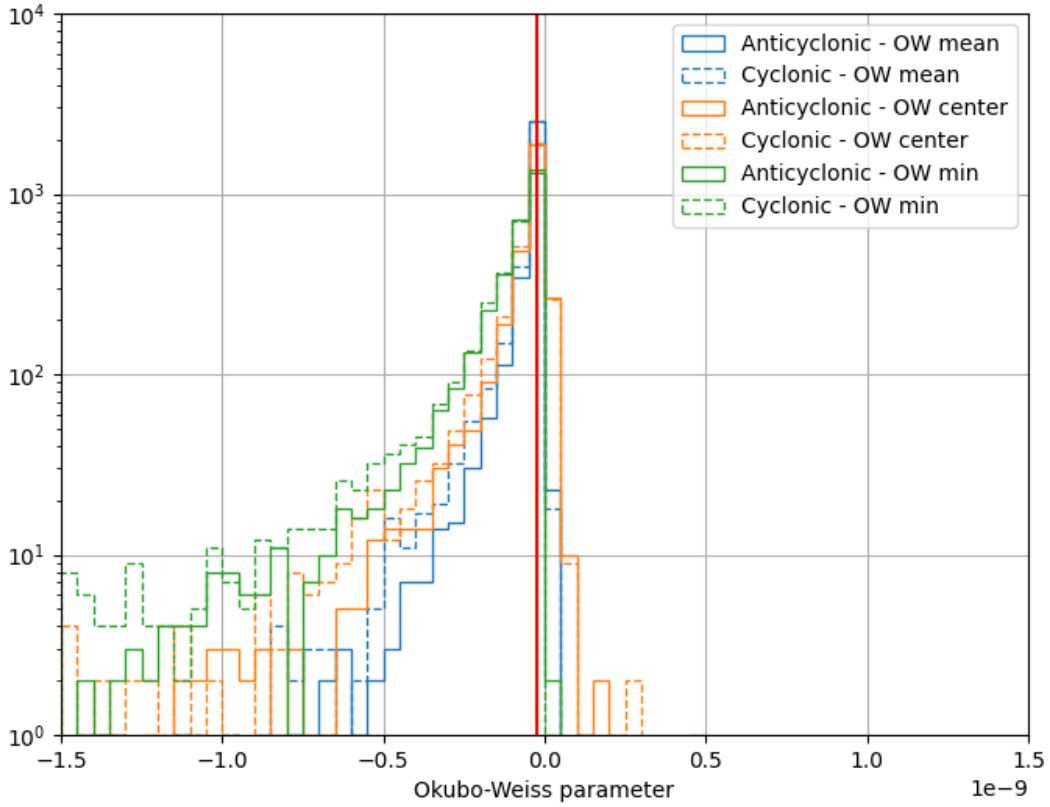
(continued from previous page)

```
a.display(ax, **kw_ed), c.display(ax, **kw_ed)
update_axes(ax, m)
```



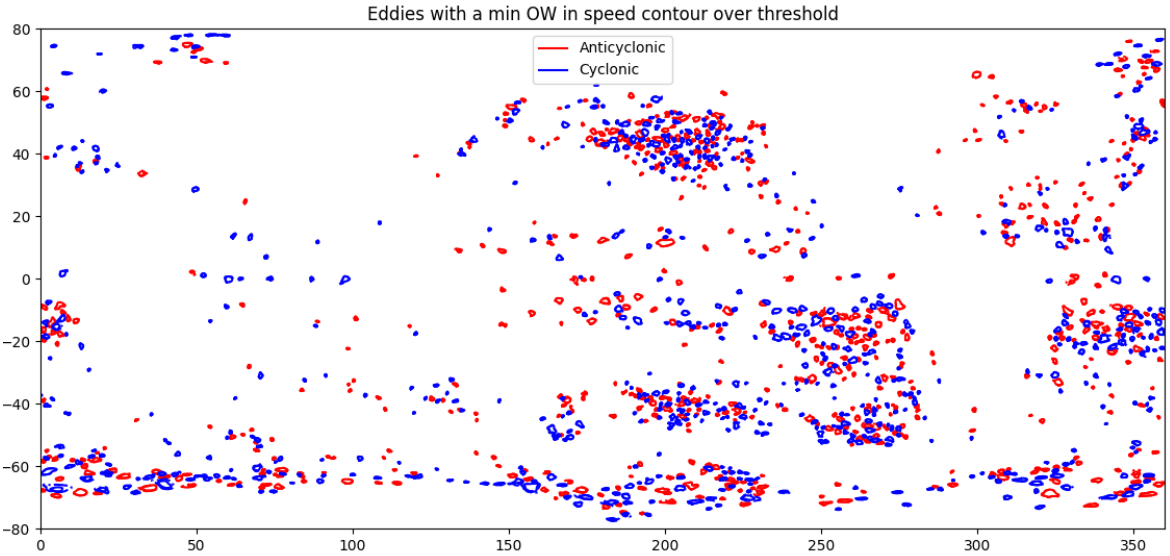
Get okubo-weiss mean/min/center in eddies

```
plt.figure(figsize=(8, 6))
ax = plt.subplot(111)
ax.set_xlabel("Okubo-Weiss parameter")
kw_hist = dict(bins=arange(-20e-10, 20e-10, 50e-12), histtype="step")
for method in ("mean", "center", "min"):
    kw_interp = dict(grid_object=g, varname="ow", method=method, intern=True)
    _, _, m = ax.hist(
        a.interp_grid(**kw_interp), label=f"Anticyclonic - OW {method}", **kw_hist
    )
    ax.hist(
        c.interp_grid(**kw_interp),
        label=f"Cyclonic - OW {method}",
        color=m[0].get_edgecolor(),
        ls="--",
        **kw_hist,
    )
ax.axvline(threshold, color="r")
ax.set_yscale("log")
ax.grid()
ax.set_ylim(1, 1e4)
ax.set_xlim(-15e-10, 15e-10)
ax.legend()
```



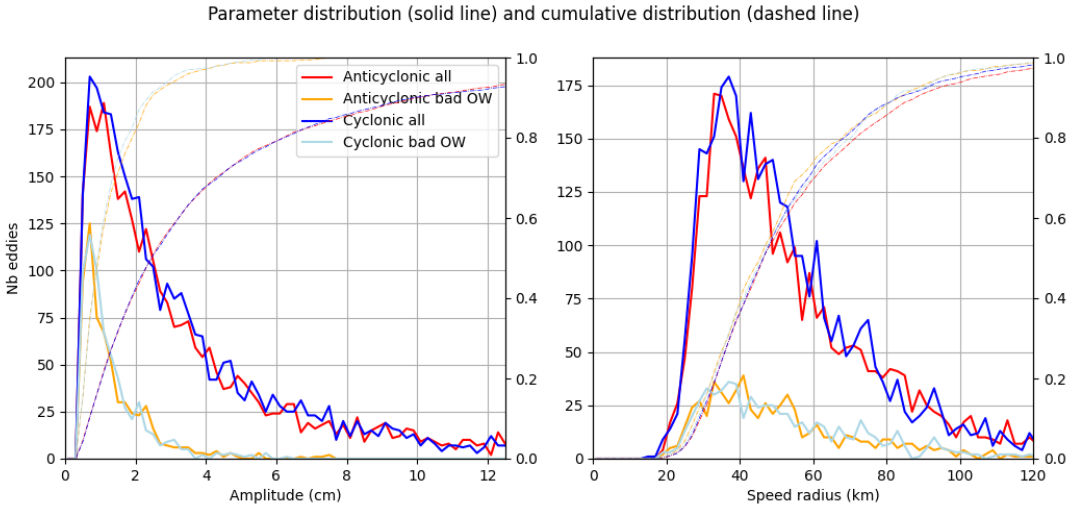
Catch eddies with bad OW

```
ax = start_axes("Eddies with a min OW in speed contour over threshold")
ow_min = a.interp_grid(**kw_interp)
a_bad_ow = a.index(where(ow_min > threshold)[0])
a_bad_ow.display(ax, color="r", label="Anticyclonic")
ow_min = c.interp_grid(**kw_interp)
c_bad_ow = c.index(where(ow_min > threshold)[0])
c_bad_ow.display(ax, color="b", label="Cyclonic")
ax.legend()
```



Display Radius and amplitude of eddies

```
fig = plt.figure(figsize=(12, 5))
fig.suptitle(
    "Parameter distribution (solid line) and cumulative distribution (dashed line)"
)
ax_amp, ax_rad = fig.add_subplot(121), fig.add_subplot(122)
ax_amp_c, ax_rad_c = ax_amp.twinx(), ax_rad.twinx()
ax_amp_c.set_ylim(0, 1), ax_rad_c.set_ylim(0, 1)
kw_a = dict(xname="amplitude", bins=arange(0, 2, 0.002).astype("f4"))
kw_r = dict(xname="radius_s", bins=arange(0, 500e6, 2e3).astype("f4"))
for d, label, color in (
    (a, "Anticyclonic all", "r"),
    (a_bad_ow, "Anticyclonic bad OW", "orange"),
    (c, "Cyclonic all", "blue"),
    (c_bad_ow, "Cyclonic bad OW", "lightblue"),
):
    x, y = d.bins_stat(**kw_a)
    ax_amp.plot(x * 100, y, label=label, color=color)
    ax_amp_c.plot(
        x * 100, y.cumsum() / y.sum(), label=label, color=color, ls="-.", lw=0.5
    )
    x, y = d.bins_stat(**kw_r)
    ax_rad.plot(x * 1e-3, y, label=label, color=color)
    ax_rad_c.plot(
        x * 1e-3, y.cumsum() / y.sum(), label=label, color=color, ls="-.", lw=0.5
    )

ax_amp.set_xlim(0, 12.5), ax_amp.grid(), ax_amp.set_ylim(0), ax_amp.legend()
ax_rad.set_xlim(0, 120), ax_rad.grid(), ax_rad.set_ylim(0)
ax_amp.set_xlabel("Amplitude (cm)"), ax_amp.set_ylabel("Nb eddies")
ax_rad.set_xlabel("Speed radius (km)")
```



Total running time of the script: (0 minutes 7.616 seconds)

CHAPTER

FIVE

TRACKING MANIPULATION

5.1 Track animation

Run in a terminal this script, which allow to watch eddy evolution

```
import py_eddy_tracker_sample

from py_eddy_tracker.appli.gui import Anim
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

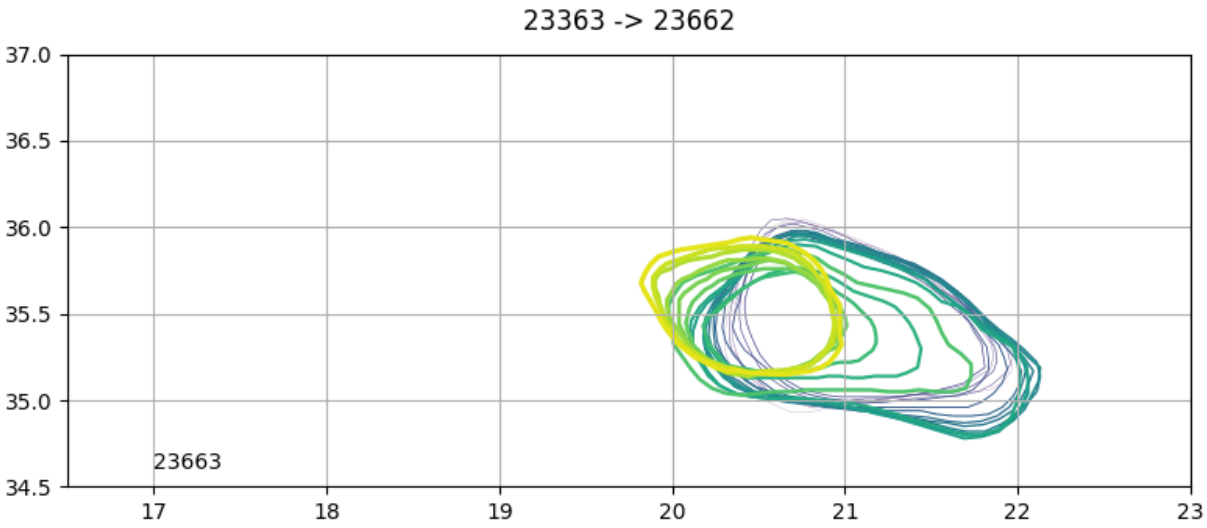
Load experimental atlas, and we select one eddy

```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
# We get only 300 first step to save time of documentation builder
eddy = a.extract_ids([9672]).index(slice(0, 300))
```

Run animation Key shortcut :

- Escape => exit
- SpaceBar => pause
- left arrow => t - 1
- right arrow => t + 1
- + => speed increase of 10 %
- - => speed decrease of 10 %

```
a = Anim(eddy, sleep_event=1e-10, intern=True, figsize=(8, 3.5), cmap="viridis")
a.txt.set_position((17, 34.6))
a.ax.set_xlim(16.5, 23)
a.ax.set_ylim(34.5, 37)
a.show(infinity_loop=False)
```



Total running time of the script: (0 minutes 6.810 seconds)

5.2 Display fields

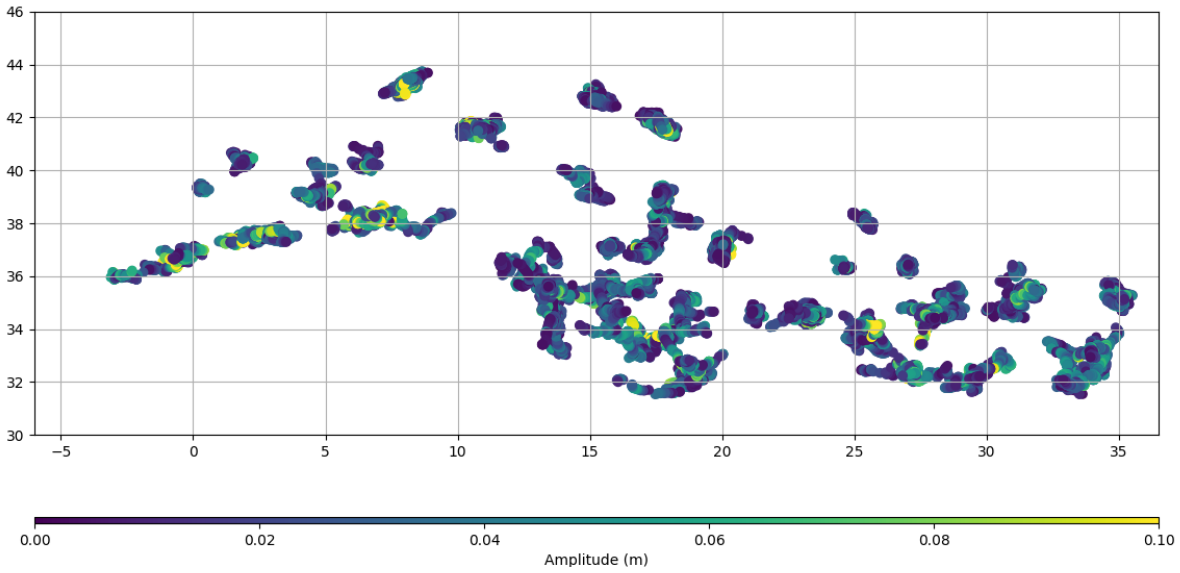
```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load an experimental cyclonic atlas, we keep only eddies which are follow more than 180 days

```
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
c = c.extract_with_length((180, -1))
```

Plot amplitude field

```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_axes((0.05, 0.1, 0.9, 0.9))
ax.set_aspect("equal")
ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
m = c.scatter(ax, "amplitude", ref=-10, vmin=0, vmax=0.1)
ax.grid()

cb = plt.colorbar(
    m, cax=fig.add_axes([0.05, 0.07, 0.9, 0.01]), orientation="horizontal"
)
cb.set_label("Amplitude (m)")
```



Total running time of the script: (0 minutes 2.271 seconds)

5.3 Track animation with standard matplotlib

Run in a terminal this script, which allow to watch eddy evolution

```
import py_eddy_tracker_sample
from matplotlib.animation import FuncAnimation
from numpy import arange

from py_eddy_tracker.appli.gui import Anim
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load experimental atlas, and we select one eddy

```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
eddy = a.extract_ids([9672])
```

Run animation

```
a = Anim(eddy, intern=True, figsize=(8, 3.5), cmap="magma_r", nb_step=6)
a.txt.set_position((17, 34.6))
a.ax.set_xlim(16.5, 23)
a.ax.set_ylim(34.5, 37)

# arguments to get full animation
# kwargs = dict(frames=arange(*a.period), interval=50)
# arguments to reduce compute cost for doucmentation, we display only every 10 days
kwargs = dict(frames=arange(*a.period)[200:800:10], save_count=60, interval=200)

ani = FuncAnimation(a.fig, a.func_animation, **kwargs)
```

Total running time of the script: (0 minutes 16.154 seconds)

5.4 Display Tracks

```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load experimental atlas

```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
print(a)
```

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.3.0-py3.7.egg/py_eddy_tracker/observations/
↪observation.py:220: RuntimeWarning: invalid value encountered in true_divide
  v = v.astype("f4") / v.sum() * 100
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.3.0-py3.7.egg/py_eddy_tracker/observations/
↪observation.py:222: RuntimeWarning: invalid value encountered in true_divide
  v /= hist_numba(self[x], bins=bins)[0]
    | 473554 observations from 15706 to 25334 (9629 days, ~49 obs/day)
    |   Speed area      : 0.00 Mkm²/day
    |   Effective area  : 0.00 Mkm²/day
    ----Distribution in Amplitude:
    |   Amplitude bounds (cm)        0.00      1.00      2.00      3.00      4.00
↪ 5.00     10.00    500.00
    |   Percent of eddies         :      21.69     25.72     15.32      9.95      6.
↪20     14.80      6.32
    ----Distribution in Radius:
    |   Speed radius (km)            0.00     15.00     30.00     45.00     60.00
↪75.00    100.00    200.00   2000.00
    |   Percent of eddies         :       0.85     48.58     36.33     10.97      2.
↪59      0.64      0.03      0.00
    |   Effective radius (km)        0.00     15.00     30.00     45.00     60.00
↪75.00    100.00    200.00   2000.00
    |   Percent of eddies         :     100.00      0.00      0.00      0.00      0.
↪00      0.00      0.00      0.00
    ----Distribution in Latitude
        Latitude bounds            -90.00    -60.00    -15.00     15.00     60.00
↪90.00
        Percent of eddies         :       0.00      0.00      0.00    100.00      0.00
        Percent of speed area     :        nan       nan       nan       nan       nan
        Percent of effective area :        nan       nan       nan       nan       nan
        Mean speed radius (km)    :        nan       nan       nan     32.28       nan
        Mean effective radius (km):        nan       nan       nan      0.00       nan
        Mean amplitude (cm)       :        nan       nan       nan      3.51       nan
    | 12224 tracks (38.74 obs/tracks, shorter 10 obs, longer 1183 obs)
    |   27246 filled observations (2.23 obs/tracks, 5.75 % of total)
    |   Intepolated speed area      : 0.00 Mkm²/day
    |   Intepolated effective area  : 0.00 Mkm²/day
```

(continues on next page)

(continued from previous page)

```
    |   Distance by day             : Mean 3.68 , Median 2.53 km/day
    |   Distance by track           : Mean 139.01 , Median 77.28 km/track
    ----Distribution in lifetime:
    |   Lifetime (days  )            1.00     30.00     90.00    180.00    270.00
↪365.00   1000.00  10000.00
    |   Percent of tracks         :      65.03     26.87      5.72      1.20      0.
↪51       0.65      0.03
    |   Percent of eddies         :      27.88     33.88     17.90      6.74      4.
↪02       8.65      0.93
```

keep only eddies longer than 20 weeks, use -1 to have no upper limit

```
a = a.extract_with_length((7 * 20, -1))
c = c.extract_with_length((7 * 20, -1))
print(a)
```

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.3.0-py3.7.egg/py_eddy_tracker/observations/
↪observation.py:220: RuntimeWarning: invalid value encountered in true_divide
  v = v.astype("f4") / v.sum() * 100
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.3.0-py3.7.egg/py_eddy_tracker/observations/
↪observation.py:222: RuntimeWarning: invalid value encountered in true_divide
  v /= hist_numba(self[x], bins=bins)[0]
    | 124198 observations from 15706 to 25334 (9629 days, ~13 obs/day)
    |   Speed area      : 0.00 Mkm²/day
    |   Effective area  : 0.00 Mkm²/day
    ----Distribution in Amplitude:
    |   Amplitude bounds (cm)        0.00      1.00      2.00      3.00      4.00
↪ 5.00     10.00    500.00
    |   Percent of eddies         :       5.11      9.99     12.06     11.79      9.
↪32      31.51     20.21
    ----Distribution in Radius:
    |   Speed radius (km)            0.00     15.00     30.00     45.00     60.00
↪75.00    100.00    200.00   2000.00
    |   Percent of eddies         :       0.19     22.20     46.88     22.96      6.
↪31       1.39      0.07      0.00
    |   Effective radius (km)        0.00     15.00     30.00     45.00     60.00
↪75.00    100.00    200.00   2000.00
    |   Percent of eddies         :     100.00      0.00      0.00      0.00      0.
↪00       0.00      0.00      0.00
    ----Distribution in Latitude
        Latitude bounds           -90.00    -60.00    -15.00     15.00     60.00
↪90.00
        Percent of eddies         :       0.00      0.00      0.00    100.00      0.00
        Percent of speed area     :        nan       nan       nan       nan       nan
        Percent of effective area :        nan       nan       nan       nan       nan
        Mean speed radius (km)    :        nan       nan       nan     40.03       nan
        Mean effective radius (km):        nan       nan       nan      0.00       nan
        Mean amplitude (cm)       :        nan       nan       nan      6.62       nan
    | 469 tracks (264.81 obs/tracks, shorter 140 obs, longer 1183 obs)
    |   3348 filled observations (7.14 obs/tracks, 2.70 % of total)
    |   Intepolated speed area      : 0.00 Mkm²/day
    |   Intepolated effective area  : 0.00 Mkm²/day
    |   Distance by day             : Mean 3.13 , Median 2.00 km/day
```

(continues on next page)

(continued from previous page)

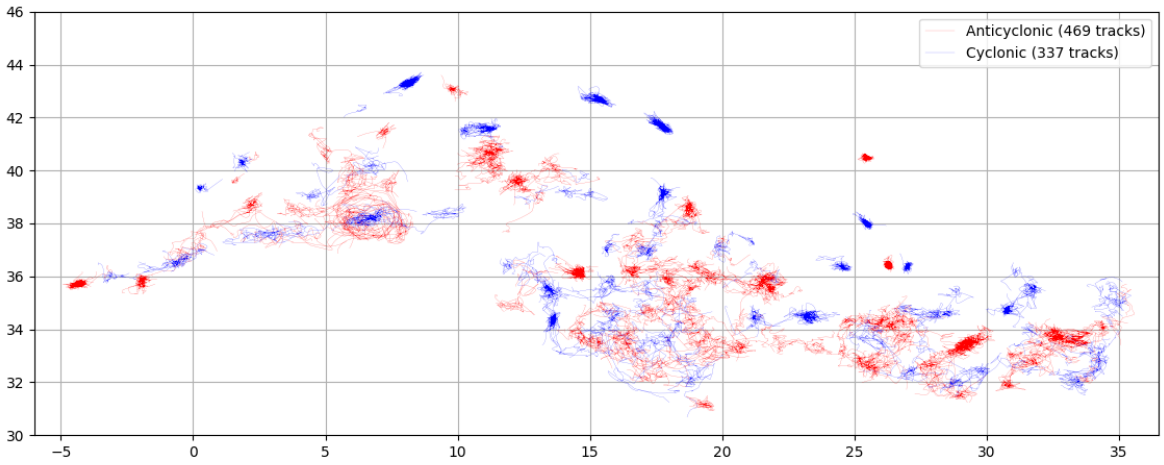
```
    |   Distance by track           : Mean 825.11 , Median 695.99 km/track
    ----Distribution in lifetime:
    |   Lifetime (days  )            1.00     30.00     90.00    180.00    270.00
↪365.00   1000.00  10000.00
    |   Percent of tracks         :       0.00      0.00     37.74     31.34     13.
↪22      16.84      0.85
    |   Percent of eddies         :       0.00      0.00     22.44     25.72     15.
↪34      32.97      3.53
```

Position filtering for nice display

```
a.position_filter(median_half_window=1, loess_half_window=5)
c.position_filter(median_half_window=1, loess_half_window=5)
```

Plot

```
fig = plt.figure(figsize=(12, 5))
ax = fig.add_axes((0.05, 0.1, 0.9, 0.9))
ax.set_aspect("equal")
ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
a.plot(ax, ref=-10, label="Anticyclonic ({nb_tracks} tracks)", color="r", lw=0.1)
c.plot(ax, ref=-10, label="Cyclonic ({nb_tracks} tracks)", color="b", lw=0.1)
ax.legend()
ax.grid()
```



Total running time of the script: (0 minutes 6.547 seconds)

5.5 One Track

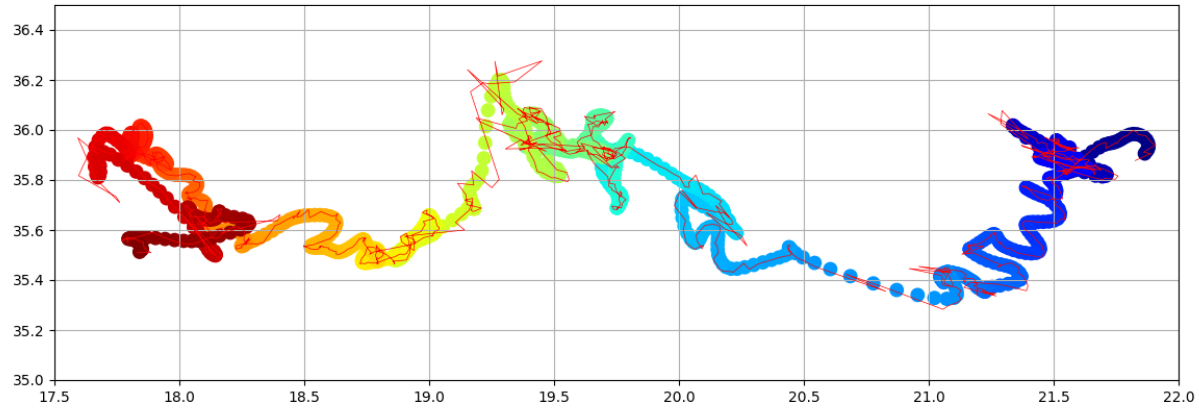
```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load experimental atlas, and we select one eddy

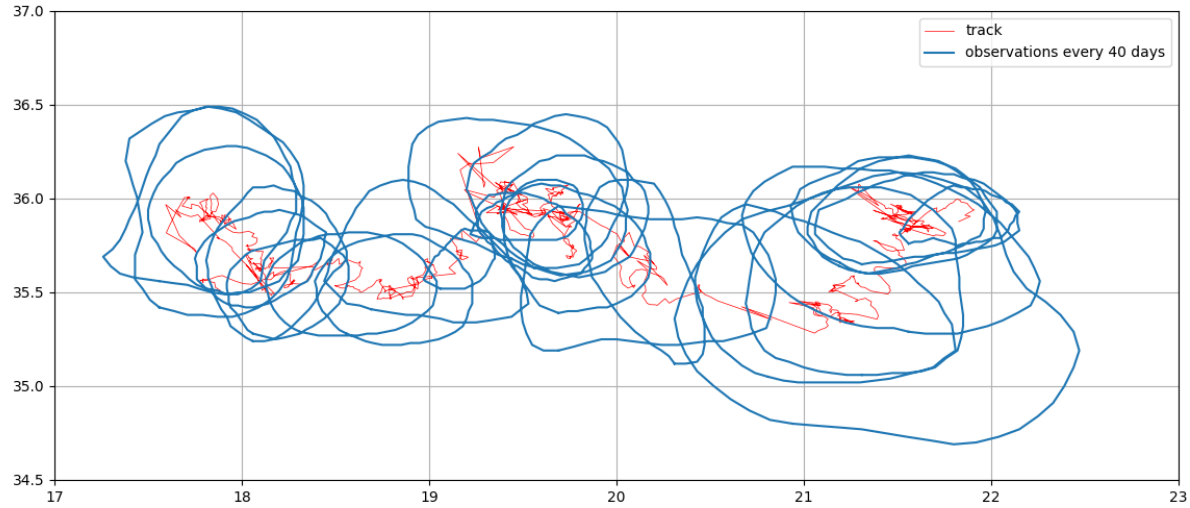
```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
eddy = a.extract_ids([9672])
eddy_f = a.extract_ids([9672])
eddy_f.position_filter(median_half_window=1, loess_half_window=5)
```

plot

```
fig = plt.figure(figsize=(12, 5))
ax = fig.add_axes((0.05, 0.05, 0.9, 0.9))
ax.set_xlim(17.5, 22)
ax.set_ylim(35, 36.5)
ax.set_aspect("equal")
ax.grid()
eddy.plot(ax, color="r", lw=0.5)
eddy_f.scatter(ax, "n", cmap="jet", s=80)
```




```
fig = plt.figure(figsize=(12, 5))
ax = fig.add_axes((0.05, 0.05, 0.9, 0.9))
ax.set_xlim(17, 23)
ax.set_ylim(34.5, 37)
ax.set_aspect("equal")
ax.grid()
eddy.plot(ax, color="r", lw=0.5, label="track")
eddy.index(range(0, len(eddy), 40)).display(
    ax, intern_only=True, label="observations every 40 days"
)
ax.legend()
```



Total running time of the script: (0 minutes 1.384 seconds)

5.6 Tracks which go through area

```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load experimental atlas, we filter position to have nice display

```
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
c.position_filter(median_half_window=1, loess_half_window=5)
```

We extract eddies in the area set below, but we ask to keep *full_path*

```
x0, x1, y0, y1 = 3, 4, 37, 38
area = dict(llcrnrlon=x0, llcrnrlat=y0, urcrnrlon=x1, urcrnrlat=y1)
c_subset = c.extract_with_area(area, full_path=True)
```

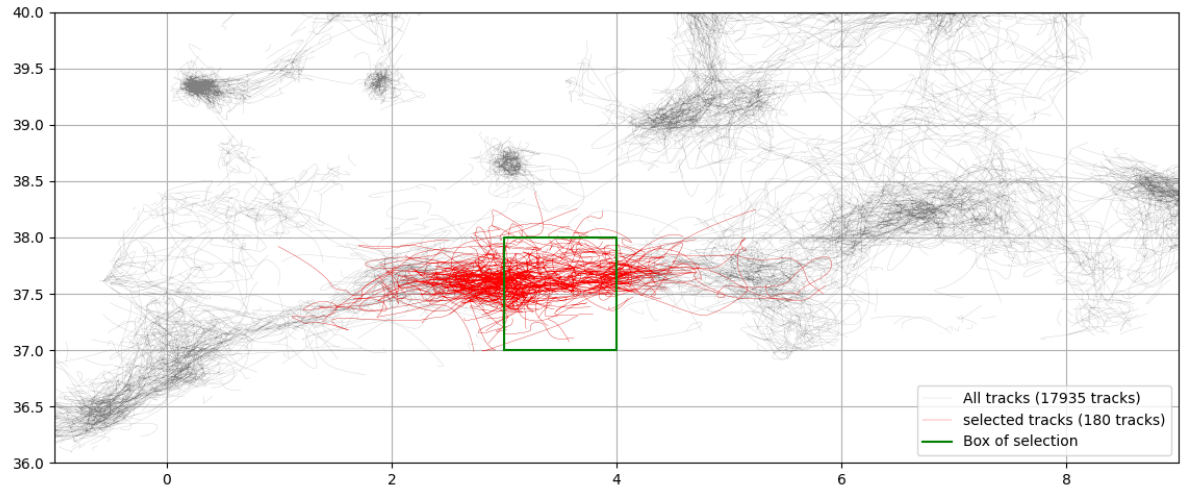
Plot

```
fig = plt.figure(figsize=(12, 5))
ax = fig.add_axes((0.05, 0.05, 0.9, 0.9))
ax.set_xlim(-1, 9)
ax.set_ylim(36, 40)
ax.set_aspect("equal")
ax.grid()
c.plot(ax, color="gray", lw=0.1, ref=-10, label="All tracks ({nb_tracks} tracks)")
c_subset.plot(
    ax, color="red", lw=0.2, ref=-10, label="selected tracks ({nb_tracks} tracks)"
)
ax.plot(
    (x0, x0, x1, x1, x0),
    (y0, y1, y1, y0, y0),
```

(continues on next page)

(continued from previous page)

```
    color="green",
    lw=1.5,
    label="Box of selection",
)
ax.legend()
```



Total running time of the script: (0 minutes 2.784 seconds)

5.7 Track in python

This example didn't replace EddyTracking, we remove check that application do and also postprocessing step.

```
from py_eddy_tracker.data import get_remote_sample
from py_eddy_tracker.featured_tracking.area_tracker import AreaTracker
from py_eddy_tracker.gui import GUI
from py_eddy_tracker.tracking import Correspondances
```

Get remote data, we will keep only 180 first days, *get_remote_sample* function is only to get demo dataset, in your own case give a list of identification filename and don't mix cyclonic and anticyclonic files.

```
file_objects = get_remote_sample(
    "eddies_med_adt_allsat_dt2018/Anticyclonic_2010_2011_2012"
)[:180]
```

We run a traking with a tracker which use contour overlap

```
c = Correspondances(datasets=file_objects, class_method=AreaTracker, virtual=3)
c.track()
c.prepare_merging()
# We have now an eddy object
eddies_area_tracker = c.merge(raw_data=False)
eddies_area_tracker.virtual[:] = eddies_area_tracker.time == 0
eddies_area_tracker.filled_by_interpolation(eddies_area_tracker.virtual == 1)
```

We run a traking with default tracker

```
c = Correspondances(datasets=file_objects, virtual=3)
c.track()
c.prepare_merging()
eddies_default_tracker = c.merge(raw_data=False)
eddies_default_tracker.virtual[:] = eddies_default_tracker.time == 0
eddies_default_tracker.filled_by_interpolation(eddies_default_tracker.virtual == 1)
```

Out:

```
High number of conflict : 56 (nb_conflict)
High number of conflict : 46 (nb_conflict)
High number of conflict : 49 (nb_conflict)
High number of conflict : 50 (nb_conflict)
High number of conflict : 58 (nb_conflict)
High number of conflict : 62 (nb_conflict)
High number of conflict : 67 (nb_conflict)
High number of conflict : 67 (nb_conflict)
High number of conflict : 51 (nb_conflict)
High number of conflict : 50 (nb_conflict)
High number of conflict : 54 (nb_conflict)
High number of conflict : 60 (nb_conflict)
High number of conflict : 59 (nb_conflict)
High number of conflict : 61 (nb_conflict)
High number of conflict : 68 (nb_conflict)
High number of conflict : 74 (nb_conflict)
High number of conflict : 64 (nb_conflict)
High number of conflict : 71 (nb_conflict)
High number of conflict : 67 (nb_conflict)
High number of conflict : 67 (nb_conflict)
High number of conflict : 71 (nb_conflict)
High number of conflict : 78 (nb_conflict)
High number of conflict : 76 (nb_conflict)
High number of conflict : 78 (nb_conflict)
High number of conflict : 71 (nb_conflict)
High number of conflict : 66 (nb_conflict)
High number of conflict : 55 (nb_conflict)
High number of conflict : 60 (nb_conflict)
High number of conflict : 59 (nb_conflict)
High number of conflict : 67 (nb_conflict)
High number of conflict : 57 (nb_conflict)
High number of conflict : 57 (nb_conflict)
High number of conflict : 37 (nb_conflict)
High number of conflict : 72 (nb_conflict)
High number of conflict : 75 (nb_conflict)
High number of conflict : 73 (nb_conflict)
High number of conflict : 77 (nb_conflict)
High number of conflict : 90 (nb_conflict)
High number of conflict : 89 (nb_conflict)
High number of conflict : 88 (nb_conflict)
High number of conflict : 95 (nb_conflict)
High number of conflict : 87 (nb_conflict)
High number of conflict : 78 (nb_conflict)
High number of conflict : 73 (nb_conflict)
High number of conflict : 83 (nb_conflict)
High number of conflict : 86 (nb_conflict)
High number of conflict : 89 (nb_conflict)
High number of conflict : 72 (nb_conflict)
```

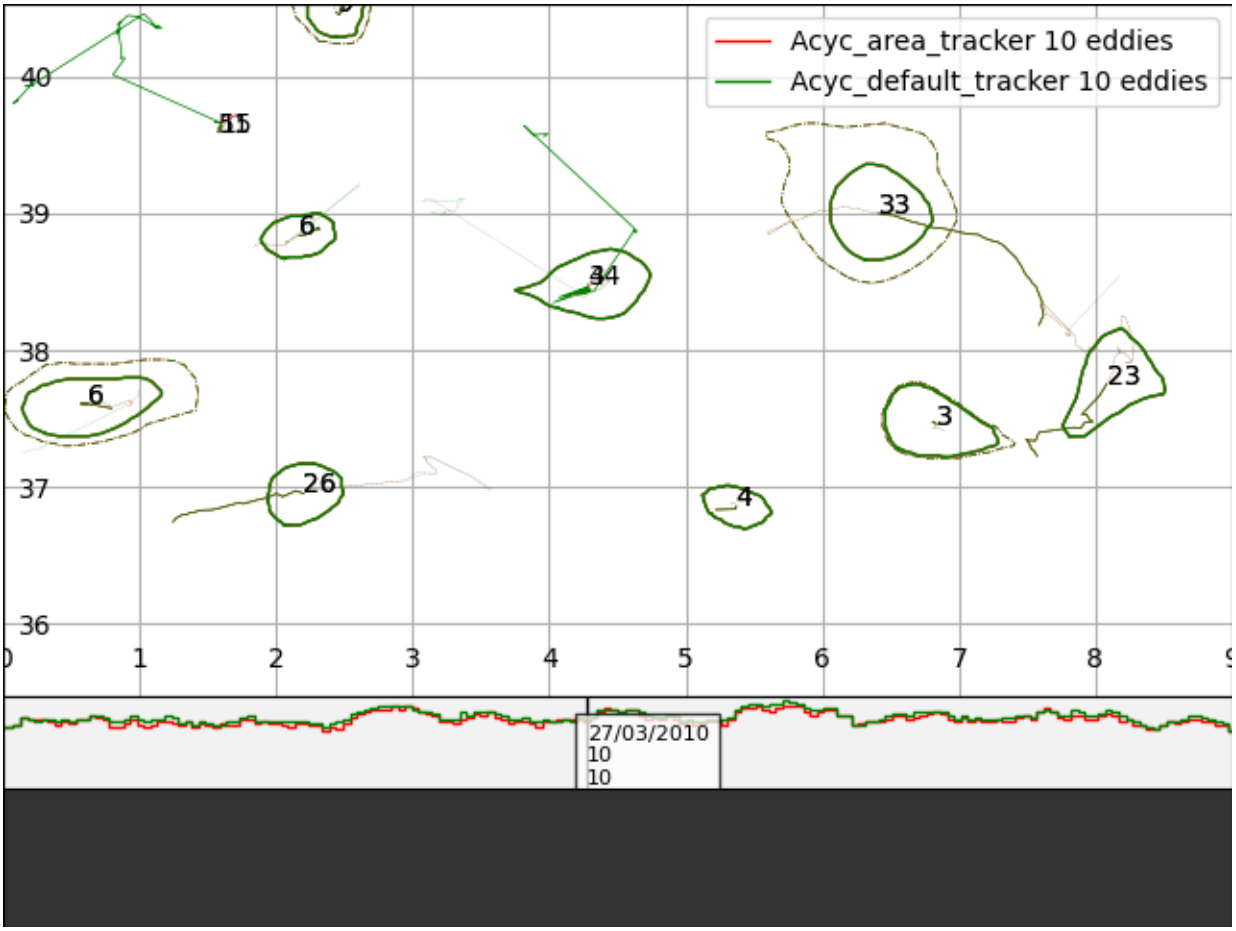
(continues on next page)

(continued from previous page)

```
High number of conflict : 65 (nb_conflict)
High number of conflict : 48 (nb_conflict)
High number of conflict : 83 (nb_conflict)
High number of conflict : 81 (nb_conflict)
High number of conflict : 77 (nb_conflict)
High number of conflict : 89 (nb_conflict)
High number of conflict : 84 (nb_conflict)
High number of conflict : 89 (nb_conflict)
High number of conflict : 99 (nb_conflict)
High number of conflict : 79 (nb_conflict)
High number of conflict : 79 (nb_conflict)
High number of conflict : 75 (nb_conflict)
High number of conflict : 75 (nb_conflict)
High number of conflict : 65 (nb_conflict)
High number of conflict : 80 (nb_conflict)
High number of conflict : 50 (nb_conflict)
High number of conflict : 73 (nb_conflict)
High number of conflict : 73 (nb_conflict)
High number of conflict : 72 (nb_conflict)
High number of conflict : 75 (nb_conflict)
High number of conflict : 62 (nb_conflict)
High number of conflict : 53 (nb_conflict)
High number of conflict : 45 (nb_conflict)
High number of conflict : 49 (nb_conflict)
High number of conflict : 41 (nb_conflict)
High number of conflict : 50 (nb_conflict)
High number of conflict : 46 (nb_conflict)
High number of conflict : 37 (nb_conflict)
High number of conflict : 52 (nb_conflict)
```

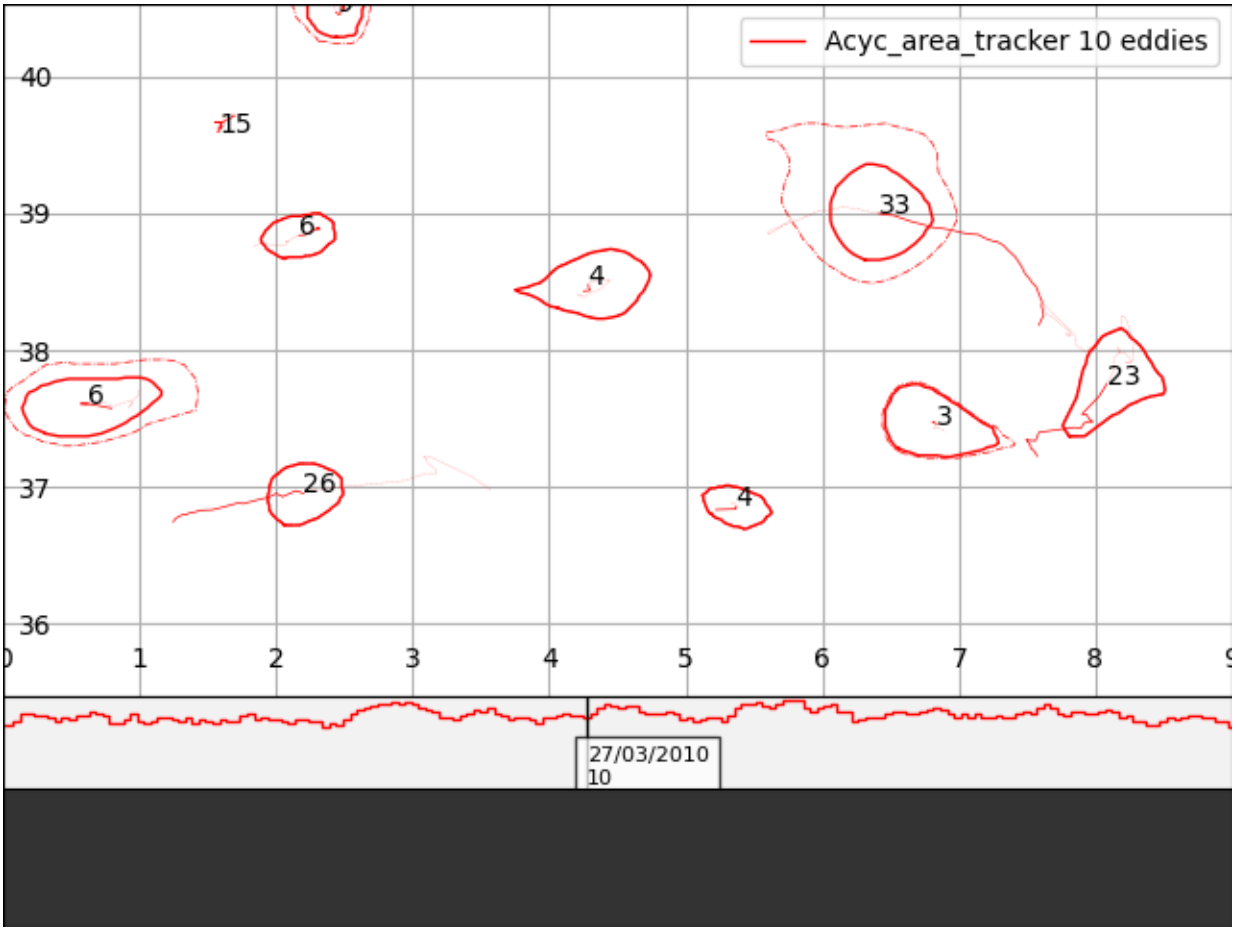
Start GUI to compare tracking

```
g = GUI(
    Acyc_area_tracker=eddies_area_tracker, Acyc_default_tracker=eddies_default_tracker
)
g.now = 22000
g.bbox = 0, 9, 36, 40
g.adjust()
g.show()
```



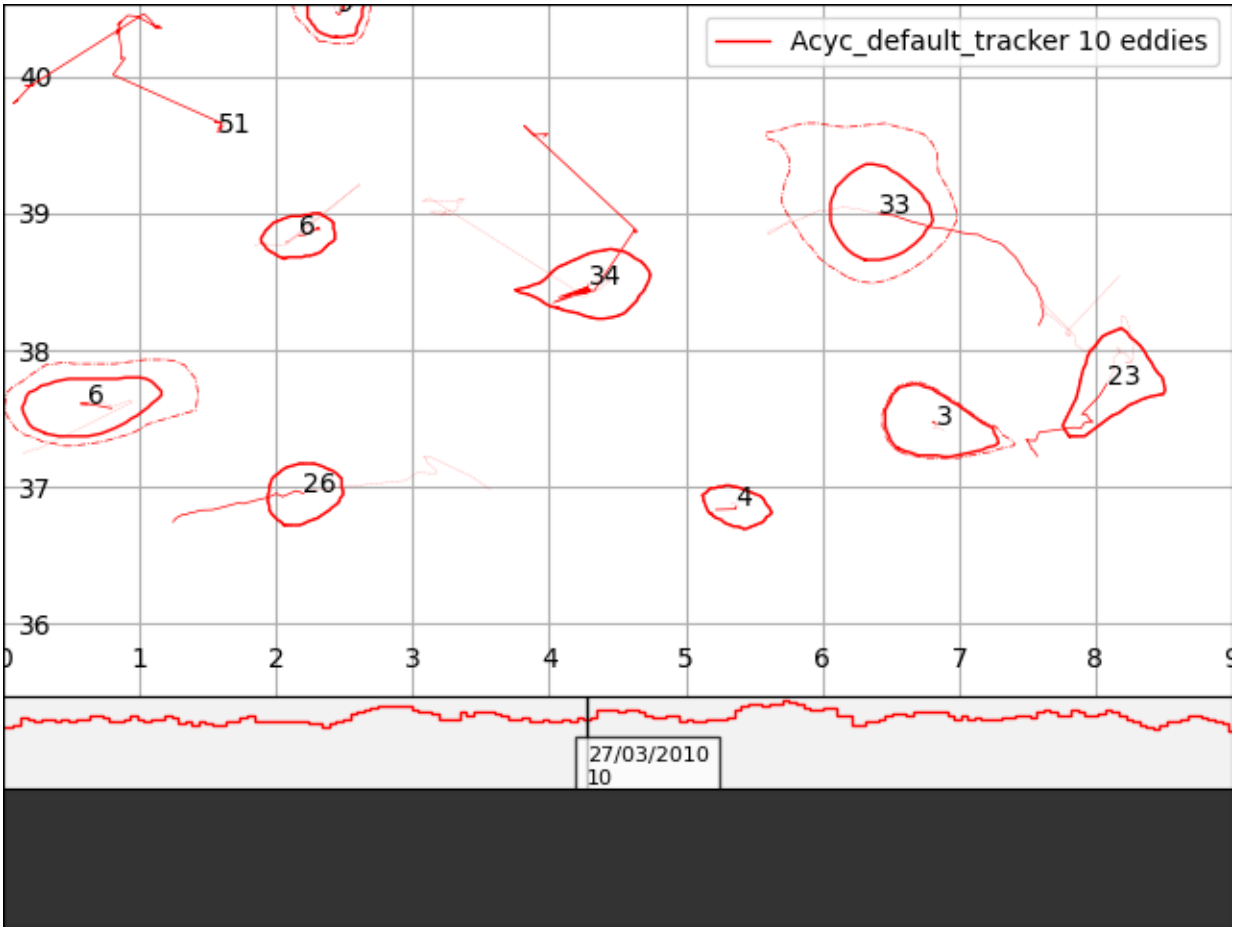
Start GUI with area tracker

```
g = GUI(Acyc_area_tracker=eddies_area_tracker)
g.now = 22000
g.bbox = 0, 9, 36, 40
g.adjust()
g.show()
```



Start GUI with default one

```
g = GUI(Acyc_default_tracker=eddies_default_tracker)
g.now = 22000
g.bbox = 0, 9, 36, 40
g.adjust()
g.show()
```



Total running time of the script: (0 minutes 39.290 seconds)

CHAPTER

SIX

TRACKING DIAGNOSTICS

6.1 Geographical statistics

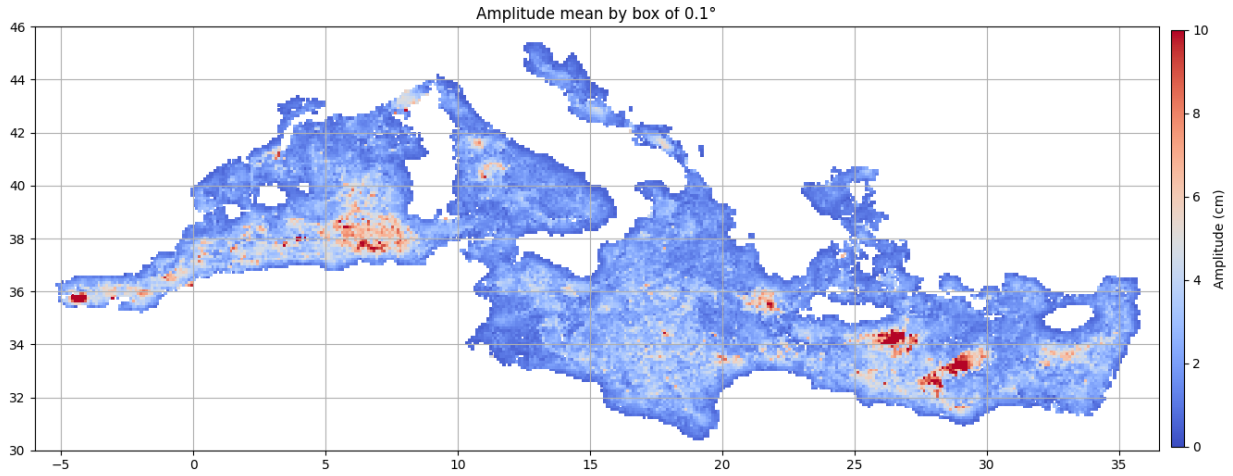
```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt

from py_eddy_tracker.observations.tracking import TrackEddiesObservations


def start_axes(title):
    fig = plt.figure(figsize=(13.5, 5))
    ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
    ax.set_aspect("equal")
    ax.set_title(title)
    return ax
```

Load an experimental med atlas over a period of 26 years (1993-2019), we merge the 2 datasets

```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
a = a.merge(c)

step = 0.1
```

Mean of amplitude in each box

```
ax = start_axes("Amplitude mean by box of %s°" % step)
g = a.grid_stat(((-7, 37, step), (30, 46, step)), "amplitude")
m = g.display(ax, name="amplitude", vmin=0, vmax=10, factor=100)
ax.grid()
cb = plt.colorbar(m, cax=ax.figure.add_axes([0.94, 0.05, 0.01, 0.9]))
cb.set_label("Amplitude (cm)")
```

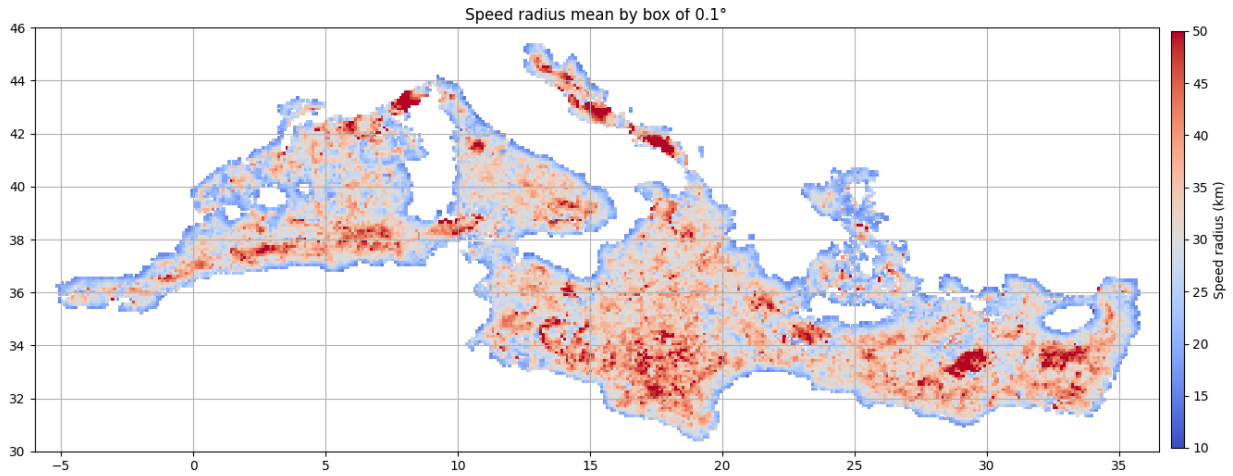


Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.3.0-py3.7.egg/py_eddy_tracker/dataset/grid.
↪py:1896: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same␣
↪dimensions as C is deprecated since 3.3.  Either specify the corners of the␣
↪quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or set␣
↪rcParams['pcolor.shading'].  This will become an error two minor releases later.
  return ax.pcolormesh(x, self.y_bounds, data.T * factor, **kwargs)
```

Mean of speed radius in each box

```
ax = start_axes("Speed radius mean by box of %s°" % step)
g = a.grid_stat(((-7, 37, step), (30, 46, step)), "radius_s")
m = g.display(ax, name="radius_s", vmin=10, vmax=50, factor=0.001)
ax.grid()
cb = plt.colorbar(m, cax=ax.figure.add_axes([0.94, 0.05, 0.01, 0.9]))
cb.set_label("Speed radius (km)")
```



Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.3.0-py3.7.egg/py_eddy_tracker/dataset/grid.
↪py:1896: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same␣
↪dimensions as C is deprecated since 3.3.  Either specify the corners of the␣
↪quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or set␣
↪rcParams['pcolor.shading'].  This will become an error two minor releases later.
```

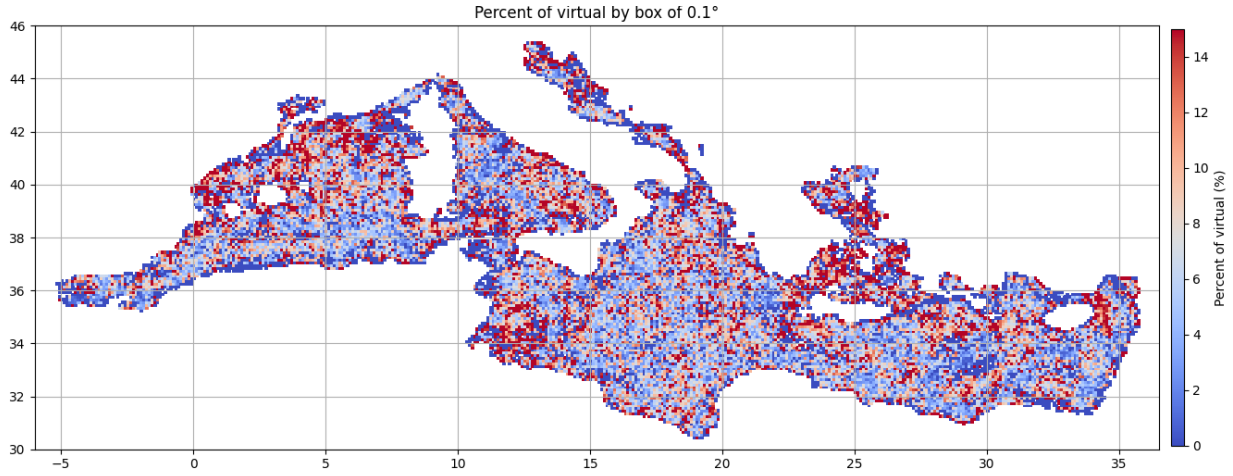
(continues on next page)

(continued from previous page)

```
  return ax.pcolormesh(x, self.y_bounds, data.T * factor, **kwargs)
```

Percent of virtual on the whole obs in each box

```
ax = start_axes("Percent of virtual by box of %s°" % step)
g = a.grid_stat(((-7, 37, step), (30, 46, step)), "virtual")
g.vars["virtual"] *= 100
m = g.display(ax, name="virtual", vmin=0, vmax=15)
ax.grid()
cb = plt.colorbar(m, cax=ax.figure.add_axes([0.94, 0.05, 0.01, 0.9]))
cb.set_label("Percent of virtual (%)")
```



Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.3.0-py3.7.egg/py_eddy_tracker/dataset/grid.
↪py:1896: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same␣
↪dimensions as C is deprecated since 3.3.  Either specify the corners of the␣
↪quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or set␣
↪rcParams['pcolor.shading'].  This will become an error two minor releases later.
  return ax.pcolormesh(x, self.y_bounds, data.T * factor, **kwargs)
```

Total running time of the script: (0 minutes 4.987 seconds)

6.2 Birth and death

Following figures are based on https://doi.org/10.1016/j.pocean.2011.01.002

```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

```
def start_axes(title):
    fig = plt.figure(figsize=(13, 5))
    ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
```

(continues on next page)

(continued from previous page)

```
    ax.set_aspect("equal")
    ax.set_title(title)
    return ax


def update_axes(ax, mappable=None):
    ax.grid()
    if mappable:
        plt.colorbar(mappable, cax=ax.figure.add_axes([0.95, 0.05, 0.01, 0.9]))
```

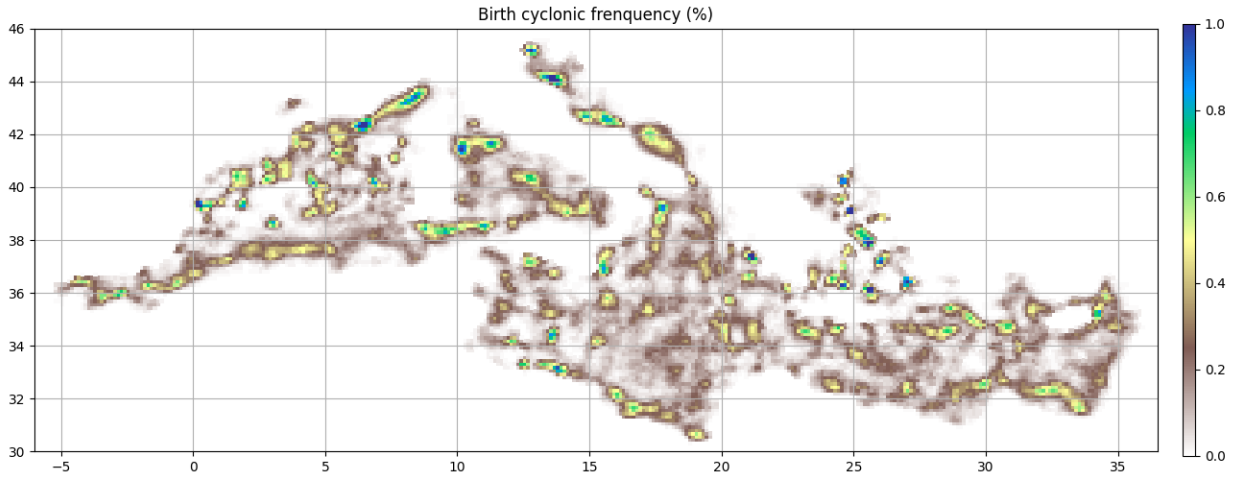
Load an experimental med atlas over a period of 26 years (1993-2019)

```
kwargs_load = dict(
    include_vars=(
        "longitude",
        "latitude",
        "observation_number",
        "track",
        "time",
        "speed_contour_longitude",
        "speed_contour_latitude",
    )
)
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
```

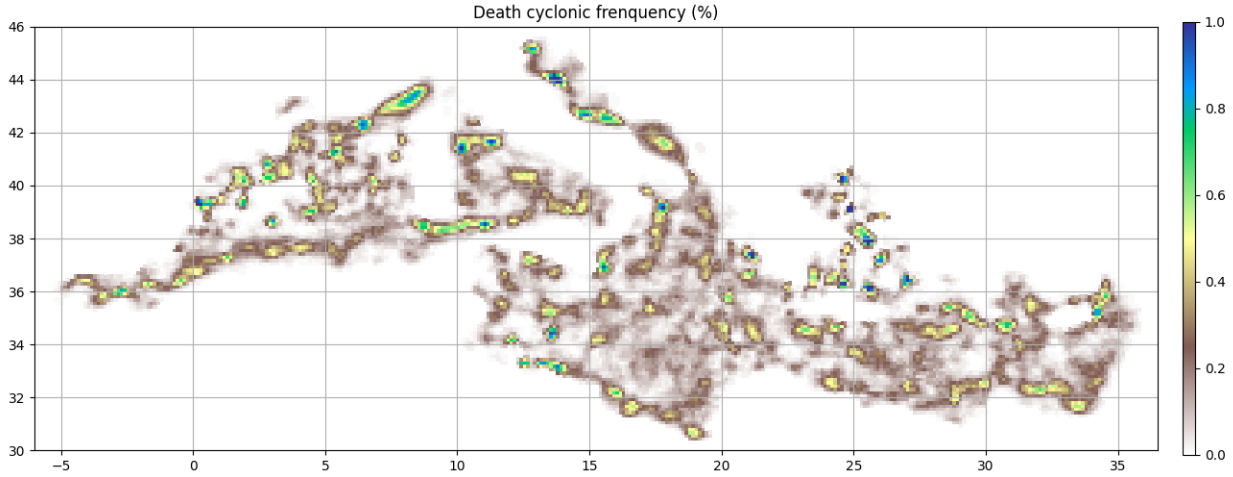
```
t0, t1 = a.period
step = 0.125
bins = ((-10, 37, step), (30, 46, step))
kwargs = dict(cmap="terrain_r", factor=100 / (t1 - t0), name="count", vmin=0, vmax=1)
```

6.2.1 Cyclonic

```
ax = start_axes("Birth cyclonic frenquency (%)")
g_c_first = c.first_obs().grid_count(bins, intern=True)
m = g_c_first.display(ax, **kwargs)
update_axes(ax, m)
```

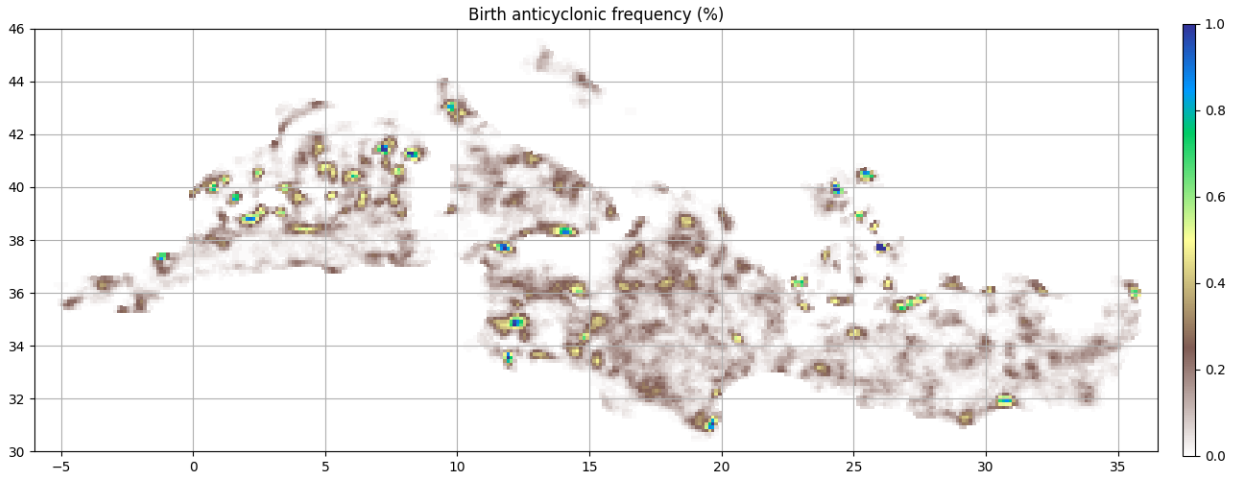



```
ax = start_axes("Death cyclonic frenquency (%)")
g_c_last = c.last_obs().grid_count(bins, intern=True)
m = g_c_last.display(ax, **kwargs)
update_axes(ax, m)
```

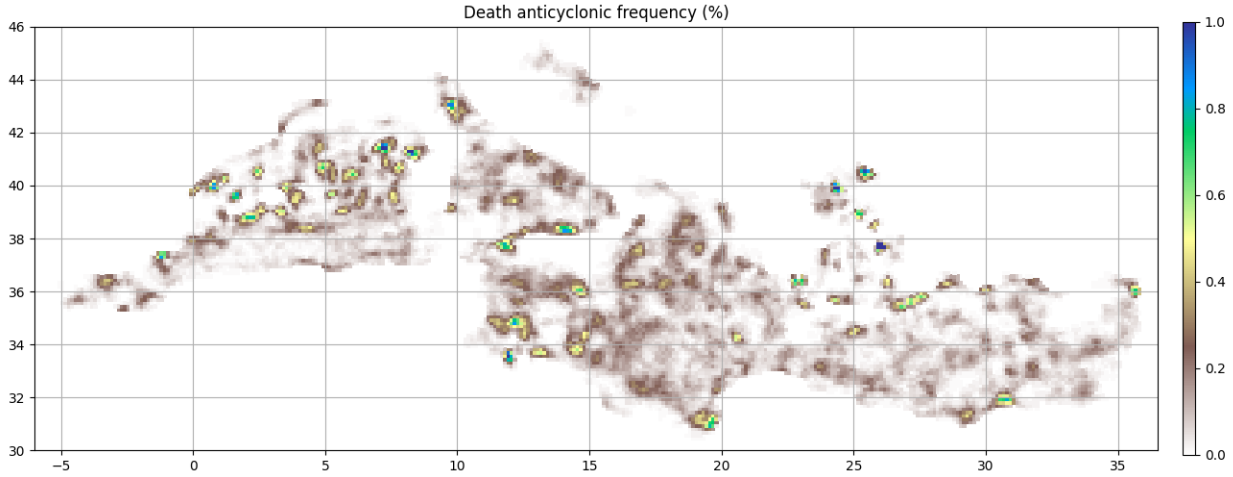


6.2.2 Anticyclonic

```
ax = start_axes("Birth anticyclonic frequency (%)")
g_a_first = a.first_obs().grid_count(bins, intern=True)
m = g_a_first.display(ax, **kwargs)
update_axes(ax, m)
```




```
ax = start_axes("Death anticyclonic frequency (%)")
g_a_last = a.last_obs().grid_count(bins, intern=True)
m = g_a_last.display(ax, **kwargs)
update_axes(ax, m)
```



Total running time of the script: (0 minutes 5.135 seconds)

6.3 Lifetime Histogram

```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt
from numpy import arange, ones

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load an experimental med atlas over a period of 26 years (1993-2019)

```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
c = TrackEddiesObservations.load_file(
```

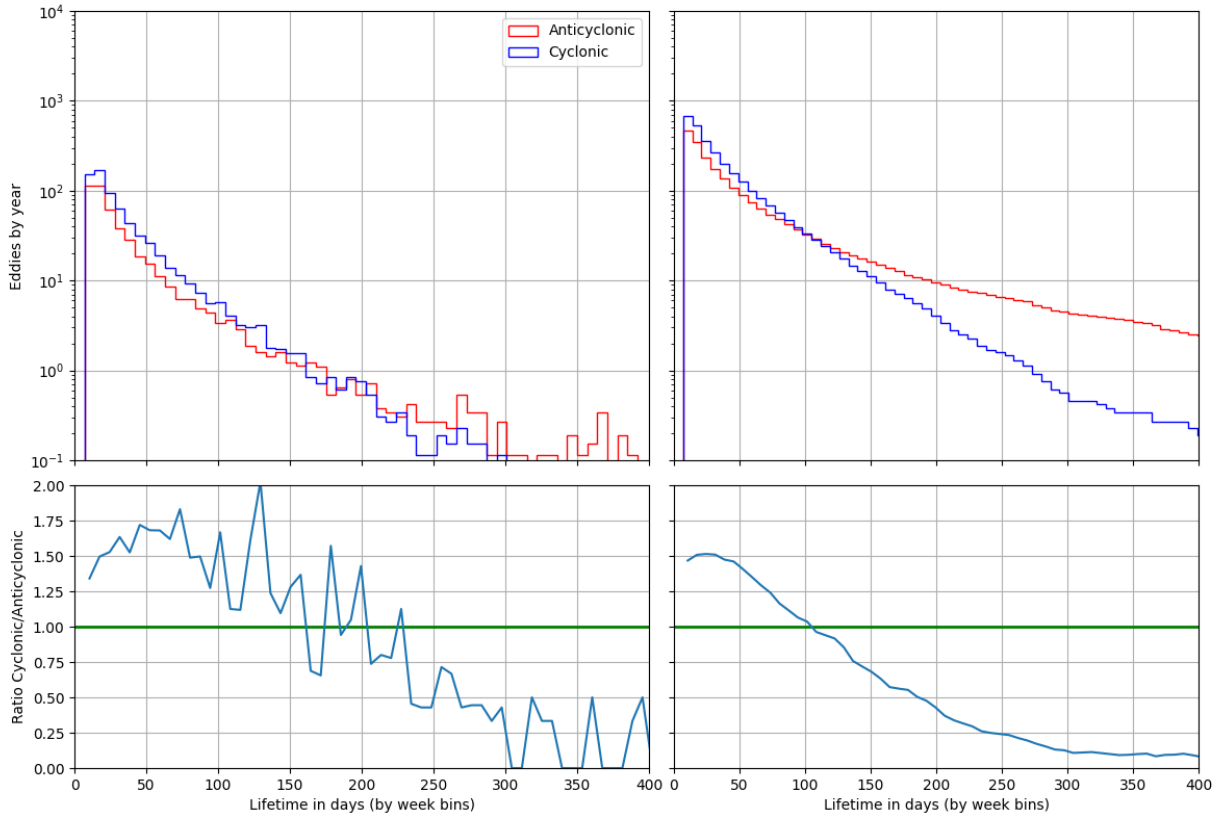
(continues on next page)

(continued from previous page)

```
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
nb_year = (a.period[1] - a.period[0] + 1) / 365.25
```

Setup axes

```
figure = plt.figure(figsize=(12, 8))
ax_ratio_cum = figure.add_axes([0.55, 0.06, 0.42, 0.34])
ax_ratio = figure.add_axes([0.07, 0.06, 0.46, 0.34])
ax_cum = figure.add_axes([0.55, 0.43, 0.42, 0.54])
ax = figure.add_axes([0.07, 0.43, 0.46, 0.54])
ax.set_ylabel("Eddies by year")
ax_ratio.set_ylabel("Ratio Cyclonic/Anticyclonic")
for ax_ in (ax, ax_cum, ax_ratio_cum, ax_ratio):
    ax_.set_xlim(0, 400)
    if ax_ in (ax, ax_cum):
        ax_.set_ylim(1e-1, 1e4), ax_.set_yscale("log")
    else:
        ax_.set_xlabel("Lifetime in days (by week bins)")
        ax_.set_ylim(0, 2)
        ax_.axhline(1, color="g", lw=2)
    ax_.grid()
ax_cum.xaxis.set_ticklabels([]), ax_cum.yaxis.set_ticklabels([])
ax.xaxis.set_ticklabels([]), ax_ratio_cum.yaxis.set_ticklabels([])

# plot data
bin_hist = arange(7, 2000, 7)
x = (bin_hist[1:] + bin_hist[:-1]) / 2.0
a_nb, c_nb = a.nb_obs_by_track, c.nb_obs_by_track
a_nb, c_nb = a_nb[a_nb != 0], c_nb[c_nb != 0]
w_a, w_c = ones(a_nb.shape) / nb_year, ones(c_nb.shape) / nb_year
kwargs_a = dict(histtype="step", bins=bin_hist, x=a_nb, color="r", weights=w_a)
kwargs_c = dict(histtype="step", bins=bin_hist, x=c_nb, color="b", weights=w_c)
cum_a, _, _ = ax_cum.hist(cumulative=-1, **kwargs_a)
cum_c, _, _ = ax_cum.hist(cumulative=-1, **kwargs_c)
nb_a, _, _ = ax.hist(label="Anticyclonic", **kwargs_a)
nb_c, _, _ = ax.hist(label="Cyclonic", **kwargs_c)
ax_ratio_cum.plot(x, cum_c / cum_a)
ax_ratio.plot(x, nb_c / nb_a)
ax.legend()
```



Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_lifetime.py:55: RuntimeWarning: invalid value␣
↪encountered in true_divide
  ax_ratio_cum.plot(x, cum_c / cum_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_lifetime.py:56: RuntimeWarning: divide by zero␣
↪encountered in true_divide
  ax_ratio.plot(x, nb_c / nb_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_lifetime.py:56: RuntimeWarning: invalid value␣
↪encountered in true_divide
  ax_ratio.plot(x, nb_c / nb_a)
```

Total running time of the script: (0 minutes 3.156 seconds)

6.4 Parameter Histogram

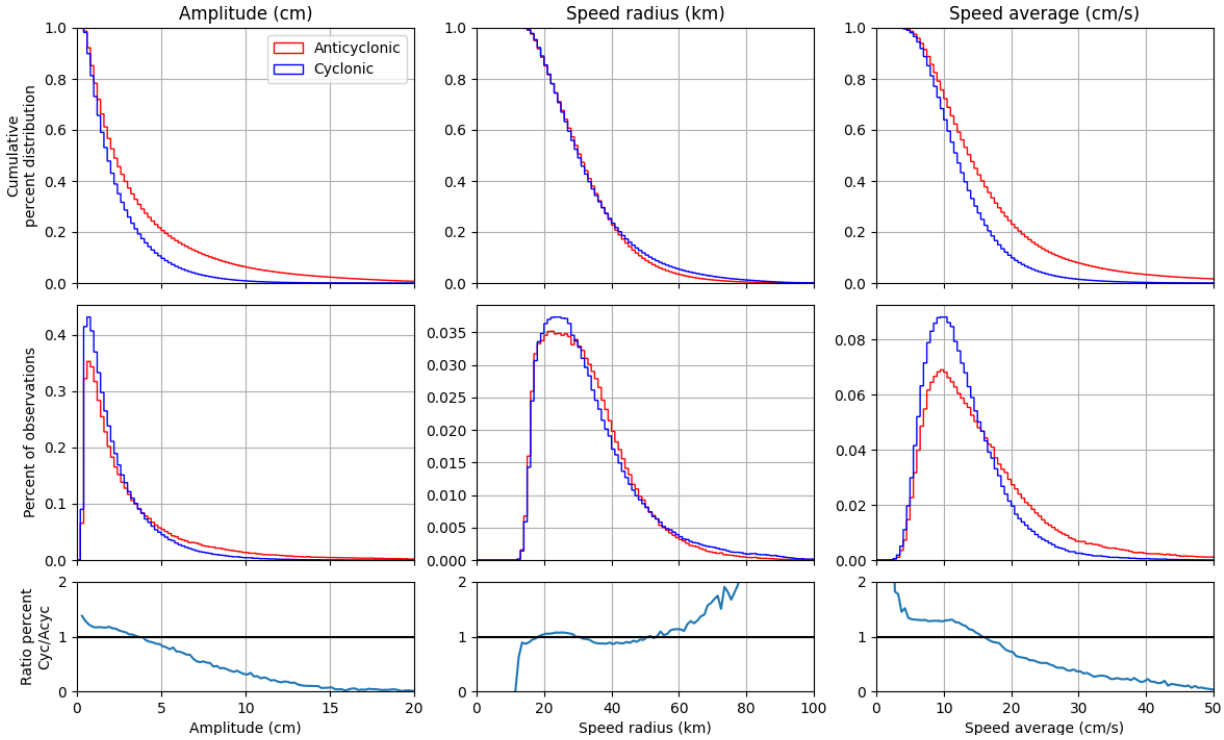
```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt
from numpy import arange

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load an experimental med atlas over a period of 26 years (1993-2019)

```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
kwargs_a = dict(label="Anticyclonic", color="r", histtype="step", density=True)
kwargs_c = dict(label="Cyclonic", color="b", histtype="step", density=True)
```

Plot

```
fig = plt.figure(figsize=(12, 7))

for x0, name, title, xmax, factor, bins in zip(
    (0.4, 0.72, 0.08),
    ("speed_radius", "speed_average", "amplitude"),
    ("Speed radius (km)", "Speed average (cm/s)", "Amplitude (cm)"),
    (100, 50, 20),
    (0.001, 100, 100),
    (arange(0, 2000, 1), arange(0, 1000, 0.5), arange(0.0005, 1000, 0.2)),
):
    ax_hist = fig.add_axes((x0, 0.24, 0.27, 0.35))
    nb_a, _, _ = ax_hist.hist(a[name] * factor, bins=bins, **kwargs_a)
    nb_c, _, _ = ax_hist.hist(c[name] * factor, bins=bins, **kwargs_c)
    ax_hist.set_xticklabels([])
    ax_hist.set_xlim(0, xmax)
    ax_hist.grid()

    ax_cum = fig.add_axes((x0, 0.62, 0.27, 0.35))
    ax_cum.hist(a[name] * factor, bins=bins, cumulative=-1, **kwargs_a)
    ax_cum.hist(c[name] * factor, bins=bins, cumulative=-1, **kwargs_c)
    ax_cum.set_xticklabels([])
    ax_cum.set_title(title)
    ax_cum.set_xlim(0, xmax)
    ax_cum.set_ylim(0, 1)
    ax_cum.grid()

    ax_ratio = fig.add_axes((x0, 0.06, 0.27, 0.15))
    ax_ratio.set_xlim(0, xmax)
    ax_ratio.set_ylim(0, 2)
    ax_ratio.plot((bins[1:] + bins[:-1]) / 2, nb_c / nb_a)
    ax_ratio.axhline(1, color="k")
    ax_ratio.grid()
    ax_ratio.set_xlabel(title)

ax_cum.set_ylabel("Cumulative\npercent distribution")
ax_hist.set_ylabel("Percent of observations")
ax_ratio.set_ylabel("Ratio percent\nCyc/Acyc")
ax_cum.legend()
```



Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_histo.py:54: RuntimeWarning: divide by zero␣
↪encountered in true_divide
  ax_ratio.plot((bins[1:] + bins[:-1]) / 2, nb_c / nb_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_histo.py:54: RuntimeWarning: invalid value␣
↪encountered in true_divide
  ax_ratio.plot((bins[1:] + bins[:-1]) / 2, nb_c / nb_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_histo.py:54: RuntimeWarning: divide by zero␣
↪encountered in true_divide
  ax_ratio.plot((bins[1:] + bins[:-1]) / 2, nb_c / nb_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_histo.py:54: RuntimeWarning: invalid value␣
↪encountered in true_divide
  ax_ratio.plot((bins[1:] + bins[:-1]) / 2, nb_c / nb_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_histo.py:54: RuntimeWarning: invalid value␣
↪encountered in true_divide
  ax_ratio.plot((bins[1:] + bins[:-1]) / 2, nb_c / nb_a)
```

Total running time of the script: (0 minutes 3.634 seconds)

6.5 Groups distribution

```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt
from numpy import arange, ones, percentile

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load an experimental med atlas over a period of 26 years (1993-2019)

```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
```

Group distribution

```
groups = dict()
bins_time = [10, 20, 30, 60, 90, 180, 360, 100000]
for t0, t1 in zip(bins_time[:-1], bins_time[1:]):
    groups[f"lifetime_{t0}_{t1}"] = lambda dataset, t0=t0, t1=t1: (
        dataset.lifetime >= t0
    ) * (dataset.lifetime < t1)
bins_percentile = arange(0, 100.0001, 5)
```

Function to build stats

```
def stats_compilation(dataset, groups, field, bins, filter=None):
    datas = dict(ref=dataset.bins_stat(field, bins=bins, mask=filter)[1], y=dict())
    for k, index in groups.items():
        i = dataset.merge_filters(filter, index)
        x, datas["y"][k] = dataset.bins_stat(field, bins=bins, mask=i)
    datas["x"], datas["bins"] = x, bins
    return datas


def plot_stats(ax, bins, x, y, ref, box=False, cmap=None, percentiles=None, **kw):
    base, ref = ones(x.shape) * 100.0, ref / 100.0
    x = arange(bins.shape[0]).repeat(2)[1:-1] if box else x
    y0 = base
    if cmap is not None:
        cmap, nb_groups = plt.get_cmap(cmap), len(y)
    keys = tuple(y.keys())
    for i, k in enumerate(keys[::-1]):
        y1 = y0 - y[k] / ref
        args = (y0.repeat(2), y1.repeat(2)) if box else (y0, y1)
        if cmap is not None:
            kw["color"] = cmap(1 - i / (nb_groups - 1))
        ax.fill_between(x, *args, label=k, **kw)
        y0 = y1
    if percentiles:
        for b in bins:
            ax.axvline(b, **percentiles)
```

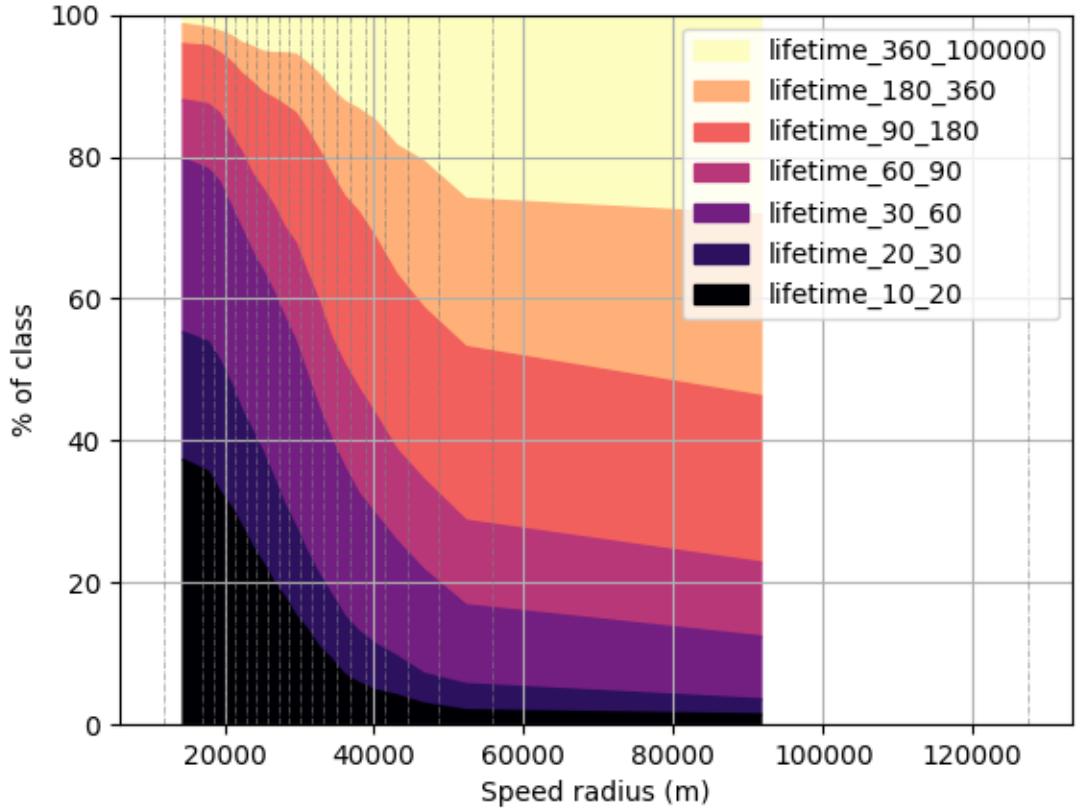
Speed radius by track period

```
stats = stats_compilation(
    a, groups, "radius_s", percentile(a.radius_s, bins_percentile)
```

(continues on next page)

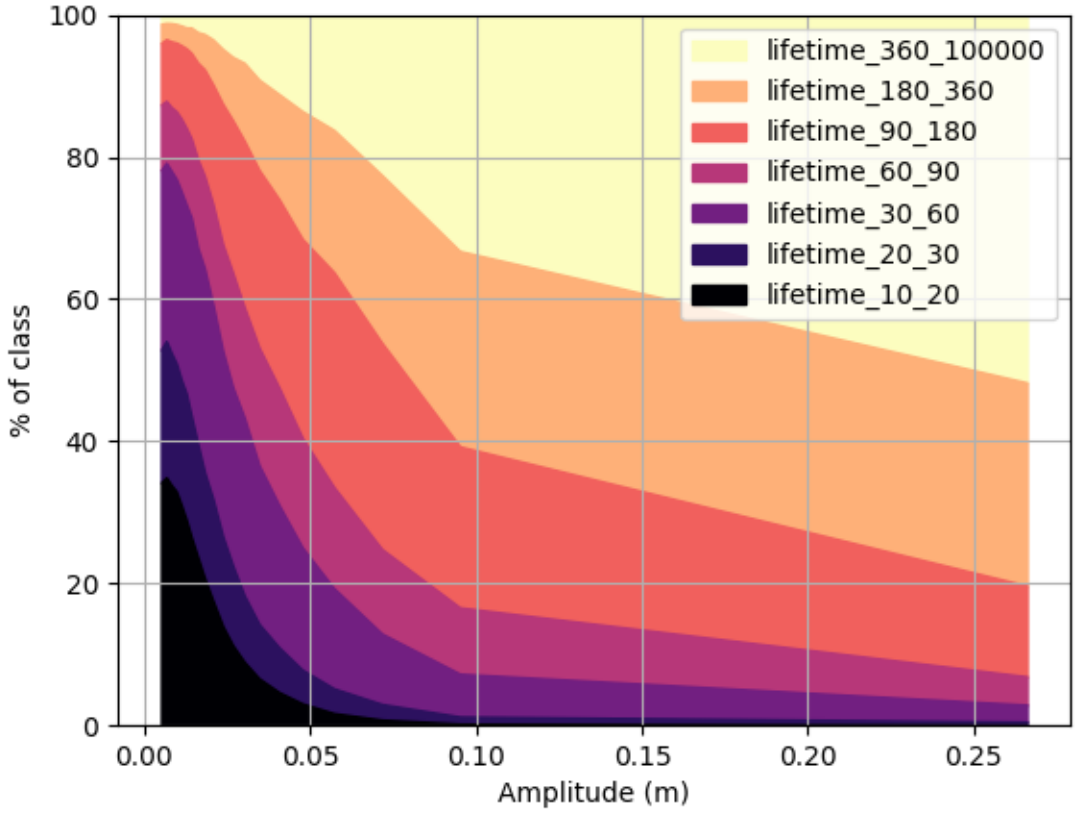
(continued from previous page)

```
)
fig = plt.figure()
ax = fig.add_subplot(111)
plot_stats(ax, **stats, cmap="magma", percentiles=dict(color="gray", ls="-.", lw=0.4))
ax.set_xlabel("Speed radius (m)"), ax.set_ylabel("% of class"), ax.set_ylim(0, 100)
ax.grid(), ax.legend()
```



Amplitude by track period

```
stats = stats_compilation(
    a, groups, "amplitude", percentile(a.amplitude, bins_percentile)
)
fig = plt.figure()
ax = fig.add_subplot(111)
plot_stats(ax, **stats, cmap="magma")
ax.set_xlabel("Amplitude (m)"), ax.set_ylabel("% of class"), ax.set_ylim(0, 100)
ax.grid(), ax.legend()
```



Total running time of the script: (0 minutes 2.031 seconds)

6.6 Propagation Histogram

```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt
from numpy import arange, ones

from py_eddy_tracker.generic import cumsum_by_track
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load an experimental med atlas over a period of 26 years (1993-2019)

```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
nb_year = (a.period[1] - a.period[0] + 1) / 365.25
```

Filtering position to remove noisy position

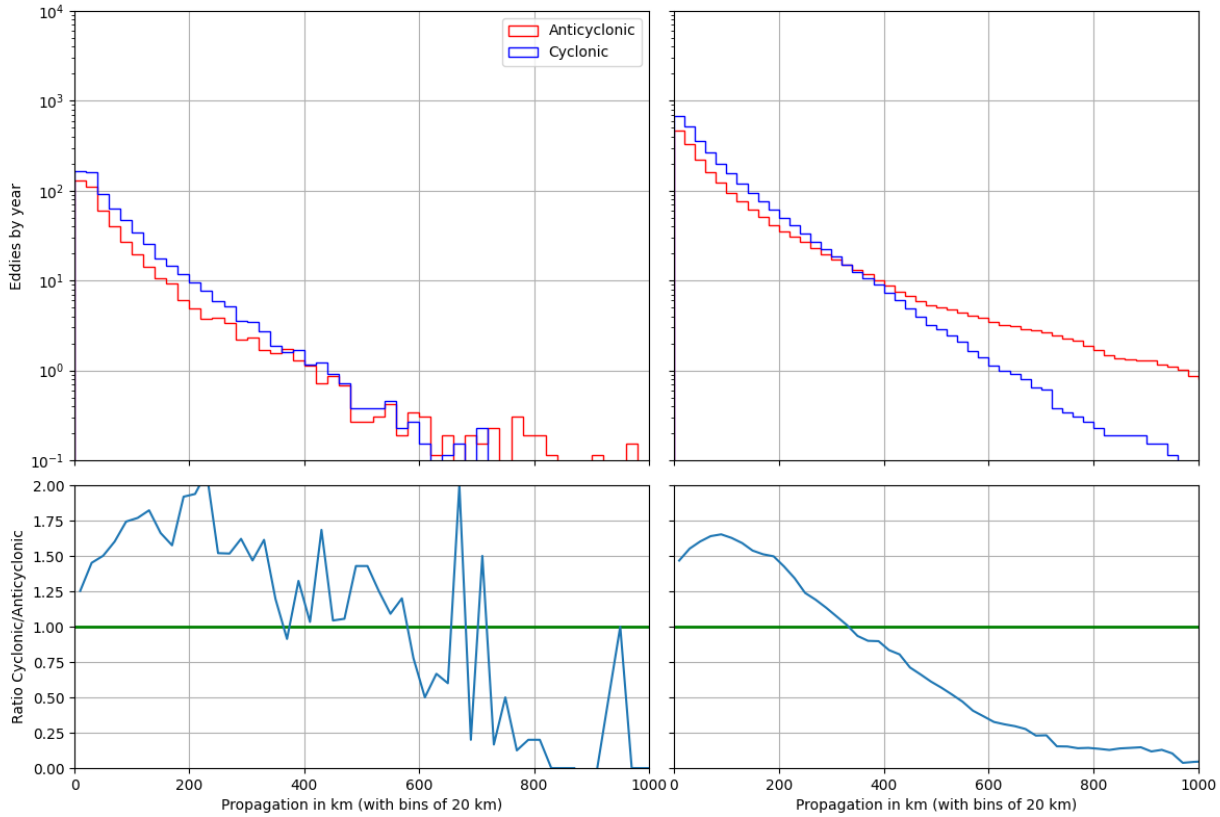
```
a.position_filter(median_half_window=1, loess_half_window=5)
c.position_filter(median_half_window=1, loess_half_window=5)
```

Compute curvilign distance

```
i0, nb = a.index_from_track, a.nb_obs_by_track
d_a = cumsum_by_track(a.distance_to_next(), a.tracks)[(i0 - 1 + nb)[nb != 0]] / 1000.0
i0, nb = c.index_from_track, c.nb_obs_by_track
d_c = cumsum_by_track(c.distance_to_next(), c.tracks)[(i0 - 1 + nb)[nb != 0]] / 1000.0
```

Setup axes

```
figure = plt.figure(figsize=(12, 8))
ax_ratio_cum = figure.add_axes([0.55, 0.06, 0.42, 0.34])
ax_ratio = figure.add_axes([0.07, 0.06, 0.46, 0.34])
ax_cum = figure.add_axes([0.55, 0.43, 0.42, 0.54])
ax = figure.add_axes([0.07, 0.43, 0.46, 0.54])
ax.set_ylabel("Eddies by year")
ax_ratio.set_ylabel("Ratio Cyclonic/Anticyclonic")
for ax_ in (ax, ax_cum, ax_ratio_cum, ax_ratio):
    ax_.set_xlim(0, 1000)
    if ax_ in (ax, ax_cum):
        ax_.set_ylim(1e-1, 1e4), ax_.set_yscale("log")
    else:
        ax_.set_xlabel("Propagation in km (with bins of 20 km)")
        ax_.set_ylim(0, 2)
        ax_.axhline(1, color="g", lw=2)
    ax_.grid()
ax_cum.xaxis.set_ticklabels([]), ax_cum.yaxis.set_ticklabels([])
ax.xaxis.set_ticklabels([]), ax_ratio_cum.yaxis.set_ticklabels([])

# plot data
bin_hist = arange(0, 2000, 20)
x = (bin_hist[1:] + bin_hist[:-1]) / 2.0
w_a, w_c = ones(d_a.shape) / nb_year, ones(d_c.shape) / nb_year
kwargs_a = dict(histtype="step", bins=bin_hist, x=d_a, color="r", weights=w_a)
kwargs_c = dict(histtype="step", bins=bin_hist, x=d_c, color="b", weights=w_c)
cum_a, _, _ = ax_cum.hist(cumulative=-1, **kwargs_a)
cum_c, _, _ = ax_cum.hist(cumulative=-1, **kwargs_c)
nb_a, _, _ = ax.hist(label="Anticyclonic", **kwargs_a)
nb_c, _, _ = ax.hist(label="Cyclonic", **kwargs_c)
ax_ratio_cum.plot(x, cum_c / cum_a)
ax_ratio.plot(x, nb_c / nb_a)
ax.legend()
```



Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_propagation.py:66: RuntimeWarning: invalid␣
↪value encountered in true_divide
  ax_ratio_cum.plot(x, cum_c / cum_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_propagation.py:67: RuntimeWarning: divide by␣
↪zero encountered in true_divide
  ax_ratio.plot(x, nb_c / nb_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.3.0/
↪examples/10_tracking_diagnostics/pet_propagation.py:67: RuntimeWarning: invalid␣
↪value encountered in true_divide
  ax_ratio.plot(x, nb_c / nb_a)
```

Total running time of the script: (0 minutes 4.234 seconds)

6.7 Count pixel used

Do Geo stat with frequency and compare with center count method: *Count center*

```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt
from matplotlib.colors import LogNorm

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load an experimental med atlas over a period of 26 years (1993-2019)

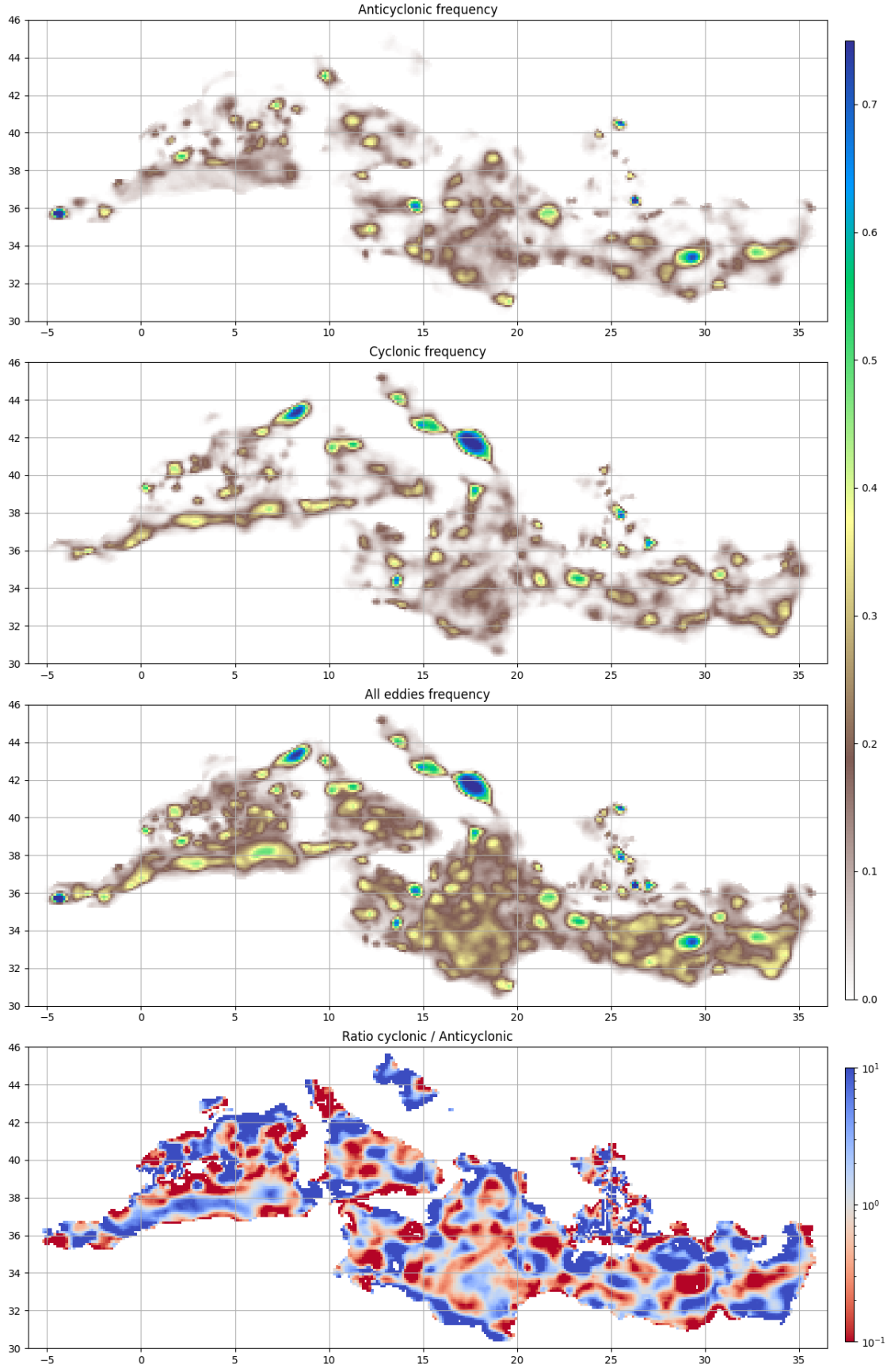
```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
```

Parameters

```
step = 0.125
bins = ((-10, 37, step), (30, 46, step))
kwargs_pcolormesh = dict(
    cmap="terrain_r", vmin=0, vmax=0.75, factor=1 / a.nb_days, name="count"
)
```

Plot

```
fig = plt.figure(figsize=(12, 18.5))
ax_a = fig.add_axes([0.03, 0.75, 0.90, 0.25])
ax_a.set_title("Anticyclonic frequency")
ax_c = fig.add_axes([0.03, 0.5, 0.90, 0.25])
ax_c.set_title("Cyclonic frequency")
ax_all = fig.add_axes([0.03, 0.25, 0.90, 0.25])
ax_all.set_title("All eddies frequency")
ax_ratio = fig.add_axes([0.03, 0.0, 0.90, 0.25])
ax_ratio.set_title("Ratio cyclonic / Anticyclonic")

# Count pixel used for each contour
g_a = a.grid_count(bins, intern=True)
g_a.display(ax_a, **kwargs_pcolormesh)
g_c = c.grid_count(bins, intern=True)
g_c.display(ax_c, **kwargs_pcolormesh)
# Compute a ratio Cyclonic / Anticyclonic
ratio = g_c.vars["count"] / g_a.vars["count"]

# Mask manipulation to be able to sum the 2 grids
m_c = g_c.vars["count"].mask
m = m_c & g_a.vars["count"].mask
g_c.vars["count"][m_c] = 0
g_c.vars["count"] += g_a.vars["count"]
g_c.vars["count"].mask = m

m = g_c.display(ax_all, **kwargs_pcolormesh)
plt.colorbar(m, cax=fig.add_axes([0.95, 0.27, 0.01, 0.7]))

g_c.vars["count"] = ratio
m = g_c.display(
    ax_ratio, name="count", vmin=0.1, vmax=10, norm=LogNorm(), cmap="coolwarm_r"
)
plt.colorbar(m, cax=fig.add_axes([0.95, 0.02, 0.01, 0.2]))

for ax in (ax_a, ax_c, ax_all, ax_ratio):
    ax.set_aspect("equal")
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
    ax.grid()
```

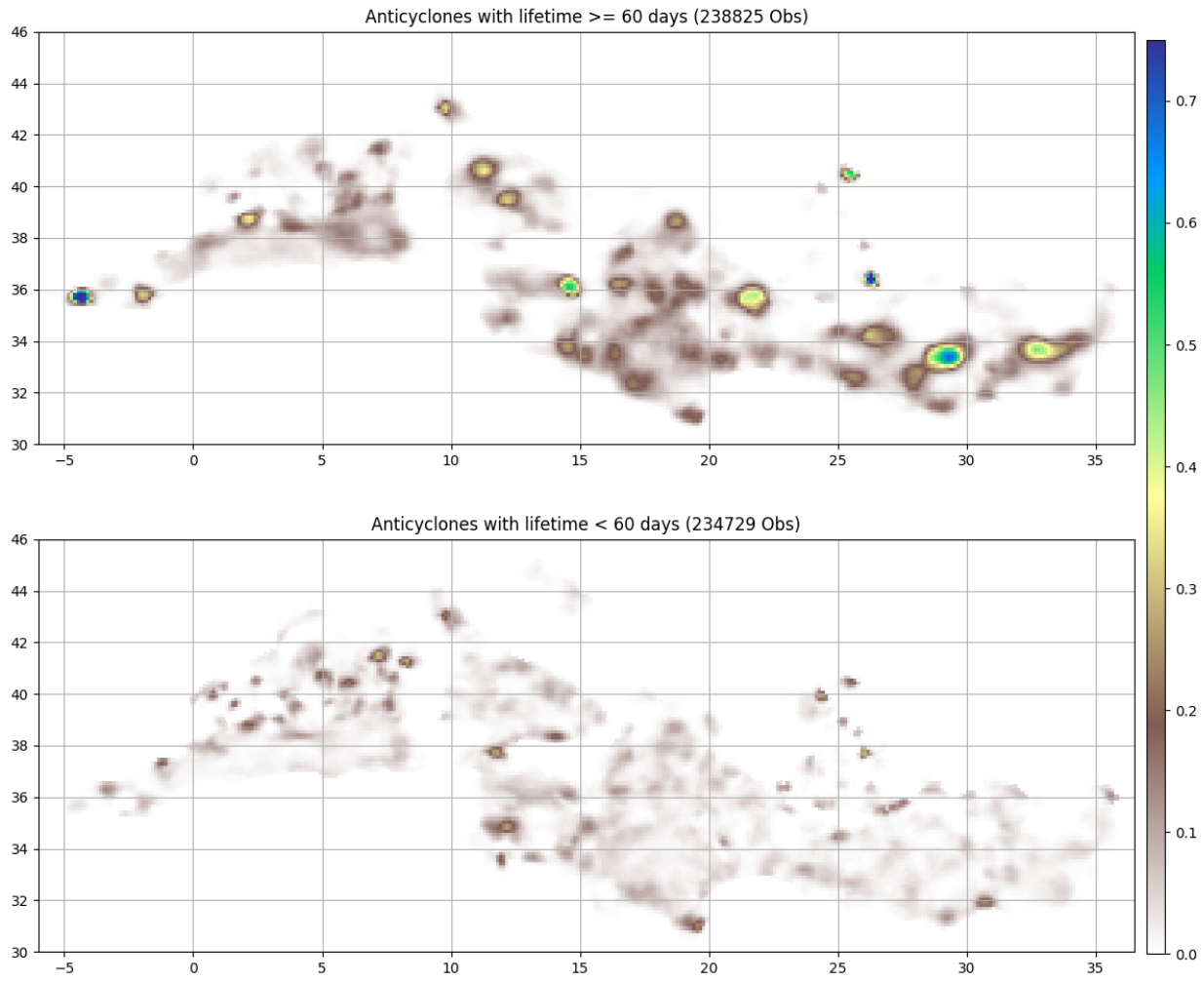

Anticyclonic frequency
Cyclonic frequency
All eddies frequency
Ratio cyclonic / Anticyclonic

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.3.0-py3.7.egg/py_eddy_tracker/dataset/grid.
↪py:1896: MatplotlibDeprecationWarning: Passing parameters norm and vmin/vmax␣
↪simultaneously is deprecated since 3.3 and will become an error two minor releases␣
↪later. Please pass vmin/vmax directly to the norm when creating it.
  return ax.pcolormesh(x, self.y_bounds, data.T * factor, **kwargs)
```

6.7.1 Count Anticyclones as a function of lifetime

```
fig = plt.figure(figsize=(12, 10))
mask = a.lifetime >= 60
ax_long = fig.add_axes([0.03, 0.53, 0.90, 0.45])
g_a = a.grid_count(bins, intern=True, filter=mask)
g_a.display(ax_long, **kwargs_pcolormesh)
ax_long.set_title(f"Anticyclones with lifetime >= 60 days ({mask.sum()} Obs)")
ax_short = fig.add_axes([0.03, 0.03, 0.90, 0.45])
g_a = a.grid_count(bins, intern=True, filter=~mask)
m = g_a.display(ax_short, **kwargs_pcolormesh)
ax_short.set_title(f"Anticyclones with lifetime < 60 days ({(~mask).sum()} Obs)")
for ax in (ax_short, ax_long):
    ax.set_aspect("equal"), ax.grid()
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
cb = plt.colorbar(m, cax=fig.add_axes([0.94, 0.05, 0.015, 0.9]))
```



Total running time of the script: (0 minutes 27.104 seconds)

6.8 Count center

Do Geo stat with center and compare with frequency method show: *Count pixel used*

```
import py_eddy_tracker_sample
from matplotlib import pyplot as plt
from matplotlib.colors import LogNorm

from py_eddy_tracker.observations.tracking import TrackEddiesObservations
```

Load an experimental med atlas over a period of 26 years (1993-2019)

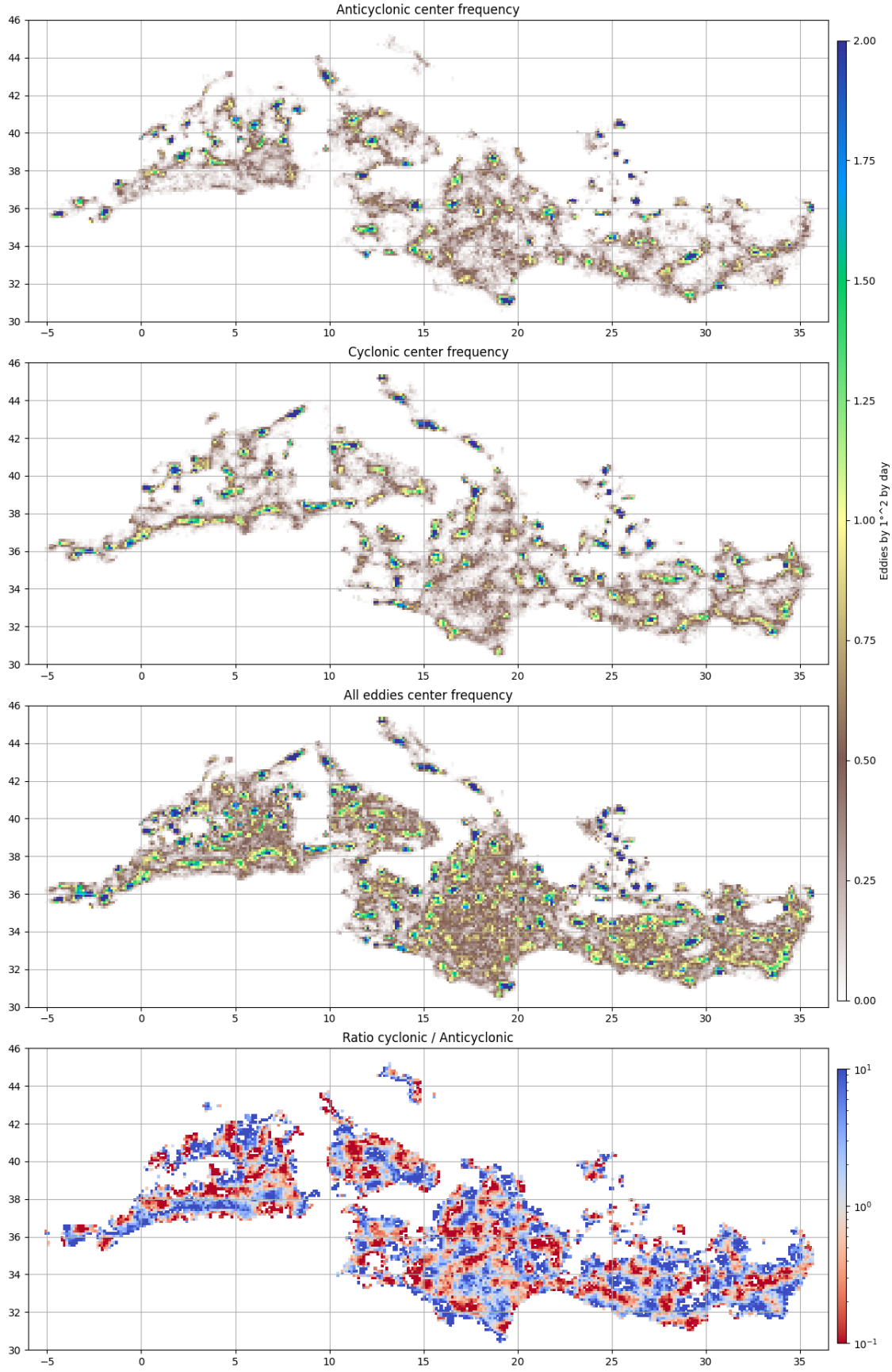
```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
```

Parameters

```
step = 0.125
bins = ((-10, 37, step), (30, 46, step))
kwargs_pcolormesh = dict(
    cmap="terrain_r", vmin=0, vmax=2, factor=1 / (a.nb_days * step ** 2), name="count"
)
```

Plot

```
fig = plt.figure(figsize=(12, 18.5))
ax_a = fig.add_axes([0.03, 0.75, 0.90, 0.25])
ax_a.set_title("Anticyclonic center frequency")
ax_c = fig.add_axes([0.03, 0.5, 0.90, 0.25])
ax_c.set_title("Cyclonic center frequency")
ax_all = fig.add_axes([0.03, 0.25, 0.90, 0.25])
ax_all.set_title("All eddies center frequency")
ax_ratio = fig.add_axes([0.03, 0.0, 0.90, 0.25])
ax_ratio.set_title("Ratio cyclonic / Anticyclonic")

# Count pixel used for each center
g_a = a.grid_count(bins, intern=True, center=True)
g_a.display(ax_a, **kwargs_pcolormesh)
g_c = c.grid_count(bins, intern=True, center=True)
g_c.display(ax_c, **kwargs_pcolormesh)
# Compute a ratio Cyclonic / Anticyclonic
ratio = g_c.vars["count"] / g_a.vars["count"]

# Mask manipulation to be able to sum the 2 grids
m_c = g_c.vars["count"].mask
m = m_c & g_a.vars["count"].mask
g_c.vars["count"][m_c] = 0
g_c.vars["count"] += g_a.vars["count"]
g_c.vars["count"].mask = m

m = g_c.display(ax_all, **kwargs_pcolormesh)
cb = plt.colorbar(m, cax=fig.add_axes([0.94, 0.27, 0.01, 0.7]))
cb.set_label("Eddies by 1°^2 by day")

g_c.vars["count"] = ratio
m = g_c.display(
    ax_ratio, name="count", vmin=0.1, vmax=10, norm=LogNorm(), cmap="coolwarm_r"
)
plt.colorbar(m, cax=fig.add_axes([0.94, 0.02, 0.01, 0.2]))

for ax in (ax_a, ax_c, ax_all, ax_ratio):
    ax.set_aspect("equal")
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
    ax.grid()
```


Anticyclonic center frequency
Cyclonic center frequency
All eddies center frequency
Ratio cyclonic / Anticyclonic
Eddies by 1°^2 by day
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
10^{1}
10^{0}
10^{-1}

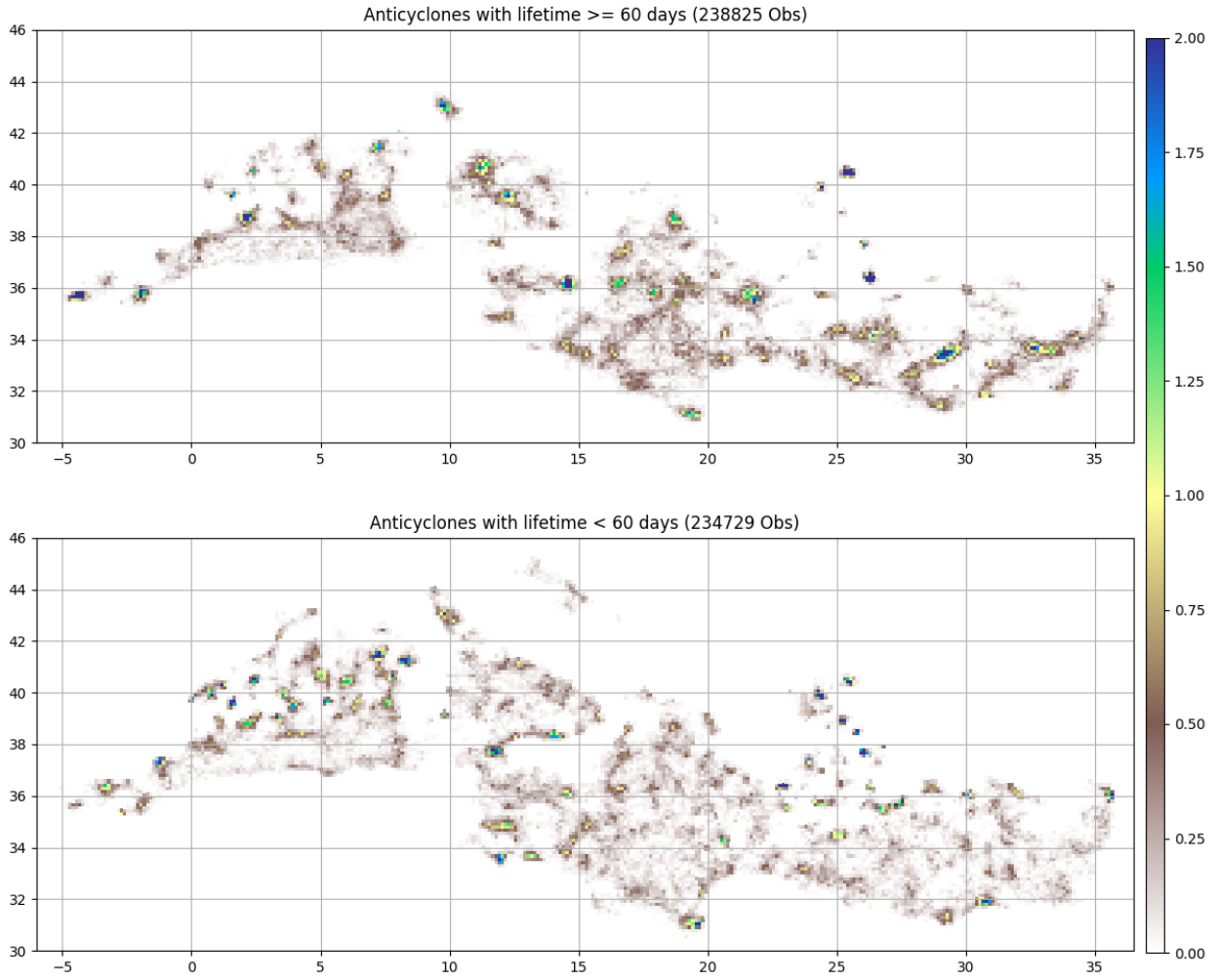
Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.3.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.3.0-py3.7.egg/py_eddy_tracker/dataset/grid.
↪py:1896: MatplotlibDeprecationWarning: Passing parameters norm and vmin/vmax␣
↪simultaneously is deprecated since 3.3 and will become an error two minor releases␣
↪later. Please pass vmin/vmax directly to the norm when creating it.
  return ax.pcolormesh(x, self.y_bounds, data.T * factor, **kwargs)
```

6.8.1 Count Anticyclones as a function of lifetime

Count at the center's position

```
fig = plt.figure(figsize=(12, 10))
mask = a.lifetime >= 60
ax_long = fig.add_axes([0.03, 0.53, 0.90, 0.45])
g_a = a.grid_count(bins, center=True, filter=mask)
g_a.display(ax_long, **kwargs_pcolormesh)
ax_long.set_title(f"Anticyclones with lifetime >= 60 days ({mask.sum()} Obs)")
ax_short = fig.add_axes([0.03, 0.03, 0.90, 0.45])
g_a = a.grid_count(bins, center=True, filter=~mask)
m = g_a.display(ax_short, **kwargs_pcolormesh)
ax_short.set_title(f"Anticyclones with lifetime < 60 days ({(~mask).sum()} Obs)")
for ax in (ax_short, ax_long):
    ax.set_aspect("equal"), ax.grid()
    ax.set_xlim(-6, 36.5), ax.set_ylim(30, 46)
cb = plt.colorbar(m, cax=fig.add_axes([0.94, 0.05, 0.015, 0.9]))
```



Total running time of the script: (0 minutes 3.789 seconds)

CHAPTER

SEVEN

EXTERNAL DATA

7.1 Collocating external data

Script will use py-eddy-tracker methods to upload external data (sea surface temperature, SST) in a common structure with altimetry.

Figures higlights the different steps.

```
from datetime import datetime

from matplotlib import pyplot as plt

from py_eddy_tracker import data
from py_eddy_tracker.dataset.grid import RegularGridDataset

date = datetime(2016, 7, 7)

filename_alt = data.get_path(f"dt_blacksea_allsat_phy_l4_{date:%Y%m%d}_20200801.nc")
filename_sst = data.get_path(
    f"{date:%Y%m%d}000000-GOS-L4_GHRSST-SSTfnd-OISST_HR_REP-BLK-v02.0-fv01.0.nc"
)
var_name_sst = "analysed_sst"

extent = [27, 42, 40.5, 47]
```

7.1.1 Loading data

```
sst = RegularGridDataset(filename=filename_sst, x_name="lon", y_name="lat")
alti = RegularGridDataset(
    data.get_path(filename_alt), x_name="longitude", y_name="latitude"
)
# We can use `Grid` tools to interpolate ADT on the sst grid
sst.regrid(alti, "sla")
sst.add_uv("sla")
```

Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/20160707000000-GOS-L4_GHRSST-SSTfnd-OISST_HR_REP-
↪BLK-v02.0-fv01.0.nc
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.3.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪3.0-py3.7.egg/py_eddy_tracker/data/dt_blacksea_allsat_phy_l4_20160707_20200801.nc
```

(continues on next page)

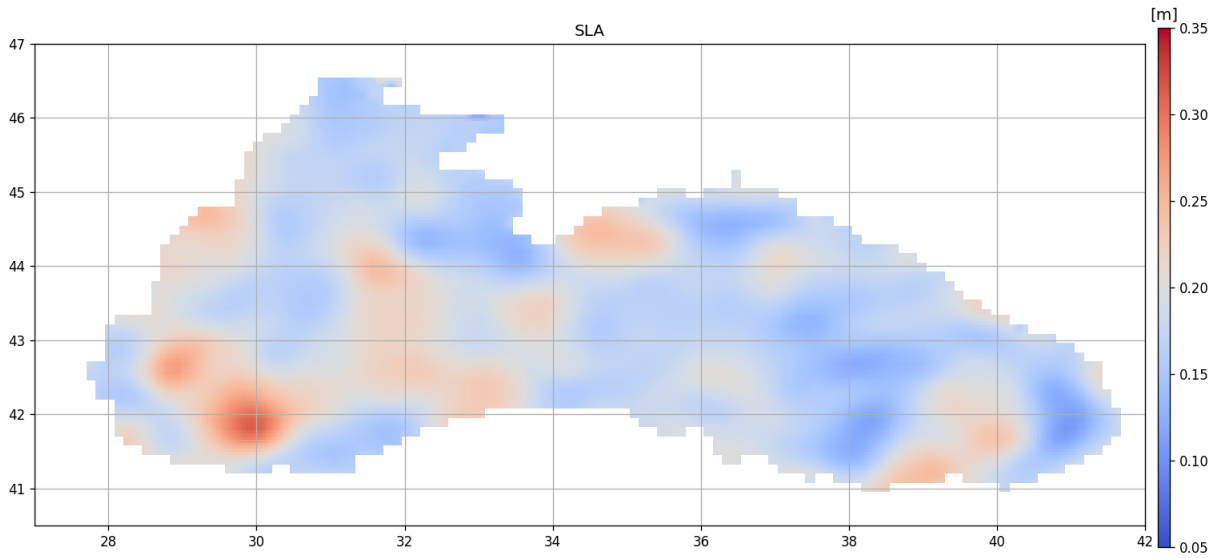
(continued from previous page)

Functions to initiate figure axes

```
def start_axes(title, extent=extent):
    fig = plt.figure(figsize=(13, 6), dpi=120)
    ax = fig.add_axes([0.03, 0.05, 0.89, 0.91])
    ax.set_xlim(extent[0], extent[1])
    ax.set_ylim(extent[2], extent[3])
    ax.set_title(title)
    ax.set_aspect("equal")
    return ax


def update_axes(ax, mappable=None, unit=""):
    ax.grid()
    if mappable:
        cax = ax.figure.add_axes([0.93, 0.05, 0.01, 0.9], title=unit)
        plt.colorbar(mappable, cax=cax)
```

7.1.2 ADT first display

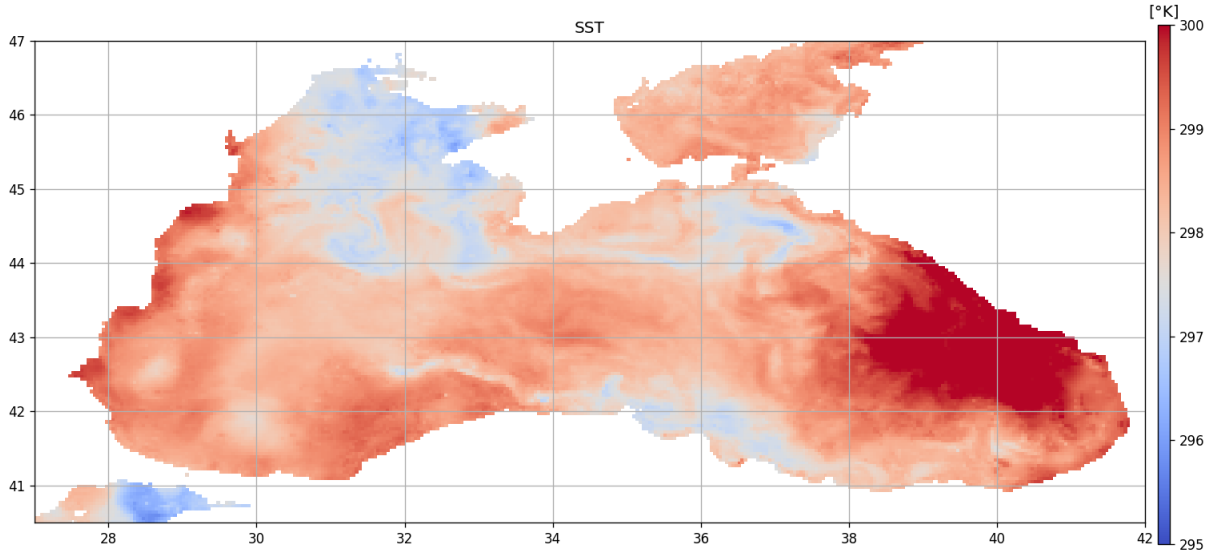
```
ax = start_axes("SLA", extent=extent)
m = sst.display(ax, "sla", vmin=0.05, vmax=0.35)
update_axes(ax, m, unit="[m]")
```



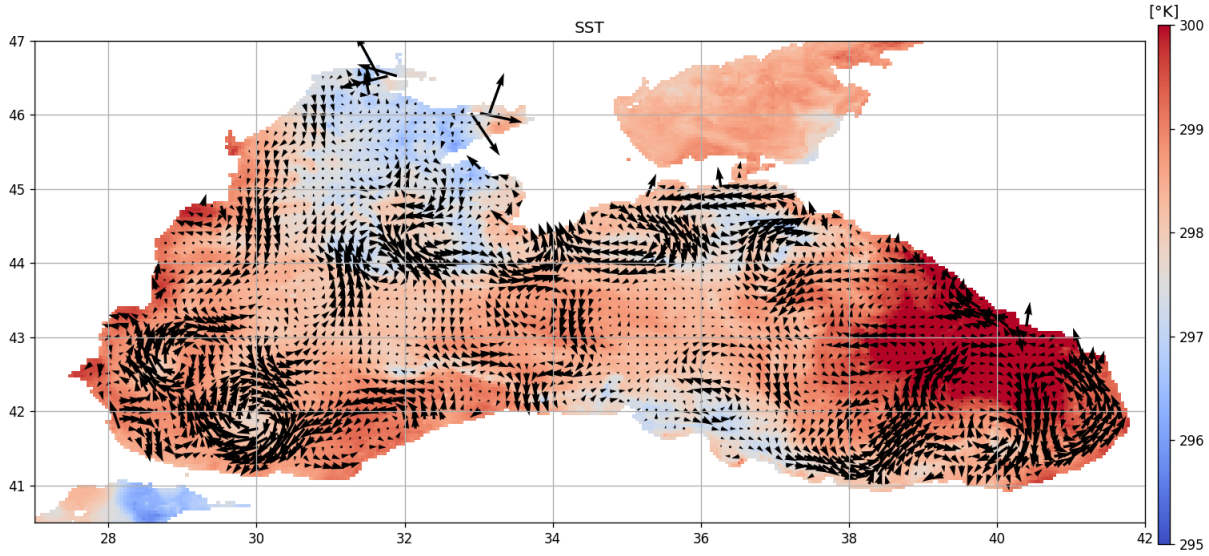
7.1.3 SST first display

We can now plot SST from *sst*

```
ax = start_axes("SST")
m = sst.display(ax, "analysed_sst", vmin=295, vmax=300)
update_axes(ax, m, unit="[°K]")
```




```
ax = start_axes("SST")
m = sst.display(ax, "analysed_sst", vmin=295, vmax=300)
u, v = sst.grid("u").T, sst.grid("v").T
ax.quiver(sst.x_c[::3], sst.y_c[::3], u[::3, ::3], v[::3, ::3], scale=10)
update_axes(ax, m, unit="[°K]")
```

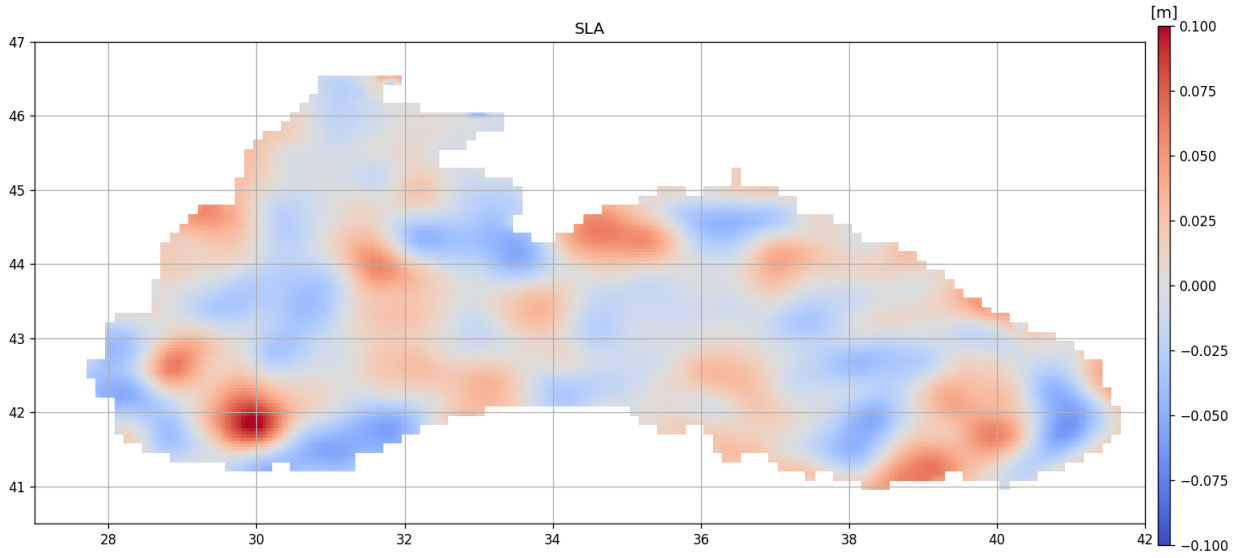


Now, with eddy contours, and displaying SST anomaly

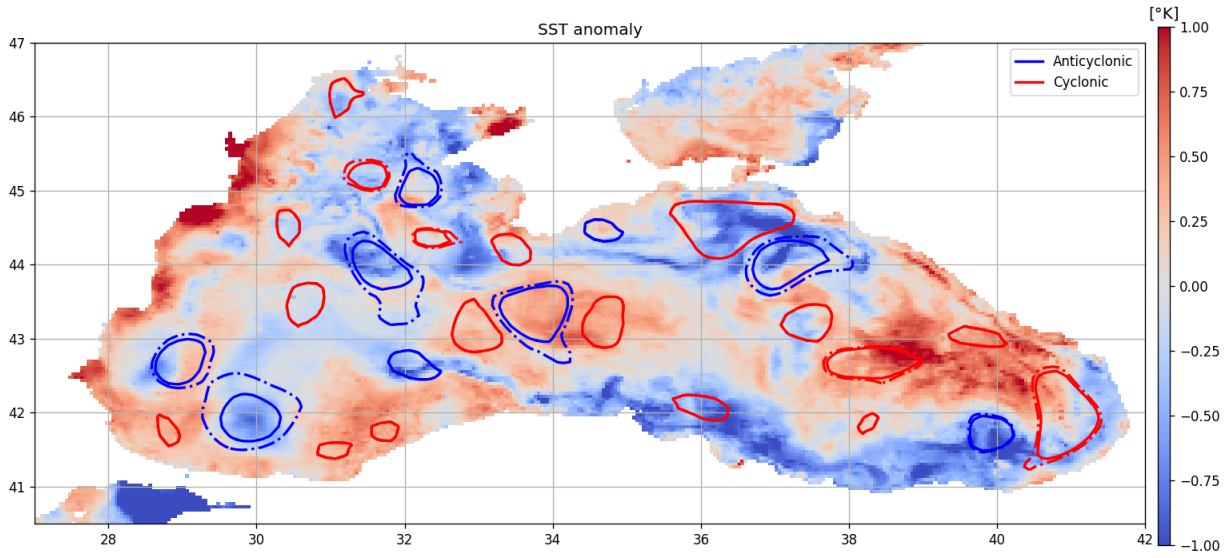
```
sst.bessel_high_filter("analysed_sst", 400)
```

Eddy detection

```
sst.bessel_high_filter("sla", 400)
# ADT filtered
ax = start_axes("SLA", extent=extent)
m = sst.display(ax, "sla", vmin=-0.1, vmax=0.1)
update_axes(ax, m, unit="[m]")
a, c = sst.eddy_identification("sla", "u", "v", date, 0.002)
```




```
kwargs_a = dict(lw=2, label="Anticyclonic", ref=-10, color="b")
kwargs_c = dict(lw=2, label="Cyclonic", ref=-10, color="r")
ax = start_axes("SST anomaly")
m = sst.display(ax, "analysed_sst", vmin=-1, vmax=1)
a.display(ax, **kwargs_a), c.display(ax, **kwargs_c)
ax.legend()
update_axes(ax, m, unit="[°K]")
```



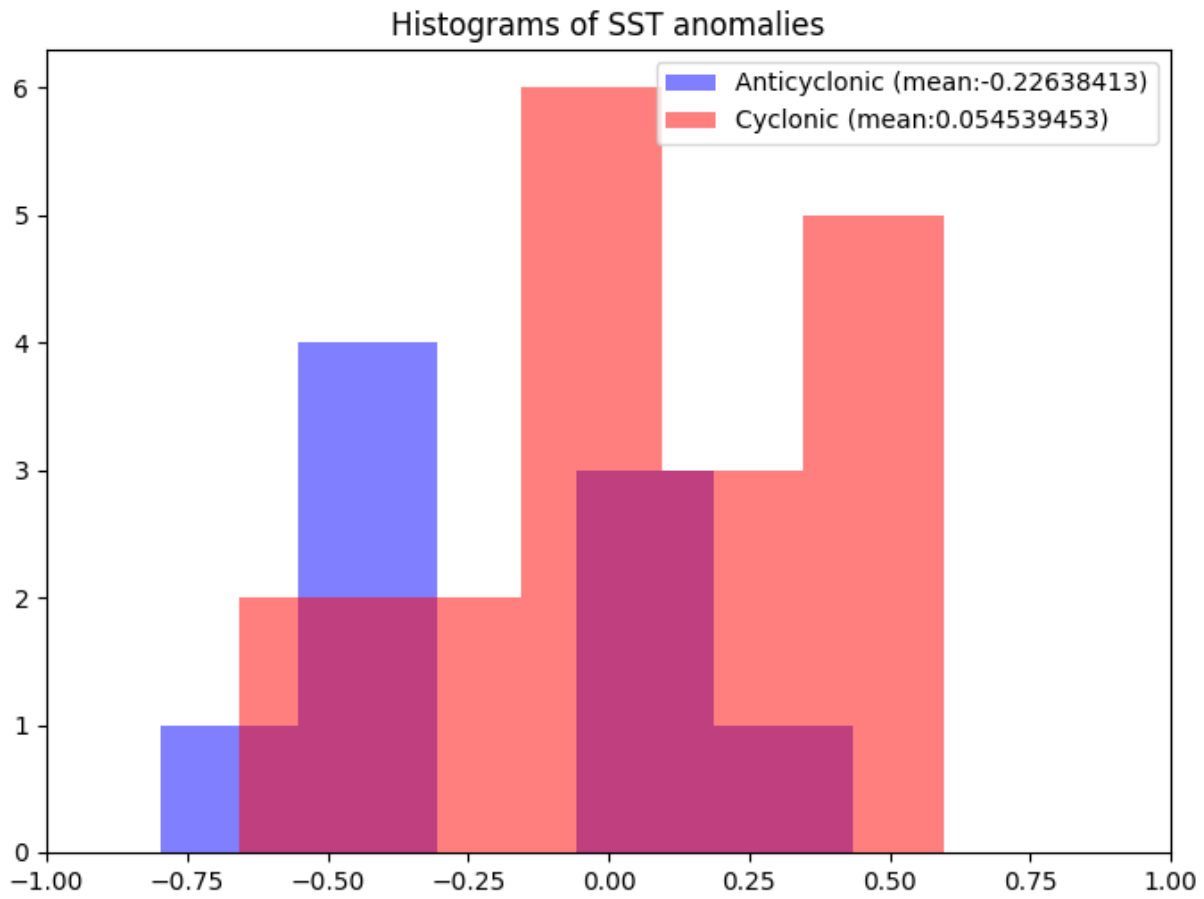
7.1.4 Example of post-processing

Get mean of sst anomaly_high in each internal contour

```
anom_a = a.interp_grid(sst, "analysed_sst", method="mean", intern=True)
anom_c = c.interp_grid(sst, "analysed_sst", method="mean", intern=True)
```

Are cyclonic (resp. anticyclonic) eddies generally associated with positive (resp. negative) SST anomaly ?

```
fig = plt.figure(figsize=(7, 5))
ax = fig.add_axes([0.05, 0.05, 0.90, 0.90])
ax.set_xlabel("SST anomaly")
ax.set_xlim([-1, 1])
ax.set_title("Histograms of SST anomalies")
ax.hist(
    anom_a, 5, alpha=0.5, color="b", label="Anticyclonic (mean:%s)" % (anom_a.mean())
)
ax.hist(anom_c, 5, alpha=0.5, color="r", label="Cyclonic (mean:%s)" % (anom_c.mean()))
ax.legend()
```



Not clearly so in that case ..

Total running time of the script: (0 minutes 10.201 seconds)

CHAPTER

EIGHT

EDDY IDENTIFICATION

Run the identification process for a single day

8.1 Shell/bash command

Bash command will allow to process one grid, it will apply a filter and an identification.

```
EddyId share/nrt_global_allsat_phy_l4_20190223_20190226.nc 20190223 \
    adt ugos vgos longitude latitude \
    out_directory -v DEBUG
```

Filter could be modify with options *–cut_wavelength* and *–filter_order*. You could also defined height between two isolines with *–isoline_step*, which could improve speed profile quality and detect accurately tiny eddies. You could also use *–fit_errmax* to manage acceptable shape of eddies.

An eddy identification will produce two files in the output directory, one for anticyclonic eddies and the other one for cyclonic.

In regional area which are away from the equator, current could be deduce from height, juste write *None None* inplace of *ugos vgos*

In case of **datacube**, you need to specify index for each layer (time, depth, . . .) wiht *–indexs* option like:

```
EddyId share/nrt_global_allsat_phy_l4_20190223_20190226.nc 20190223 \
    adt ugos vgos longitude latitude \
    out_directory -v DEBUG --indexs time=0
```

Warning: If no index are specified, you will apply identification only on dataset first layer, which could be a problem for datacube. Date set in command is used only for output storage.

8.2 Python code

If we want customize eddies identification, python module is here.

Activate verbose

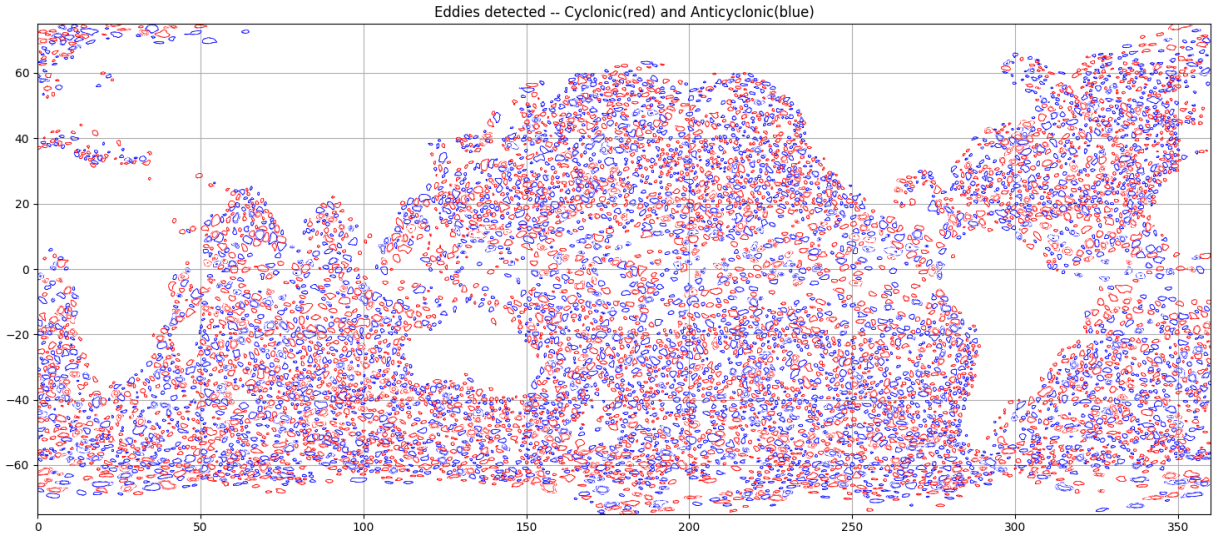
```
from py_eddy_tracker import start_logger
start_logger().setLevel('DEBUG') # Available options: ERROR, WARNING, INFO, DEBUG
```

Run identification

```
from datetime import datetime
h = RegularGridDataset(grid_name, lon_name, lat_name)
h.bessel_high_filter('adt', 500, order=3)
date = datetime(2019, 2, 23)
a, c = h.eddy_identification(
    'adt', 'ugos', 'vgos', # Variables used for identification
    date, # Date of identification
    0.002, # step between two isolines of detection (m)
    pixel_limit=(5, 2000), # Min and max pixel count for valid contour
    shape_error=55, # Error max (%) between ratio of circle fit and contour
    )
```

Plot the resulting identification

```
fig = plt.figure(figsize=(15,7))
ax = fig.add_axes([.03,.03,.94,.94])
ax.set_title('Eddies detected -- Cyclonic(red) and Anticyclonic(blue)')
ax.set_ylim(-75,75)
ax.set_xlim(0,360)
ax.set_aspect('equal')
a.display(ax, color='b', linewidth=.5)
c.display(ax, color='r', linewidth=.5)
ax.grid()
fig.savefig('share/png/eddies.png')
```



Save identification data

```
from netCDF import Dataset
with Dataset(date.strftime('share/Anticyclonic_%Y%m%d.nc'), 'w') as h:
    a.to_netcdf(h)
with Dataset(date.strftime('share/Cyclonic_%Y%m%d.nc'), 'w') as h:
    c.to_netcdf(h)
```

CHAPTER

NINE

LOAD, DISPLAY AND FILTERING

Loading grid

```
from py_eddy_tracker.dataset.grid import RegularGridDataset
grid_name, lon_name, lat_name = 'share/nrt_global_allsat_phy_l4_20190223_20190226.nc',
↪ 'longitude', 'latitude'
h = RegularGridDataset(grid_name, lon_name, lat_name)
```

Plotting grid

```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(14, 12))
ax = fig.add_axes([.02, .51, .9, .45])
ax.set_title('ADT (m)')
ax.set_ylim(-75, 75)
ax.set_aspect('equal')
m = h.display(ax, name='adt', vmin=-1, vmax=1)
ax.grid(True)
plt.colorbar(m, cax=fig.add_axes([.94, .51, .01, .45]))
```

Filtering

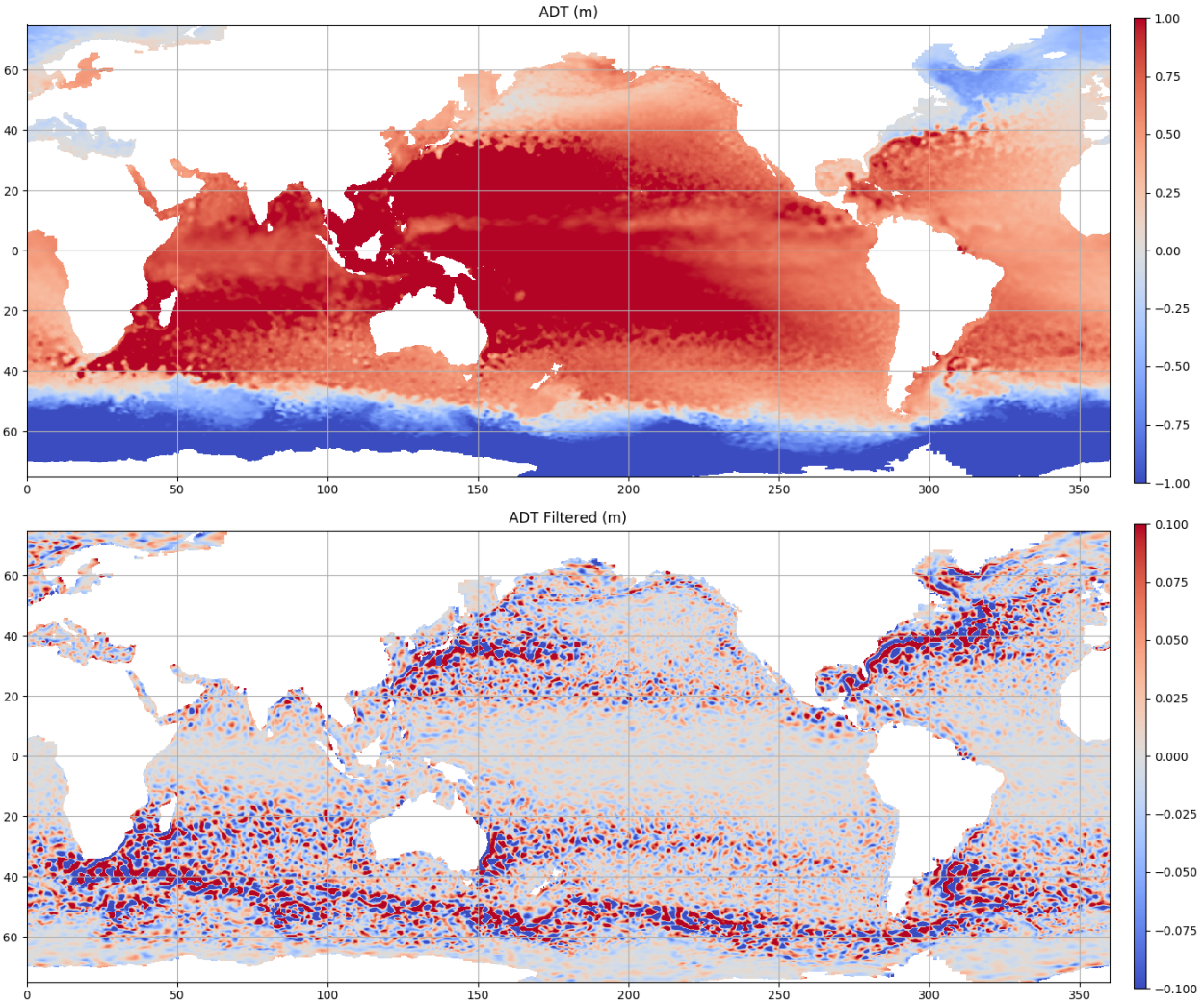
```
h = RegularGridDataset(grid_name, lon_name, lat_name)
h.bessel_high_filter('adt', 500, order=3)
```

Save grid

```
h.write('/tmp/grid.nc')
```

Add second plot

```
ax = fig.add_axes([.02, .02, .9, .45])
ax.set_title('ADT Filtered (m)')
ax.set_aspect('equal')
ax.set_ylim(-75, 75)
m = h.display(ax, name='adt', vmin=-.1, vmax=.1)
ax.grid(True)
plt.colorbar(m, cax=fig.add_axes([.94, .02, .01, .45]))
fig.savefig('share/png/filter.png')
```


ADT (m)
60
40
20
0
20
40
60
0
50
100
150
200
250
300
350
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
ADT Filtered (m)
60
40
20
0
20
40
60
0
50
100
150
200
250
300
350
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100

CHAPTER

TEN

SPECTRUM

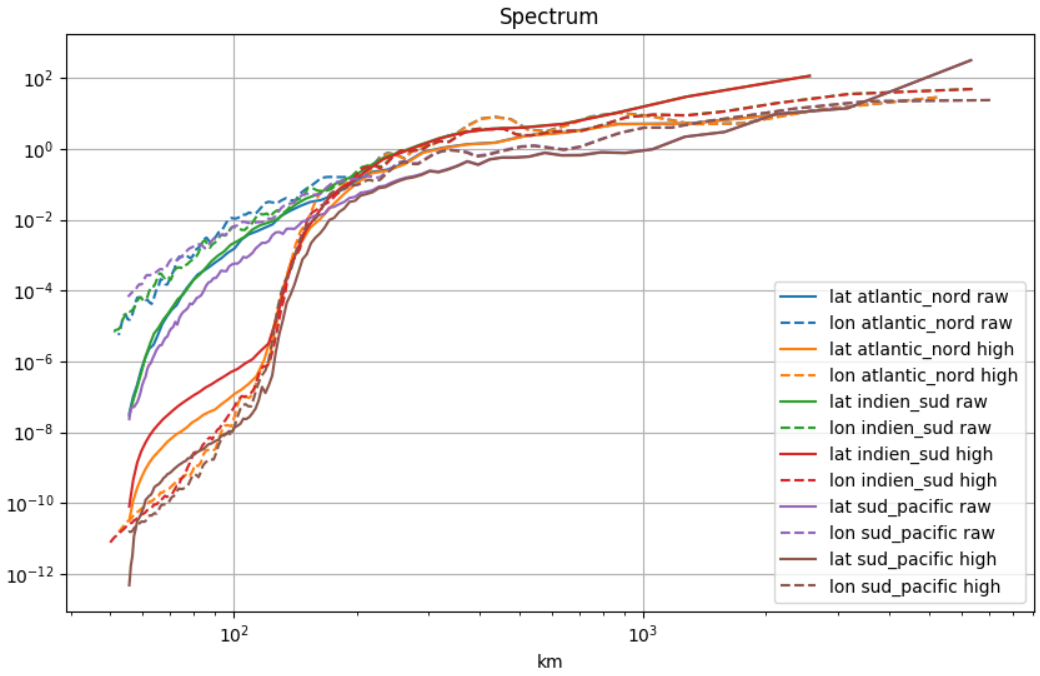
10.1 Compute spectrum and spectrum ratio on some area

Load data

```
raw = RegularGridDataset(grid_name, lon_name, lat_name)
filtered = RegularGridDataset(grid_name, lon_name, lat_name)
filtered.bessel_low_filter('adt', 150, order=3)

areas = dict(
    sud_pacific=dict(llcrnrlon=188, urcrnrlon=280, llcrnrlat=-64, urcrnrlat=-7),
    atlantic_nord=dict(llcrnrlon=290, urcrnrlon=340, llcrnrlat=19.5, urcrnrlat=43),
    indien_sud=dict(llcrnrlon=35, urcrnrlon=110, llcrnrlat=-49, urcrnrlat=-26),
)
```

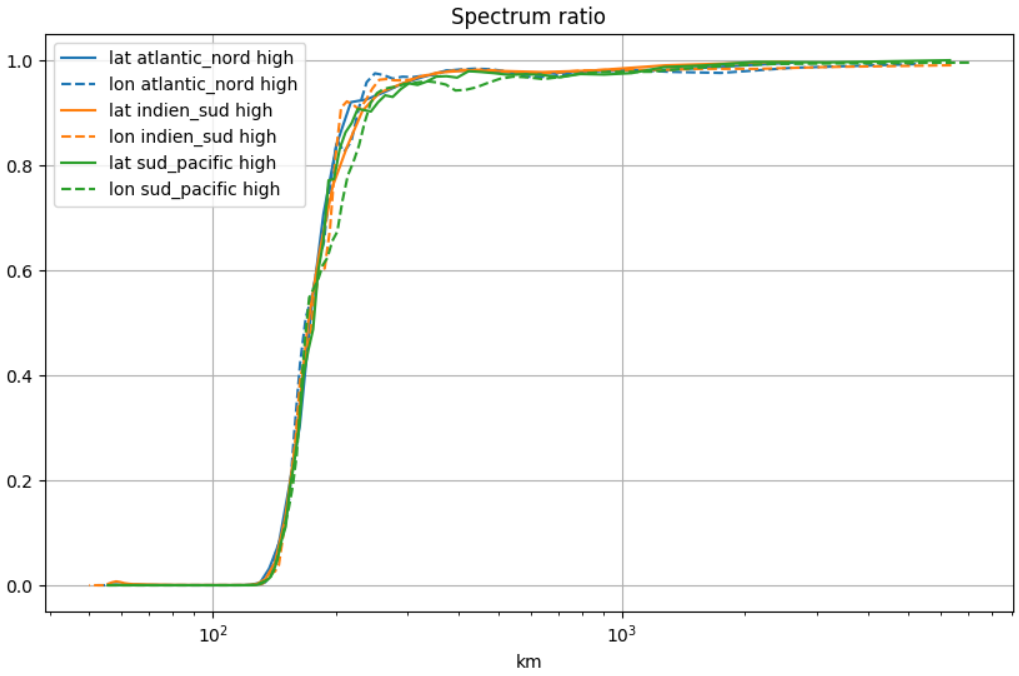
Compute and display spectrum

```
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(111)
ax.set_title('Spectrum')
ax.set_xlabel('km')
for name_area, area in areas.items():

    lon_spec, lat_spec = raw.spectrum_lonlat('adt', area=area)
    mappable = ax.loglog(*lat_spec, label='lat %s raw' % name_area)[0]
    ax.loglog(*lon_spec, label='lon %s raw' % name_area, color=mappable.get_color(),␣
↪linestyle='--')

    lon_spec, lat_spec = filtered.spectrum_lonlat('adt', area=area)
    mappable = ax.loglog(*lat_spec, label='lat %s high' % name_area)[0]
    ax.loglog(*lon_spec, label='lon %s high' % name_area, color=mappable.get_color(),␣
↪linestyle='--')

ax.set_xscale('log')
ax.legend()
ax.grid()
fig.savefig('share/png/spectrum.png')
```



Compute and display spectrum ratio

```
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(111)
ax.set_title('Spectrum ratio')
ax.set_xlabel('km')
for name_area, area in areas.items():
    lon_spec, lat_spec = filtered.spectrum_lonlat('adt', area=area, ref=raw)
    mappable = ax.plot(*lat_spec, label='lat %s high' % name_area)[0]
    ax.plot(*lon_spec, label='lon %s high' % name_area, color=mappable.get_color(),␣
↪linestyle='--')

ax.set_xscale('log')
ax.legend()
ax.grid()
fig.savefig('share/png/spectrum_ratio.png')
```


Spectrum ratio
lat atlantic_nord high
lon atlantic_nord high
lat indien_sud high
lon indien_sud high
lat sud_pacific high
lon sud_pacific high
1.0
0.8
0.6
0.4
0.2
0.0
10^2
10^3
km

CHAPTER

ELEVEN

TRACKING

11.1 Requirements

Before to run tracking, you will need to run identification on every time step of the period(Period of your study).

Advice : Before to run tracking, display some identification file allow to learn a lot

11.2 Default method

To run a tracking just create an yaml file with minimal specification (*FILES_PATTERN* and *SAVE_DIR*). You will run tracking separately between Cyclonic eddies and Anticyclonic eddies.

Example of conf.yaml

```
PATHS:
  # Files produces with EddyIdentification
  FILES_PATTERN: MY_IDENTIFICATION_PATH/Anticyclonic*.nc
  SAVE_DIR: MY_OUTPUT_PATH

# Number of timestep for missing detection
VIRTUAL_LENGTH_MAX: 3
# Minimal time to consider as a full track
TRACK_DURATION_MIN: 10
```

To run:

```
EddyTracking conf.yaml -v DEBUG
```

It will use default tracker:

- No travel longer than 125 km between two observation
- Amplitude and speed radius must be close to previous observation
- In case of several candidate only closest is kept

It will produce 4 files by run:

- A file of correspondances which will contains all the information to merge all identifications file
- A file which will contains all the observations which are alone
- A file which will contains all the short track which are shorter than **TRACK_DURATION_MIN**
- A file which will contains all the long track which are longer than **TRACK_DURATION_MIN**

11.3 Use python module

An example of tracking with python module is available in the gallery: *Track in python*

11.4 Choose a tracker

With yaml you could also select another tracker:

```
PATHS:
  # Files produces with EddyIdentification
  FILES_PATTERN: MY/IDENTIFICATION_PATH/Anticyclonic*.nc
  SAVE_DIR: MY_OUTPUT_PATH

# Number of timestep for missing detection
VIRTUAL_LENGTH_MAX: 3
# Minimal time to consider as a full track
TRACK_DURATION_MIN: 10

CLASS:
    # Give the module to import,
    # must be available when you do "import module" in python
    MODULE: py_eddy_tracker.featured_tracking.old_tracker_reference
    # Give class name which must be inherit from
    # py_eddy_tracker.observations.observation.EddiesObservations
    CLASS: CheltonTracker
```

This tracker is like described in CHELTON11[https://doi.org/10.1016/j.pocean.2011.01.002]. Code is here *py_eddy_tracker.featured_tracking.old_tracker_reference()*

CHAPTER

TWELVE

CUSTOMIZE TRACKING

12.1 Code my own tracking

To use your own tracking method, you just need to create a class which inherit from *py_eddy_tracker.observations.observation.EddiesObservations()* and set this class in yaml file like we see in the previous topic.

CHAPTER

THIRTEEN

API REFERENCE

py_eddy_tracker.appli	Entry point
py_eddy_tracker.dataset.grid	Class to load and manipulate RegularGrid and UnRegularGrid
py_eddy_tracker.featured_tracking	
py_eddy_tracker.observations.network	Class to create network of observations
py_eddy_tracker.observations.observation	Base class to manage eddy observation
py_eddy_tracker.observations.tracking	Class to manage observations gathered in track
py_eddy_tracker.eddy_feature	Class to compute Amplitude and average speed profile
py_eddy_tracker.generic	Tool method which use mostly numba
py_eddy_tracker.gui	GUI class
py_eddy_tracker.poly	Method for polygon
py_eddy_tracker.tracking	Class to store link between observations

13.1 py_eddy_tracker.appli

Entry point

py_eddy_tracker.appli.eddies	Applications on detection and tracking files
py_eddy_tracker.appli.grid	All entry point to manipulate grid
py_eddy_tracker.appli.gui	Entry point of graphic user interface
py_eddy_tracker.appli.misc	Entry point with no direct link with eddies
py_eddy_tracker.appli.network	Entry point to create and manipulate observations network

13.1.1 py_eddy_tracker.appli.eddies

Applications on detection and tracking files

Functions

display_infos
eddies_add_circle
get_frequency_grid
merge_eddies

py_eddy_tracker.appli.eddies.display_infos

py_eddy_tracker.appli.eddies.**display_infos**()

py_eddy_tracker.appli.eddies.eddies_add_circle

py_eddy_tracker.appli.eddies.**eddies_add_circle**()

py_eddy_tracker.appli.eddies.get_frequency_grid

py_eddy_tracker.appli.eddies.**get_frequency_grid**()

py_eddy_tracker.appli.eddies.merge_eddies

py_eddy_tracker.appli.eddies.**merge_eddies**()

13.1.2 py_eddy_tracker.appli.grid

All entry point to manipulate grid

Functions

eddy_id
filtering_parser
grid_filtering
identification

py_eddy_tracker.appli.grid.eddy_id

py_eddy_tracker.appli.grid.**eddy_id**(*args=None*)

py_eddy_tracker.appli.grid.filtering_parser

py_eddy_tracker.appli.grid.**filtering_parser**()

py_eddy_tracker.appli.grid.grid_filtering

py_eddy_tracker.appli.grid.**grid_filtering**()

py_eddy_tracker.appli.grid.identification

py_eddy_tracker.appli.grid.**identification**(*filename*, *lon*, *lat*, *date*, *h*, *u='None'*, *v='None'*, *unregular=False*, *cut_wavelength=500*, *filter_order=1*, *indexs=None*, ***kwargs*)

Classes

DictAction

py_eddy_tracker.appli.grid.DictAction

class py_eddy_tracker.appli.grid.**DictAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: argparse.Action

Methods

13.1.3 py_eddy_tracker.appli.gui

Entry point of graphic user interface

Functions

anim
gui_parser
guieddy

py_eddy_tracker.appli.gui.anim

py_eddy_tracker.appli.gui.**anim**()

py_eddy_tracker.appli.gui.gui_parser

py_eddy_tracker.appli.gui.**gui_parser**()

py_eddy_tracker.appli.gui.guieddy

py_eddy_tracker.appli.gui.**guieddy**()

Classes

Anim

py_eddy_tracker.appli.gui.Anim

class py_eddy_tracker.appli.gui.**Anim**(*eddy*, *intern=False*, *sleep_event=0.1*, *graphic_information=False*, ***kwargs*)

Bases: object

Methods

draw_contour
func_animation
keyboard
next
prev
reset_bliting
setup
show
update

draw_contour()

func_animation(*frame*)

keyboard(*event*)

next()

prev()

reset_bliting(*event*)

setup(*cmap='jet'*, *nb_step=25*, *figsize=(8, 6)*, ***kwargs*)

show(*infinity_loop=False*)

update()

13.1.4 py_eddy_tracker.appli.misc

Entry point with no direct link with eddies

Functions

zarr_header_parser	
zarrdump	

py_eddy_tracker.appli.misc.zarr_header_parser

py_eddy_tracker.appli.misc.**zarr_header_parser**()

py_eddy_tracker.appli.misc.zarrdump

py_eddy_tracker.appli.misc.**zarrdump**()

13.1.5 py_eddy_tracker.appli.network

Entry point to create and manipulate observations network

Functions

build_network	
build_track	
display_network	
divide_network	
next_obs	
set_tracks	
split_network	Divide each group in track

py_eddy_tracker.appli.network.build_network

py_eddy_tracker.appli.network.**build_network**()

py_eddy_tracker.appli.network.build_track

py_eddy_tracker.appli.network.**build_track**(*first_index*, *track_id*, *used*, *track*, *previous_observation*, *next_observation*, *ref_index*, *next_cost*, *previous_cost*, **args*)

py_eddy_tracker.appli.network.display_network

py_eddy_tracker.appli.network.**display_network**(*x*, *y*, *tr*, *t*, *c*)

py_eddy_tracker.appli.network.divide_network

py_eddy_tracker.appli.network.**divide_network**()

py_eddy_tracker.appli.network.next_obs

py_eddy_tracker.appli.network.**next_obs**(*i_current*, *next_cost*, *previous_cost*, *polygons*, *t*, *t_start*, *t_end*, *t_ref*, *window*)

py_eddy_tracker.appli.network.set_tracks

py_eddy_tracker.appli.network.**set_tracks**(*x*, *y*, *t*, *ref_index*, *track*, *previous_cost*, *next_cost*, *previous_observation*, *next_observation*, *window*)

py_eddy_tracker.appli.network.split_network

py_eddy_tracker.appli.network.**split_network**(*input*, *output*)

Divide each group in track

13.2 py_eddy_tracker.dataset.grid

Class to load and manipulate RegularGrid and UnRegularGrid

Functions

compute_pixel_path	Give a serie of indexes describing the path between two position
fit_circle_path	
has_masked_value	
has_value	
mean_on_regular_contour	

continues on next page

Table 12 – continued from previous page

pixels_in
raw_resample
uniform_resample_stack
value_on_regular_contour

13.2.1 py_eddy_tracker.dataset.grid.compute_pixel_path

py_eddy_tracker.dataset.grid.**compute_pixel_path**(*x0*, *y0*, *x1*, *y1*, *x_ori*, *y_ori*, *x_step*, *y_step*, *nb_x*)

Give a serie of indexes describing the path between two position

13.2.2 py_eddy_tracker.dataset.grid.fit_circle_path

py_eddy_tracker.dataset.grid.**fit_circle_path**(*self*, *method='fit'*)

13.2.3 py_eddy_tracker.dataset.grid.has_masked_value

py_eddy_tracker.dataset.grid.**has_masked_value**(*grid*, *i_x*, *i_y*)

13.2.4 py_eddy_tracker.dataset.grid.has_value

py_eddy_tracker.dataset.grid.**has_value**(*grid*, *i_x*, *i_y*, *value*, *below=False*)

13.2.5 py_eddy_tracker.dataset.grid.mean_on_regular_contour

py_eddy_tracker.dataset.grid.**mean_on_regular_contour**(*x_g*, *y_g*, *z_g*, *m_g*, *vertices*, *num_fac=2*, *fixed_size=None*, *nan_remove=False*)

13.2.6 py_eddy_tracker.dataset.grid.pixels_in

py_eddy_tracker.dataset.grid.**pixels_in**(*self*, *grid*)

13.2.7 py_eddy_tracker.dataset.grid.raw_resample

py_eddy_tracker.dataset.grid.**raw_resample**(*datas*, *fixed_size*)

13.2.8 py_eddy_tracker.dataset.grid.uniform_resample_stack

py_eddy_tracker.dataset.grid.**uniform_resample_stack**(*vertices*, *num_fac=2*, *fixed_size=None*)

13.2.9 py_eddy_tracker.dataset.grid.value_on_regular_contour

py_eddy_tracker.dataset.grid.**value_on_regular_contour**(*x_g*, *y_g*, *z_g*, *m_g*, *vertices*, *num_fac=2*, *fixed_size=None*)

Classes

GridDataset	Class to have basic tool on NetCDF Grid
RegularGridDataset	Class only for regular grid
UnRegularGridDataset	Class managing unregular grid

13.2.10 py_eddy_tracker.dataset.grid.GridDataset

class py_eddy_tracker.dataset.grid.**GridDataset**(*filename*, *x_name*, *y_name*, *centered=None*, *indexs=None*, *unset=False*)

Bases: object

Class to have basic tool on NetCDF Grid

Parameters

- **filename** (*str*) – Filename to load
- **x_name** (*str*) – Name of longitude coordinates
- **y_name** (*str*) – Name of latitude coordinates
- **centered** (*bool*, *None*) – Allow to know how coordinates could be used with pixel
- **indexs** (*dict*) – A dictionary which set indexs to use for non-coordinate dimensions
- **unset** (*bool*) – Set to True to create an empty grid object without file

Methods

add_grid	Add a grid in handler
c_to_bounds	Centred coordinates to bounds coordinates
copy	Duplicate the variable from grid_in in grid_out
eddy_identification	Compute eddy identification on the pecified grid
get_amplitude	
get_uavg	Calculate geostrophic speed around successive contours Returns the average
grid	Give the grid required
grid_tiles	Give the grid tiles required, without buffer system

continues on next page

Table 14 – continued from previous page

high_filter	Return the grid high-pass filtered, by substracting to the grid the low-pass filter (default: order=1)
is_circular	Check grid circularity
load	Load variable (data).
load_general_features	Load attrs to be stored in object
low_filter	Return the grid low-pass filtered (default: order=1)
setup_coordinates	
units	Get unit from variable
write	Write dataset output with same format as input

Attributes

EARTH_RADIUS	
GRAVITY	
N	
bounds	Give bounds
centered	
contours	
coordinates	
dimensions	
filename	
global_attrs	
indexs	
interpolators	
is_centered	Give True if pixel is described with its center's position or a corner
speed_coef	
variables	
variables_description	
vars	
x_bounds	
x_c	
x_dim	
xinterp	
y_bounds	
y_c	
y_dim	
yinterp	

EARTH_RADIUS = 6370997.0

GRAVITY = 9.807

N = 1

add_grid(*varname*, *grid*)
Add a grid in handler

Parameters

- **varname** (*str*) – name of the future grid
- **grid** (*array*) – grid array

property bounds
Give bounds

static c_to_bounds(*c*)
Centred coordinates to bounds coordinates

Parameters **c** (*array*) – centred coordinates to translate

Returns bounds coordinates

centered

contours

coordinates

copy(*grid_in*, *grid_out*)
Duplicate the variable from grid_in in grid_out

Parameters

- **grid_in** –
- **grid_out** –

dimensions

eddy_identification(*grid_height*, *uname*, *vname*, *date*, *step=0.005*, *shape_error=55*, *sampling=50*, *pixel_limit=None*, *precision=None*, *force_height_unit=None*, *force_speed_unit=None*, ***kwargs*)
Compute eddy identification on the pecified grid

Parameters

- **grid_height** (*str*) – Grid name of Sea Surface Height
- **uname** (*str*) – Grid name of u speed component
- **vname** (*str*) – Grid name of v speed component
- **date** (*datetime.datetime*) – Date which will be stored in object to date data
- **step** (*float*, *int*) – Height between two layers in m
- **shape_error** (*float*, *int*) – Maximal error allowed for outter contour in %
- **sampling** (*int*) – Number of points to store contours and speed profile
- **pixel_limit** ((*int*, *int*), *None*) – Min and max number of pixels inside the inner and the outer contour to be considered as an eddy
- **precision** (*float*, *None*) – Truncate values at the defined precision in m
- **force_height_unit** (*str*) – Unit used for height unit
- **force_speed_unit** (*str*) – Unit used for speed unit
- **kwargs** (*dict*) – Argument given to amplitude

Returns Return a list of 2 elements: Anticyclone and Cyclone

Return type *py_eddy_tracker.observations.observation.EddiesObservations*

- *Eddy detection : Med*
- *Eddy detection : Gulf stream*
- *Eddy detection and filter*

- *Eddy detection on SLA and ADT*
- *Collocating external data*

filename

static get_amplitude(*contour*, *contour_height*, *data*, *anticyclonic_search=True*, *level=None*, ***kwargs*)

get_uavg(*all_contours*, *centlon_e*, *centlat_e*, *original_contour*, *anticyclonic_search*, *level_start*, *pixel_min=3*)

Calculate geostrophic speed around successive contours Returns the average

global_attrs

grid(*varname*, *indexs=None*)

Give the grid required

Parameters

- **varname** (*str*) – Variable to get
- **indexs** (*dict*, *None*) – If defined dict must have dimensions name as key

Returns array asked, reduced by the indexes

Return type array

- *Shape error gallery*
- *Get mean of grid in each eddies*
- *Eddy detection : Med*
- *Eddy detection : Gulf stream*
- *Eddy detection and filter*
- *Select pixel in eddies*
- *Get Okubo Weis*
- *Collocating external data*

grid_tiles(*varname*, *slice_x*, *slice_y*)

Give the grid tiles required, without buffer system

high_filter(*grid_name*, *w_cut*, ***kwargs*)

Return the grid high-pass filtered, by substracting to the grid the low-pass filter (default: order=1)

Parameters

- **grid_name** – the name of the grid
- **w_cut** (*int*,) – the half-power wavelength cutoff (km)

indexs

interpolators

property is_centered

Give True if pixel is described with its center's position or a corner

Returns True if centered

Return type bool

is_circular()
Check grid circularity

load()
Load variable (data). Get coordinates and setup coordinates function

load_general_features()
Load attrs to be stored in object

low_filter(*grid_name*, *w_cut*, ***kwargs*)
Return the grid low-pass filtered (default: order=1)

Parameters

- **grid_name** – the name of the grid
- **w_cut** (*int,*) – the half-power wavelength cutoff (km)

setup_coordinates()

speed_coef

units(*varname*)
Get unit from variable

property variables

variables_description

vars

write(*filename*)
Write dataset output with same format as input

Parameters **filename** (*str*) – filename used to save the grid

x_bounds

x_c

x_dim

xinterp

y_bounds

y_c

y_dim

yinterp

13.2.11 py_eddy_tracker.dataset.grid.RegularGridDataset

class py_eddy_tracker.dataset.grid.**RegularGridDataset**(**args*, ***kwargs*)
Bases: *py_eddy_tracker.dataset.grid.GridDataset*

Class only for regular grid

Parameters

- **filename** (*str*) – Filename to load
- **x_name** (*str*) – Name of longitude coordinates
- **y_name** (*str*) – Name of latitude coordinates
- **centered** (*bool*,*None*) – Allow to know how coordinates could be used with pixel
- **indexs** (*dict*) – A dictionary which set indexs to use for non-coordinate dimensions
- **unset** (*bool*) – Set to True to create an empty grid object without file

Methods

add_grid	Add a grid in handler
add_uv	Compute a u and v grid
add_uv_lagerloef	
bbox_indice	
bessel_band_filter	
bessel_high_filter	**param str grid_name** grid to filter, data will replace original one
bessel_low_filter	
c_to_bounds	Centred coordinates to bounds coordinates
check_order	
clean_land	Function to remove all land pixel
compute_finite_difference	
compute_pixel_path	Give a series of indexes which describe the path between to position
compute_stencil	Apply stencil ponderation on field.
contour	**param matplotlib.axes.Axes ax** matplotlib axes use to draw
convolve_filter_with_dynamic_kernel	**param str grid** grid name
copy	Duplicate the variable from grid_in in grid_out
display	**param matplotlib.axes.Axes ax** matplotlib axes use to draw
eddy_identification	Compute eddy identification on the pecified grid
estimate_kernel_shape	
finalize_kernel	

continues on next page

Table 16 – continued from previous page

get_amplitude	
get_pixels_in	Get indices of pixels in contour.
get_step_in_km	
get_uavg	Calculate geostrophic speed around successive contours Returns the average
grid	Give the grid required
grid_tiles	Give the grid tiles required, without buffer system
high_filter	Return the grid high-pass filtered, by substracting to the grid the low-pass filter (default: order=1)
init_pos_interpolator	Create function to have a quick index interpolator
init_speed_coef	Draft
interp	Compute z over lons, lats
is_circular	Check if the grid is circular
kernel_bessel	wave_length in km order must be int
kernel_lanczos	Not really operational wave_length in km order must be int
lanczos_high_filter	
lanczos_low_filter	
load	Load variable (data).
load_general_features	Load attrs to be stored in object
low_filter	Return the grid low-pass filtered (default: order=1)
nearest_grd_indice	
normalize_x_indice	
regrid	Interpolate another grid at the current grid position
setup_coordinates	
spectrum_lonlat	
speed_coef_mean	Some nan can be computed over contour if we are near border, something to explore
units	Get unit from variable
with_array	
write	Write dataset output with same format as input

Attributes

EARTH_RADIUS	
GRAVITY	
N	
bounds	Give bounds
centered	
contours	
coordinates	
dimensions	
filename	
global_attrs	
indexs	
interpolators	
is_centered	Give True if pixel is described with its center's position or a corner
speed_coef	

continues on next page

Table 17 – continued from previous page

variables	
variables_description	
vars	
x_bounds	
x_c	
x_dim	
x_size	
xinterp	
xstep	Only for regular grid with no step variation
y_bounds	
y_c	
y_dim	
yinterp	
ystep	Only for regular grid with no step variation

add_uv(*grid_height*, *uname='u'*, *vname='v'*, *stencil_halfwidth=4*)

Compute a u and v grid

Parameters

- **grid_height** (*str*) – grid name where the funtion will apply stencil method
- **uname** (*str*) – future name of u
- **vname** (*str*) – future name of v
- **stencil_halfwidth** (*int*) – largest stencil could be apply (max: 4)

- *Get mean of grid in each eddies*
- *Eddy detection : Med*
- *Eddy detection : Gulf stream*
- *Eddy detection and filter*
- *Eddy detection on SLA and ADT*
- *Collocating external data*

add_uv_lagerloef(*grid_height*, *uname='u'*, *vname='v'*, *schema=15*)

bbox_indice(*vertices*)

bessel_band_filter(*grid_name*, *wave_length_inf*, *wave_length_sup*, ***kwargs*)

bessel_high_filter(*grid_name*, *wave_length*, *order=1*, *lat_max=85*, ***kwargs*)

Parameters

- **grid_name** (*str*) – grid to filter, data will replace original one
- **wave_length** (*float*) – in km
- **order** (*int*) – order to use, if > 1 negative values of the cardinal sinus are present in kernel
- **lat_max** (*float*) – absolute latitude above no filtering apply

- **kwargs** (*dict*) – look at *RegularGridDataset.convolve_filter_with_dynamic_kernel()*

- *Eddy detection : Med*
- *Eddy detection : Gulf stream*
- *Eddy detection and filter*
- *Eddy detection on SLA and ADT*
- *Grid filtering in PET*
- *Collocating external data*

bessel_low_filter(*grid_name*, *wave_length*, *order=1*, *lat_max=85*, ***kwargs*)

static check_order(*order*)

clean_land()
Function to remove all land pixel

compute_finite_difference(*data*, *schema=1*, *mode='reflect'*, *vertical=False*)

compute_pixel_path(*x0*, *y0*, *x1*, *y1*)
Give a series of indexes which describe the path between to position

compute_stencil(*data*, *stencil_halfwidth=4*, *mode='reflect'*, *vertical=False*)
Apply stencil ponderation on field.

Parameters

- **data** (*array*) – array where apply stencil
- **stencil_halfwidth** (*int*) – from 1 t0 4, maximal stencil used
- **mode** (*str*) – convolution mode
- **vertical** (*bool*) – if True, method apply a vertical convolution

Returns gradient array from stencil application

Return type array

Short story, how to get stencil coefficient for stencil (3 points, 5 points and 7 points)

Taylor's theorem:

$$f(x \pm h) = f(x) \pm f'(x)h + \frac{f''(x)h^2}{2!} \pm \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!} \pm \frac{f^{(5)}(x)h^5}{5!} + O(h^6)$$

If we stop at *O(h^2)*, we get classic differenciation (stencil 3 points):

$$f(x+h) - f(x-h) = f(x) - f(x) + 2f'(x)h + O(h^2)$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

If we stop at *O(h^4)*, we will get stencil 5 points:

$$f(x+h) - f(x-h) = 2f'(x)h + 2\frac{f^{(3)}(x)h^3}{3!} + O(h^4) \tag{13.1}$$

$$f(x+2h) - f(x-2h) = 4f'(x)h + 16\frac{f^{(3)}(x)h^3}{3!} + O(h^4) \tag{13.2}$$

If we multiply equation (13.1) by 8 and substract equation (13.2), we get:

$$8(f(x+h) - f(x-h)) - (f(x+2h) - f(x-2h)) = 16f'(x)h - 4f'(x)h + O(h^4)$$

$$f'(x) = \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} + O(h^4)$$

If we stop at *O(h^6)*, we will get stencil 7 points:

$$f(x+h) - f(x-h) = 2f'(x)h + 2\frac{f^{(3)}(x)h^3}{3!} + 2\frac{f^{(5)}(x)h^5}{5!} + O(h^6) \tag{13.3}$$

$$f(x+2h) - f(x-2h) = 4f'(x)h + 16\frac{f^{(3)}(x)h^3}{3!} + 64\frac{f^{(5)}(x)h^5}{5!} + O(h^6) \tag{13.4}$$

$$f(x+3h) - f(x-3h) = 6f'(x)h + 54\frac{f^{(3)}(x)h^3}{3!} + 486\frac{f^{(5)}(x)h^5}{5!} + O(h^6) \tag{13.5}$$

If we multiply equation (13.3) by 45 and substract equation (13.4) multiply by 9 and add equation (13.5), we get:

$$45(f(x+h) - f(x-h)) - 9(f(x+2h) - f(x-2h)) + (f(x+3h) - f(x-3h)) = 90f'(x)h - 36f'(x)h + 6f'(x)h -$$

$$f'(x) = \frac{-f(x-3h) + 9f(x-2h) - 45f(x-h) + 45f(x+h) - 9f(x+2h) + f(x+3h)}{60h} + O(h^6)$$

...

contour(*ax*, *name*, *factor=1*, *ref=None*, ***kwargs*)

Parameters

- **ax** (*matplotlib.axes.Axes*) – matplotlib axes use to draw
- **name** (*str,array*) – variable to display, could be an array
- **factor** (*float*) – multiply grid by
- **ref** (*float,None*) – if define use like west bound
- **kwargs** (*dict*) – look at `matplotlib.axes.Axes.contour()`

convolve_filter_with_dynamic_kernel(*grid*, *kernel_func*, *lat_max=85*, *extend=False*, ***kwargs_func*)

Parameters

- **grid** (*str*) – grid name
- **kernel_func** (*func*) – function of kernel to use
- **lat_max** (*float*) – absolute latitude above no filtering apply
- **extend** (*bool*) – if False, only non masked value will return a filtered value
- **kwargs_func** (*dict*) – look at kernel_func

Returns filtered value

Return type array

display(*ax*, *name*, *factor=1*, *ref=None*, ***kwargs*)

Parameters

- **ax** (*matplotlib.axes.Axes*) – matplotlib axes use to draw

- **name** (*str,array*) – variable to display, could be an array
- **factor** (*float*) – multiply grid by
- **ref** (*float,None*) – if define use like west bound
- **kwargs** (*dict*) – look at `matplotlib.axes.Axes.pcolormesh()`

- *Get mean of grid in each eddies*
- *Eddy detection : Med*
- *Eddy detection : Gulf stream*
- *Eddy detection and filter*
- *Eddy detection on SLA and ADT*
- *Select pixel in eddies*
- *Grid filtering in PET*
- *Get Okubo Weis*
- *Geographical statistics*
- *Birth and death*
- *Count pixel used*
- *Count center*
- *Collocating external data*

estimate_kernel_shape(*lat*, *wave_length*, *order*)

finalize_kernel(*kernel*, *order*, *half_x_pt*, *half_y_pt*)

get_pixels_in(*contour*)
Get indices of pixels in contour.

Parameters **contour** (*vertice,Path*) – Contour which enclosed some pixels

Returns Indices of grid in contour

Return type array[int],array[int]

get_step_in_km(*lat*, *wave_length*)

init_pos_interpolator()
Create function to have a quick index interpolator

init_speed_coef(*uname='u'*, *vname='v'*)
Draft

interp(*grid_name*, *lons*, *lats*, *method='bilinear'*)
Compute z over lons, lats

Parameters

- **grid_name** (*str*) – Grid to be interpolated

- **lons** – new x
- **lats** – new y
- **method** (*str*) – Could be 'bilinear' or 'nearest'

Returns new z

is_circular()
Check if the grid is circular

kernel_bessel(*lat*, *wave_length*, *order=1*)
wave_length in km order must be int

kernel_lanczos(*lat*, *wave_length*, *order=1*)
Not really operational wave_length in km order must be int

lanczos_high_filter(*grid_name*, *wave_length*, *order=1*, *lat_max=85*, ***kwargs*)

lanczos_low_filter(*grid_name*, *wave_length*, *order=1*, *lat_max=85*, ***kwargs*)

nearest_grd_indice(*x*, *y*)

normalize_x_indice(*indices*)

regrid(*other*, *grid_name*, *new_name=None*)
Interpolate another grid at the current grid position

Parameters

- **other** (RegularGridDataset) –
- **grid_name** (*str*) – variable name to interpolate
- **new_name** (*str*) – name used to store, if None method will use current ont

- *Collocating external data*

setup_coordinates()

spectrum_lonlat(*grid_name*, *area=None*, *ref=None*, ***kwargs*)

speed_coef_mean(*contour*)
Some nan can be computed over contour if we are near border, something to explore

classmethod with_array(*coordinates*, *datas*, *variables_description=None*, ***kwargs*)

x_size

property xstep
Only for regular grid with no step variation

property ystep
Only for regular grid with no step variation

13.2.12 py_eddy_tracker.dataset.grid.UnRegularGridDataset

class py_eddy_tracker.dataset.grid.**UnRegularGridDataset**(*filename*, *x_name*, *y_name*, *centered=None*, *indexs=None*, *unset=False*)

Bases: *py_eddy_tracker.dataset.grid.GridDataset*

Class managing unregular grid

Parameters

- **filename** (*str*) – Filename to load
- **x_name** (*str*) – Name of longitude coordinates
- **y_name** (*str*) – Name of latitude coordinates
- **centered** (*bool*,*None*) – Allow to know how coordinates could be used with pixel
- **indexs** (*dict*) – A dictionary which set indexs to use for non-coordinate dimensions
- **unset** (*bool*) – Set to True to create an empty grid object without file

Methods

add_grid	Add a grid in handler
bbox_indice	
c_to_bounds	Centred coordinates to bounds coordinates
compute_pixel_path	
copy	Duplicate the variable from grid_in in grid_out
eddy_identification	Compute eddy identification on the pecified grid
get_amplitude	
get_pixels_in	
get_uavg	Calculate geostrophic speed around successive contours Returns the average
grid	Give the grid required
grid_tiles	Give the grid tiles required, without buffer system
high_filter	Return the grid high-pass filtered, by substracting to the grid the low-pass filter (default: order=1)
init_pos_interpolator	
init_speed_coef	
is_circular	Check grid circularity
load	Load variable (data)
load_general_features	Load attrs to be stored in object
low_filter	Return the grid low-pass filtered (default: order=1)
nearest_grd_indice	
normalize_x_indice	Not do
setup_coordinates	
speed_coef_mean	
units	Get unit from variable
write	Write dataset output with same format as input

Attributes

EARTH_RADIUS	
GRAVITY	
N	
bounds	Give bound
centered	
contours	
coordinates	
dimensions	
filename	
global_attrs	
index_interp	
indexs	
interpolators	
is_centered	Give True if pixel is described with its center's position or a corner
speed_coef	
variables	
variables_description	
vars	
x_bounds	
x_c	
x_dim	
xinterp	
y_bounds	
y_c	
y_dim	
yinterp	

bbox_indice(*vertices*)

property bounds
Give bound

compute_pixel_path(*x0*, *y0*, *x1*, *y1*)

get_pixels_in(*contour*)

index_interp

init_pos_interpolator()

init_speed_coef(*uname='u'*, *vname='v'*)

load()
Load variable (data)

nearest_grd_indice(*x*, *y*)

normalize_x_indice(*indices*)
Not do

speed_coef_mean(*contour*)

13.3 py_eddy_tracker.featured_tracking

py_eddy_tracker.featured_tracking.area_tracker
py_eddy_tracker.featured_tracking.old_tracker_reference

13.3.1 py_eddy_tracker.featured_tracking.area_tracker

Classes

AreaTracker

py_eddy_tracker.featured_tracking.area_tracker.AreaTracker

class py_eddy_tracker.featured_tracking.area_tracker.**AreaTracker**(**args*, *cmin=0.2*, ***kwargs*)

Bases: *py_eddy_tracker.observations.observation.EddiesObservations*

Methods

add_fields	Add a new field.
add_rotation_type	
align_on	Align the time indexes of two datasets.
append	Merge.
basic_formula_ellips_major_axis	Give major axis in km with a given latitude
bins_stat	**param str,array xname** variable to compute stats on
box_display	Return value evenly spaced with few numbers
build_var_list	
circle_contour	Set contours as a circles from radius and center data.
coherence	Check coherence between two datasets.
compare_units	
concatenate	
copy	
copy_data_to_zarr	Copy with buffer for zarr.
cost_function	Return the cost function between two obs.
cost_function_common_area	How does it work on x bound ?
create_variable	
create_variable_zarr	
display	Plot the speed and effective (dashed) contour of the eddies
distance	Use haversine distance for distance matrix between every self and other eddies.

continues on next page

Table 22 – continued from previous page

extract_with_area	Extract geographically with a bounding box.
extract_with_mask	Extract a subset of observations.
filled	**param matplotlib.axes.Axes ax** matplotlib axe used to draw
first_obs	Get first obs of each trajectory.
fixed_ellipsoid_mask	
format_label	
from_netcdf	
from_zarr	
get_infos	
grid_box_stat	Compute mean of eddies in each bin
grid_count	Count the eddies in each bin (use all pixels in each contour)
grid_stat	Return the mean of the eddies' variable in each bin
hist	Build histograms.
index	Return obs from self at the index.
insert_observations	Insert other obs in self at the index.
inside	True for each point inside the effective contour of an eddy
intern	
interp_grid	Interpolate a grid on a center or contour with mean, min or max method
is_convex	Get flag of the eddy's convexity
iter_on	Yield observation group for each bin.
last_obs	Get Last obs of each trajectory.
load_file	Load the netcdf or the zarr file.
load_from_netcdf	Load data from netcdf.
load_from_zarr	Load data from zarr.
mask_function	
match	Return index and score computed on the effective contour.
merge	Merge two datasets.
merge_filters	Compute an intersection between all filters after to evaluate each of them
needed_variable	
netcdf_create_dimensions	
new_like	
obs_dimension	
post_process_link	
propagate	Filled virtual obs (C).
reset	
scatter	Scatter data.
set_global_attr_netcdf	
set_global_attr_zarr	
shifted_ellipsoid_degrees_mask	
solve_conflict	
solve_first	
solve_function	
solve_simultaneous	Write something (TODO)

continues on next page

Table 22 – continued from previous page

to_netcdf	
to_zarr	
tracking	Track obs between self and other
write_file	Write a netcdf or zarr with eddy obs.
zarr_dimension	

Attributes

ELEMENTS	
array_variables	
cmin	
dtype	Return dtype to build numpy array.
elements	Return all the names of the variables.
global_attr	
nb_days	Return period days cover by dataset
obs	Return observations.
observations	
only_variables	
period	Give the time coverage
period_	
raw_data	
shape	
sign_legend	
sign_type	
track_array_variables	
track_extra_variables	
tracks	

cmin

classmethod needed_variable()

propagate(*previous_obs*, *current_obs*, *obs_to_extend*, *dead_track*, *nb_next*, *model*)
Filled virtual obs (C).

Parameters

- **previous_obs** – previous obs from current (A)
- **current_obs** – previous obs from virtual (B)
- **obs_to_extend** –
- **dead_track** –
- **nb_next** –
- **model** –

Returns New position C = B + AB

tracking(*other*)
Track obs between self and other

13.3.2 py_eddy_tracker.featured_tracking.old_tracker_reference

Functions

check_ratio	Only very few case are remove with selection

py_eddy_tracker.featured_tracking.old_tracker_reference.check_ratio

py_eddy_tracker.featured_tracking.old_tracker_reference.**check_ratio**(*current_mask*, *self_amplitude*, *other_amplitude*, *self_radius*, *other_radius*)

Only very few case are remove with selection

Parameters

- **current_mask** –
- **self_amplitude** –
- **other_amplitude** –
- **self_radius** –
- **other_radius** –

Returns

Classes

CheltonTracker	

py_eddy_tracker.featured_tracking.old_tracker_reference.CheltonTracker

class py_eddy_tracker.featured_tracking.old_tracker_reference.**CheltonTracker**(*size=0*, *track_extra_variabl*, *track_array_variabl*, *array_variables=Non*, *only_variables=No*, *raw_data=False*)

Bases: *py_eddy_tracker.observations.observation.EddiesObservations*

Methods

across_ground	
add_fields	Add a new field.
add_rotation_type	
align_on	Align the time indexes of two datasets.
append	Merge.
basic_formula_ellips_major_axis	Give major axis in km with a given latitude
bins_stat	**param str,array xname** variable to compute stats on
box_display	Return value evenly spaced with few numbers
build_var_list	
circle_contour	Set contours as a circles from radius and center data.
coherence	Check coherence between two datasets.
compare_units	
concatenate	
copy	
copy_data_to_zarr	Copy with buffer for zarr.
cost_function	We minimize on distance between two obs
cost_function_common_area	How does it work on x bound ?
create_variable	
create_variable_zarr	
display	Plot the speed and effective (dashed) contour of the eddies
distance	Use haversine distance for distance matrix between every self and other eddies.
extract_with_area	Extract geographically with a bounding box.
extract_with_mask	Extract a subset of observations.
filled	**param matplotlib.axes.Axes ax** matplotlib axe used to draw
first_obs	Get first obs of each trajectory.
fixed_ellipsoid_mask	
format_label	
from_netcdf	
from_zarr	
get_infos	
grid_box_stat	Compute mean of eddies in each bin
grid_count	Count the eddies in each bin (use all pixels in each contour)
grid_stat	Return the mean of the eddies' variable in each bin
hist	Build histograms.
index	Return obs from self at the index.
insert_observations	Insert other obs in self at the index.
inside	True for each point inside the effective contour of an eddy
intern	

continues on next page

Table 26 – continued from previous page

interp_grid	Interpolate a grid on a center or contour with mean, min or max method
is_convex	Get flag of the eddy's convexity
iter_on	Yield observation group for each bin.
last_obs	Get Last obs of each trajectory.
load_file	Load the netcdf or the zarr file.
load_from_netcdf	Load data from netcdf.
load_from_zarr	Load data from zarr.
mask_function	We mask link with ellips and ratio
match	Return index and score computed on the effective contour.
merge	Merge two datasets.
merge_filters	Compute an intersection between all filters after to evaluate each of them
needed_variable	
netcdf_create_dimensions	
new_like	
obs_dimension	
post_process_link	When two self obs use the same other obs, we keep the self obs with amplitude max
propagate	Filled virtual obs (C).
reset	
scatter	Scatter data.
set_global_attr_netcdf	
set_global_attr_zarr	
shifted_ellipsoid_degrees_mask	
solve_conflict	
solve_first	
solve_function	Give the best link for each self obs
solve_simultaneous	Write something (TODO)
to_netcdf	
to_zarr	
tracking	Track obs between self and other
write_file	Write a netcdf or zarr with eddy obs.
zarr_dimension	

Attributes

ELEMENTS	
GROUND	
array_variables	
dtype	Return dtype to build numpy array.
elements	Return all the names of the variables.
global_attr	
nb_days	Return period days cover by dataset
obs	Return observations.
observations	
only_variables	
period	Give the time coverage

continues on next page

Table 27 – continued from previous page

period_
raw_data
shape
sign_legend
sign_type
track_array_variables
track_extra_variables
tracks

GROUND = <py_eddy_tracker.dataset.grid.RegularGridDataset object>

classmethod across_ground(*record0*, *record1*)

static cost_function(*records_in*, *records_out*, *distance*)
We minimize on distance between two obs

mask_function(*other*, *distance*)
We mask link with ellips and ratio

post_process_link(*other*, *i_self*, *i_other*)
When two self obs use the same other obs, we keep the self obs with amplitude max

solve_function(*cost_matrix*)
Give the best link for each self obs

13.4 py_eddy_tracker.observations.network

Class to create network of observations

Functions

get_next_index	Return for each obs index the new position to join all group

13.4.1 py_eddy_tracker.observations.network.get_next_index

py_eddy_tracker.observations.network.**get_next_index**(*gr*)
Return for each obs index the new position to join all group

Classes

Network

13.4.2 py_eddy_tracker.observations.network.Network

class py_eddy_tracker.observations.network.**Network**(*input_regex*, *window=5*, *intern=False*)

Bases: object

Methods

build_dataset	
get_group_array	With a loop on all pair of index, we will label each obs with a group number
group_observations	
load_contour	

Attributes

DATA
FLIST
NOGROUP
contour_name
filenames
nb_input
window
xname
yname

DATA = {}

FLIST = []

NOGROUP = 0

build_dataset(*group*)

contour_name

filenames

get_group_array(*results*, *nb_obs*)
With a loop on all pair of index, we will label each obs with a group number

group_observations(***kwargs*)

load_contour(*filename*)

nb_input

window

xname

yname

13.5 py_eddy_tracker.observations.observation

Base class to manage eddy observation

Functions

grid_box_stat	Compute method on each set (one set by box)
*grid_count_*	Add one to each index
grid_count_pixel_in	Count how many time a pixel is used.
grid_stat	Compute the mean or the max of the grid for each contour
insidepoly	True for each postion inside a contour
shifted_ellipsoid_degrees_mask2	Work only if major is an array but faster * 6

13.5.1 py_eddy_tracker.observations.observation.grid_box_stat

py_eddy_tracker.observations.observation.**grid_box_stat**(*x_c*, *y_c*, *grid*, *mask*, *x*, *y*, *value*, *circular=False*, *method=50*)

Compute method on each set (one set by box)

Parameters

- **x_c** (*array_like*) – grid longitude coordinates
- **y_c** (*array_like*) – grid latitude coordinates
- **grid** (*array_like*) – grid to store the result
- **mask** (*array[bool]*) – grid to store unused boxes
- **x** (*array_like*) – longitude of observations
- **y** (*array_like*) – latitude of observations
- **value** (*array_like*) – value to group to apply method
- **circular** (*bool*) – True if grid is wrappable
- **method** (*float*) – percentile

13.5.2 py_eddy_tracker.observations.observation.grid_count_

py_eddy_tracker.observations.observation.**grid_count_**(*grid*, *i*, *j*)

Add one to each index

13.5.3 py_eddy_tracker.observations.observation.grid_count_pixel_in

py_eddy_tracker.observations.observation.**grid_count_pixel_in**(*grid*, *x*, *y*, *x_ref*, *x_bounds*, *y_bounds*, *xstep*, *ystep*, *N*, *is_circular*, *x_size*, *x_c*, *y_c*)

Count how many time a pixel is used.

Parameters

- **grid** (*array*) –
- **x** (*array*) – x for all contour
- **y** (*array*) – y for all contour
- **x_ref** (*array*) – x reference for wrapping
- **x_bounds** (*array*) – grid longitude
- **y_bounds** (*array*) – grid latitude
- **xstep** (*float*) – step between two longitude
- **ystep** (*float*) – step between two latitude
- **N** (*int*) – shift of index to enlarge window
- **is_circular** (*bool*) – To know if grid is wrappable
- **x_size** (*int*) – Number of longitude
- **x_c** (*array*) – longitude coordinate of grid
- **y_c** (*array*) – latitude coordinate of grid

13.5.4 py_eddy_tracker.observations.observation.grid_stat

py_eddy_tracker.observations.observation.**grid_stat**(*x_c*, *y_c*, *grid*, *x*, *y*, *result*, *circular=False*, *method='mean'*)

Compute the mean or the max of the grid for each contour

Parameters

- **x_c** (*array_like*) – the grid longitude coordinates
- **y_c** (*array_like*) – the grid latitude coordinates
- **grid** (*array_like*) – grid value
- **x** (*array_like*) – longitude of contours
- **y** (*array_like*) – latitude of contours
- **result** (*array_like*) – return values
- **circular** (*bool*) – True if grid is wrappable
- **method** (*str*) – ‘mean’, ‘max’

13.5.5 py_eddy_tracker.observations.observation.insidepoly

py_eddy_tracker.observations.observation.**insidepoly**(*x_p*, *y_p*, *x_c*, *y_c*)

True for each postion inside a contour

Parameters

- **x_p** (*array*) – longitude to test
- **y_p** (*array*) – latitude to test
- **x_c** (*array*) – longitude of contours
- **y_c** (*array*) – latitude of contours

13.5.6 py_eddy_tracker.observations.observation.shifted_ellipsoid_degrees_mask2

py_eddy_tracker.observations.observation.**shifted_ellipsoid_degrees_mask2**(*lon0*, *lat0*, *lon1*, *lat1*, *minor=1.5*, *major=1.5*)

Work only if major is an array but faster * 6

Classes

EddiesObservations	Class to store eddy observations.
VirtualEddiesObservations	Class to work with virtual obs

13.5.7 py_eddy_tracker.observations.observation.EddiesObservations

class py_eddy_tracker.observations.observation.**EddiesObservations**(*size=0*, *track_extra_variables=None*, *track_array_variables=0*, *array_variables=None*, *only_variables=None*, *raw_data=False*)

Bases: object

Class to store eddy observations.

Methods

add_fields	Add a new field.
add_rotation_type	
align_on	Align the time indexes of two datasets.
append	Merge.
basic_formula_ellips_major_axis	Give major axis in km with a given latitude
bins_stat	**param str,array xname** variable to compute stats on
box_display	Return value evenly spaced with few numbers
build_var_list	
circle_contour	Set contours as a circles from radius and center data.
coherence	Check coherence between two datasets.
compare_units	
concatenate	
copy	
copy_data_to_zarr	Copy with buffer for zarr.
cost_function	Return the cost function between two obs.
cost_function_common_area	How does it work on x bound ?
create_variable	
create_variable_zarr	
display	Plot the speed and effective (dashed) contour of the eddies
distance	Use haversine distance for distance matrix between every self and other eddies.
extract_with_area	Extract geographically with a bounding box.
extract_with_mask	Extract a subset of observations.
filled	**param matplotlib.axes.Axes ax** matplotlib axe used to draw
first_obs	Get first obs of each trajectory.
fixed_ellipsoid_mask	
format_label	
from_netcdf	
from_zarr	
get_infos	
grid_box_stat	Compute mean of eddies in each bin
grid_count	Count the eddies in each bin (use all pixels in each contour)
grid_stat	Return the mean of the eddies' variable in each bin
hist	Build histograms.
index	Return obs from self at the index.
insert_observations	Insert other obs in self at the index.
inside	True for each point inside the effective contour of an eddy
intern	
interp_grid	Interpolate a grid on a center or contour with mean, min or max method

continues on next page

Table 34 – continued from previous page

is_convex	Get flag of the eddy's convexity
iter_on	Yield observation group for each bin.
last_obs	Get Last obs of each trajectory.
load_file	Load the netcdf or the zarr file.
load_from_netcdf	Load data from netcdf.
load_from_zarr	Load data from zarr.
mask_function	
match	Return index and score computed on the effective contour.
merge	Merge two datasets.
merge_filters	Compute an intersection between all filters after to evaluate each of them
needed_variable	
netcdf_create_dimensions	
new_like	
obs_dimension	
post_process_link	
propagate	Filled virtual obs (C).
reset	
scatter	Scatter data.
set_global_attr_netcdf	
set_global_attr_zarr	
shifted_ellipsoid_degrees_mask	
solve_conflict	
solve_first	
solve_function	
solve_simultaneous	Write something (TODO)
to_netcdf	
to_zarr	
tracking	Track obs between self and other
write_file	Write a netcdf or zarr with eddy obs.
zarr_dimension	

Attributes

ELEMENTS	
array_variables	
dtype	Return dtype to build numpy array.
elements	Return all the names of the variables.
global_attr	
nb_days	Return period days cover by dataset
obs	Return observations.
observations	
only_variables	
period	Give the time coverage
*period_*	
raw_data	
shape	
sign_legend	

continues on next page

Table 35 – continued from previous page

sign_type
track_array_variables
track_extra_variables
tracks

ELEMENTS = ['lon', 'lat', 'radius_s', 'radius_e', 'amplitude', 'speed_average', 'time'

add_fields (*fields=[]*, *array_fields=[]*)
Add a new field.

add_rotation_type ()

align_on (*other*, *var_name='time'*, ***kwargs*)
Align the time indexes of two datasets.

append (*other*)
Merge.

array_variables

static basic_formula_ellips_major_axis (*lats*, *cmin=1.5*, *cmax=10.0*, *c0=1.5*, *lat1=13.5*, *lat2=5.0*, *degrees=False*)
Give major axis in km with a given latitude

bins_stat (*xname*, *bins=None*, *yname=None*, *method=None*, *mask=None*)

Parameters

- **xname** (*str*, *array*) – variable to compute stats on
- **None bins** (*array*,) – bins to perform statistics, if None bins = arange(variable.min(), variable.max() + 2)
- **yname** (*None*, *str*, *array*) – variable used to apply method
- **method** (*None*, *str*) – If None method counts the number of observations in each bin, can be "mean", "std"
- **mask** (*None*, *array (bool)*) – If defined use only True position

Returns x array and y array

Return type array,array

- *Get Okubo Weis*

static box_display (*value*)
Return value evenly spaced with few numbers

static build_var_list (*var_list*, *remove_vars*, *include_vars*)

circle_contour (*only_virtual=False*)
Set contours as a circles from radius and center data.

- *Display contour & circle*

coherence (*other*)
Check coherence between two datasets.

static compare_units (*input_unit*, *output_unit*, *name*)

classmethod concatenate (*observations*)

copy()

static copy_data_to_zarr(*handler_zarr*, *handler_eddies*, *sl_obs*, *buffer_size*, *factor*, *raw_data*, *scale_factor*, *add_offset*)

Copy with buffer for zarr.

Zarr need to get real value, and size could be huge, so we use a buffer to manage memory :param zarr_dataset handler_zarr: :param array handler_eddies: :param slice zarr_dataset sl_obs: :param int zarr_dataset buffer_size: :param float zarr_dataset factor: :param bool zarr_dataset raw_data: :param None,float zarr_dataset scale_factor: :param None,float add_offset:

static cost_function(*records_in*, *records_out*, *distance*)

Return the cost function between two obs.

$$cost = \sqrt{(\frac{Amp_{in} - Amp_{out}}{Amp_{in}})^2 + (\frac{Rspeed_{in} - Rspeed_{out}}{Rspeed_{in}})^2 + (\frac{distance}{125})^2}$$

Parameters

- **records_in** – starting observations
- **records_out** – observations to associate
- **distance** – computed between in and out

classmethod cost_function_common_area(*xy_in*, *xy_out*, *distance*, *intern=False*)

How does it work on x bound ?

Parameters

- **xy_in** –
- **xy_out** –
- **distance** –
- **intern** (*bool*) –

create_variable(*handler_nc*, *kwargs_variable*, *attr_variable*, *data*, *scale_factor=None*, *add_offset=None*, ***kwargs*)

create_variable_zarr(*handler_zarr*, *kwargs_variable*, *attr_variable*, *data*, *scale_factor=None*, *add_offset=None*, *filters=None*, *compressor=None*, *chunck_size=2500000*)

display(*ax*, *ref=None*, *extern_only=False*, *intern_only=False*, ***kwargs*)

Plot the speed and effective (dashed) contour of the eddies

Parameters

- **ax** (*matplotlib.axes.Axes*) – matplotlib axe used to draw
- **ref** (*float*, *None*) – western longitude reference used
- **extern_only** (*bool*) – if True, draw only the effective contour
- **intern_only** (*bool*) – if True, draw only the speed contour
- **kwargs** (*dict*) – look at `matplotlib.axes.Axes.plot()`

- *Display contour & circle*
- *Display identification*
- *Get mean of grid in each eddies*

- *Eddy detection : Med*
- *Eddy detection : Gulf stream*
- *Eddy detection and filter*
- *Eddy detection on SLA and ADT*
- *Select pixel in eddies*
- *Get Okubo Weis*
- *Collocating external data*

distance(*other*)
Use haversine distance for distance matrix between every self and other eddies.

property dtype
Return dtype to build numpy array.

property elements
Return all the names of the variables.

extract_with_area(*area*, ***kwargs*)
Extract geographically with a bounding box.

Parameters

- **area** (*dict*) – 4 coordinates in a dictionary to specify bounding box (lower left corner and upper right corner)
- **kwargs** (*dict*) – look at *extract_with_mask()*

Returns Return all eddy tracks which are in bounds

Return type *EddiesObservations*

```
area = dict(llcrnrlon=x0, llcrnrlat=y0, urcrnrlon=x1, urcrnrlat=y1)
```

- *Tracks which go through area*

extract_with_mask(*mask*)
Extract a subset of observations.

Parameters **mask** (*array(bool)*) – mask to select observations

Returns same object with selected observations

Return type self

filled(*ax*, *varname=None*, *ref=None*, *intern=False*, *cmap='magma_r'*, *lut=10*, *vmin=None*, *vmax=None*, *factor=1*, ***kwargs*)

Parameters

- **ax** (*matplotlib.axes.Axes*) – matplotlib axe used to draw
- **varname** (*str,array,None*) – variable used to fill the contours, or an array of same size than obs
- **ref** (*float,None*) – if define use like west bound?

- **intern** (*bool*) – if True draw speed contours instead of effective contours
- **cmap** (*str*) – matplotlib colormap name
- **lut** (*int,None*) – Number of colors in the colormap
- **vmin** (*float,None*) – Min value of the colorbar
- **vmax** (*float,None*) – Max value of the colorbar
- **factor** (*float*) – multiply value by

Returns Collection drawed

Return type matplotlib.collections.PolyCollection

- *Display identification*
- *Get mean of grid in each eddies*
- *Eddy detection : Med*
- *Eddy detection : Gulf stream*

first_obs()
Get first obs of each trajectory.

Return type __class__

- *Birth and death*

fixed_ellipsoid_mask(*other*, *minor=50*, *major=100*, *only_east=False*, *shifted_ellips=False*)

format_label(*label*)

classmethod from_netcdf(*handler*)

classmethod from_zarr(*handler*)

get_infos()

property global_attr

grid_box_stat(*bins*, *varname*, *method=50*, *data=None*, *filter=slice(None, None, None)*)
Compute mean of eddies in each bin

Parameters

- **bins** (*(numpy.array,numpy.array)*) – bins (grid) to count
- **varname** (*str*) – variable to apply the method
- **method** (*str,float*) – method to apply. If float, use ?
- **data** (*array*) – Array used to compute stat if defined
- **filter** (*array,mask,slice*) – keep the data selected with the filter

Returns return grid of method

Return type *py_eddy_tracker.dataset.grid.RegularGridDataset*

grid_count(*bins*, *intern=False*, *center=False*, *filter=slice(None, None, None)*)
Count the eddies in each bin (use all pixels in each contour)

Parameters

- **bins** (*(numpy.array,numpy.array)*) – bins (grid) to count
- **intern** (*bool*) – if True use speed contour only
- **center** (*bool*) – if True use of center to count
- **filter** (*array,mask,slice*) – keep the data selected with the filter

Returns return the grid of counts

Return type *py_eddy_tracker.dataset.grid.RegularGridDataset*

- *Count pixel used*
- *Count center*

grid_stat (*bins*, *varname*, *data=None*)
Return the mean of the eddies' variable in each bin

Parameters

- **bins** (*(numpy.array,numpy.array)*) – bins (grid) to compute the mean on
- **varname** (*str*) – name of variable to compute the mean on
- **data** (*array*) – Array used to compute stat if defined

Returns return the gridde mean variable

Return type *py_eddy_tracker.dataset.grid.RegularGridDataset*

- *Geographical statistics*

hist (*varname*, *x*, *bins*, *percent=False*, *mean=False*, *nb=False*)
Build histograms.

Parameters

- **varname** (*str*) – variable to use to compute stat
- **x** (*str*) – variable to use to know in which bins
- **bins** (*array*) –
- **percent** (*bool*) – normalize by sum of all bins
- **mean** (*bool*) – compute mean by bins
- **nb** (*bool*) – only count by bins

Returns value by bins

Return type array

index (*index*, *reverse=False*)
Return obs from self at the index.

insert_observations (*other*, *index*)
Insert other obs in self at the index.

inside (*x*, *y*, *intern=False*)
True for each point inside the effective contour of an eddy

Parameters

- **x** (*array*) – longitude

- **y** (*array*) – latitude
- **intern** (*bool*) – If true use speed contour instead of effective contour

Returns flag

Return type array[bool]

static intern(*flag*, *public_label=False*)

interp_grid(*grid_object*, *varname*, *method='center'*, *dtype=None*, *intern=None*)
Interpolate a grid on a center or contour with mean, min or max method

Parameters

- **grid_object** (py_eddy_tracker.dataset.grid.RegularGridDataset) – Handler of grid to interp
- **varname** (*str*) – Name of variable to use
- **method** (*str*) – 'center', 'mean', 'max', 'min', 'nearest'
- **dtype** (*str*) – if None we use var dtype
- **intern** (*bool*) – Use extern or intern contour

is_convex(*intern=False*)
Get flag of the eddy's convexity

Parameters **intern** (*bool*) – If True use speed contour instead of effective contour

Returns True if the contour is convex

Return type array[bool]

iter_on(*xname: str*, *bins=None*)
Yield observation group for each bin.

Parameters

- **xname** (*str*) –
- **bins** (*array*) – bounds of each bin ,

Returns Group observations

Return type self.__class__

last_obs()
Get Last obs of each trajectory.

Return type __class__

- *Birth and death*

classmethod load_file(*filename*, ***kwargs*)
Load the netcdf or the zarr file.

Load only latitude and longitude on the first 300 obs :

```
kwargs_latlon_300 = dict(
    include_vars=[
        "longitude",
        "latitude",
    ],
    indexs=dict(obs=slice(0, 300)),
```

(continues on next page)

(continued from previous page)

```
)
small_dataset = TrackEddiesObservations.load_file(
    filename, **kwargs_latlon_300
)
```

For ***kwargs* look at *load_from_zarr()* or *load_from_netcdf()*

classmethod load_from_netcdf(*filename*, *raw_data=False*, *remove_vars=None*, *include_vars=None*, *indexs=None*, ***class_kwargs*)

Load data from netcdf.

Parameters

- **filename** (*str*, *ExFileObject*) – path or handler to load data
- **raw_data** (*bool*) – If true load data without apply scale_factor and add_offset
- **remove_vars** (*None*, *list(str)*) – List of variable name which will be not loaded
- **include_vars** (*None*, *list(str)*) – If defined only this variable will be loaded
- **indexs** (*None*, *dict*) – Indexs to laad only a slice of data
- **class_kwargs** – argument to set up observations class

Returns Obsevations selected

Return type class

classmethod load_from_zarr(*filename*, *raw_data=False*, *remove_vars=None*, *include_vars=None*, *indexs=None*, *buffer_size=5000000*, ***class_kwargs*)

Load data from zarr.

Parameters

- **filename** (*str*, *store*) – path or store to load data
- **raw_data** (*bool*) – If true load data without apply scale_factor and add_offset
- **remove_vars** (*None*, *list(str)*) – List of variable name which will be not loaded
- **include_vars** (*None*, *list(str)*) – If defined only this variable will be loaded
- **indexs** (*None*, *dict*) – Indexs to laad only a slice of data
- **buffer_size** (*int*) – Size of buffer used to load zarr data
- **class_kwargs** – argument to set up observations class

Returns Obsevations selected

Return type class

mask_function(*other*, *distance*)

match(*other*, *method='overlap'*, *intern=False*, *cmin=0*, ***kwargs*)

Return index and score computed on the effective contour.

Parameters

- **other** (EddiesObservations) – Observations to compare
- **method** (*str*) –
  - "overlap": the score is computed with contours;

– "circle": circles are computed and used for score (TODO)

- **intern** (*bool*) – if True, speed contour is used (default = effective contour)
- **cmin** (*float*) – 0 < cmin < 1, return only couples with score >= cmin
- **kwargs** (*dict*) – look at `vertice_overlap()`

Returns return the indexes of the eddies in self coupled with eddies in other and their associated score

Return type (array(int), array(int), array(float))

- *Eddy detection and filter*
- *Eddy detection on SLA and ADT*

merge(*other*)
Merge two datasets.

merge_filters(**filters*)
Compute an intersection between all filters after to evaluate each of them

Parameters **filters** (*list(callable,None,slice,array[int],array[bool])*) –

Returns Return applicable object to numpy.array

Return type slice, index, mask

property nb_days
Return period days cover by dataset

Returns Number of days

Return type int

classmethod needed_variable()

static netcdf_create_dimensions(*handler*, *dim*, *nb*)

static new_like(*eddies*, *new_size: int*)

property obs
Return observations.

classmethod obs_dimension(*handler*)

observations

only_variables

property period
Give the time coverage

Returns first and last date

Return type (int,int)

period_

post_process_link(*other*, *i_self*, *i_other*)

propagate(*previous_obs*, *current_obs*, *obs_to_extend*, *dead_track*, *nb_next*, *model*)
Filled virtual obs (C).

Parameters

- **previous_obs** – previous obs from current (A)
- **current_obs** – previous obs from virtual (B)
- **obs_to_extend** –
- **dead_track** –
- **nb_next** –
- **model** –

Returns New position C = B + AB

raw_data

reset()

scatter(*ax*, *name=None*, *ref=None*, *factor=1*, ***kwargs*)
Scatter data.

Parameters

- **ax** (*matplotlib.axes.Axes*) – matplotlib axe used to draw
- **name** (*str,array,None*) – variable used to fill the contour, if None all elements have the same color
- **ref** (*float,None*) – if define use like west bound
- **factor** (*float*) – multiply value by
- **kwargs** (*dict*) – look at `matplotlib.axes.Axes.scatter()`

Returns scatter mappable

- *Eddy detection : Med*
- *Display fields*
- *One Track*

set_global_attr_netcdf(*h_nc*)

set_global_attr_zarr(*h_zarr*)

property shape

shifted_ellipsoid_degrees_mask(*other*, *minor=1.5*, *major=1.5*)

property sign_legend

sign_type

static solve_conflict(*cost*)

static solve_first(*cost*, *multiple_link=False*)

solve_function(*cost_matrix*)

static solve_simultaneous(*cost*)
Write something (TODO)

to_netcdf(*handler*, ***kwargs*)

to_zarr(*handler*, ***kwargs*)

track_array_variables

track_extra_variables

tracking(*other*)
Track obs between self and other

property tracks

write_file(*path='./'*, *filename='%(path)s/%(sign_type)s.nc'*, *zarr_flag=False*, ***kwargs*)
Write a netcdf or zarr with eddy obs. Zarr is usefull for large dataset > 10M observations

Parameters

- **path** (*str*) – set path variable
- **filename** (*str*) – model to store file
- **zarr_flag** (*bool*) – If True, method will use zarr format instead of netcdf
- **kwargs** (*dict*) – look at *to_zarr()* or *to_netcdf()*

static zarr_dimension(*filename*)

13.5.8 py_eddy_tracker.observations.observation.VirtualEddiesObservations

class py_eddy_tracker.observations.observation.**VirtualEddiesObservations**(*size=0*, *track_extra_variables=No*, *track_array_variables=0*, *array_variables=None*, *only_variables=None*, *raw_data=False*)

Bases: *py_eddy_tracker.observations.observation.EddiesObservations*

Class to work with virtual obs

Methods

add_fields	Add a new field.
add_rotation_type	
align_on	Align the time indexes of two datasets.
append	Merge.
basic_formula_ellips_major_axis	Give major axis in km with a given latitude
bins_stat	**param str,array xname** variable to compute stats on
box_display	Return value evenly spaced with few numbers
build_var_list	
circle_contour	Set contours as a circles from radius and center data.
coherence	Check coherence between two datasets.
compare_units	
concatenate	
copy	
copy_data_to_zarr	Copy with buffer for zarr.
cost_function	Return the cost function between two obs.

continues on next page

Table 36 – continued from previous page

cost_function_common_area	How does it work on x bound ?
create_variable	
create_variable_zarr	
display	Plot the speed and effective (dashed) contour of the eddies
distance	Use haversine distance for distance matrix between every self and other eddies.
extract_with_area	Extract geographically with a bounding box.
extract_with_mask	Extract a subset of observations.
filled	**param matplotlib.axes.Axes ax** matplotlib axe used to draw
first_obs	Get first obs of each trajectory.
fixed_ellipsoid_mask	
format_label	
from_netcdf	
from_zarr	
get_infos	
grid_box_stat	Compute mean of eddies in each bin
grid_count	Count the eddies in each bin (use all pixels in each contour)
grid_stat	Return the mean of the eddies' variable in each bin
hist	Build histograms.
index	Return obs from self at the index.
insert_observations	Insert other obs in self at the index.
inside	True for each point inside the effective contour of an eddy
intern	
interp_grid	Interpolate a grid on a center or contour with mean, min or max method
is_convex	Get flag of the eddy's convexity
iter_on	Yield observation group for each bin.
last_obs	Get Last obs of each trajectory.
load_file	Load the netcdf or the zarr file.
load_from_netcdf	Load data from netcdf.
load_from_zarr	Load data from zarr.
mask_function	
match	Return index and score computed on the effective contour.
merge	Merge two datasets.
merge_filters	Compute an intersection between all filters after to evaluate each of them
needed_variable	
netcdf_create_dimensions	
new_like	
obs_dimension	
post_process_link	
propagate	Filled virtual obs (C).
reset	
scatter	Scatter data.

continues on next page

Table 36 – continued from previous page

set_global_attr_netcdf	
set_global_attr_zarr	
shifted_ellipsoid_degrees_mask	
solve_conflict	
solve_first	
solve_function	
solve_simultaneous	Write something (TODO)
to_netcdf	
to_zarr	
tracking	Track obs between self and other
write_file	Write a netcdf or zarr with eddy obs.
zarr_dimension	

Attributes

ELEMENTS	
array_variables	
dtype	Return dtype to build numpy array.
elements	Return all the names of the variables.
global_attr	
nb_days	Return period days cover by dataset
obs	Return observations.
observations	
only_variables	
period	Give the time coverage
period_	
raw_data	
shape	
sign_legend	
sign_type	
track_array_variables	
track_extra_variables	
tracks	

property elements
Return all the names of the variables.

13.6 py_eddy_tracker.observations.tracking

Class to manage observations gathered in track

Functions

compute_index	
compute_mask_from_id	
count_by_track	
track_loess_filter	Apply a loess filter on y field
track_median_filter	Apply a median filter on y field

13.6.1 py_eddy_tracker.observations.tracking.compute_index

py_eddy_tracker.observations.tracking.**compute_index**(*tracks*, *index*, *number*)

13.6.2 py_eddy_tracker.observations.tracking.compute_mask_from_id

py_eddy_tracker.observations.tracking.**compute_mask_from_id**(*tracks*, *first_index*, *number_of_obs*, *mask*)

13.6.3 py_eddy_tracker.observations.tracking.count_by_track

py_eddy_tracker.observations.tracking.**count_by_track**(*tracks*, *mask*, *number*)

13.6.4 py_eddy_tracker.observations.tracking.track_loess_filter

py_eddy_tracker.observations.tracking.**track_loess_filter**(*half_window*, *x*, *y*, *track*)

Apply a loess filter on y field

Parameters

- **window** (*int*, *float*) – parameter of smoother
- **x** (*array_like*) – must be growing for each track but could be irregular
- **y** (*array_like*) – field to smooth
- **track** (*array_like*) – field which allow to separate path

Returns Array smoothed

Return type array_like

13.6.5 py_eddy_tracker.observations.tracking.track_median_filter

py_eddy_tracker.observations.tracking.**track_median_filter**(*half_window*, *x*, *y*, *track*)

Apply a median filter on y field

Parameters

- **half_window** (*int, float*) – parameter of smoother
- **x** (*array_like*) – must be growing for each track but could be irregular
- **y** (*array_like*) – field to smooth
- **track** (*array_like*) – field which allow to separate path

Returns Array smoothed

Return type array_like

Classes

TrackEddiesObservations	Class to practice Tracking on observations

13.6.6 py_eddy_tracker.observations.tracking.TrackEddiesObservations

class py_eddy_tracker.observations.tracking.**TrackEddiesObservations**(**args*, ***kwargs*)

Bases: *py_eddy_tracker.observations.observation.EddiesObservations*

Class to practice Tracking on observations

Methods

add_distance	Add a field of distance (m) between to consecutive observation, 0 for the last observation of each track
add_fields	Add a new field.
add_rotation_type	
align_on	Align the time indexes of two datasets.
append	Merge.
basic_formula_ellips_major_axis	Give major axis in km with a given latitude
bins_stat	**param str,array xname** variable to compute stats on
box_display	Return value evenly spaced with few numbers
build_var_list	
circle_contour	Set contours as a circles from radius and center data.
close_tracks	Get close from another atlas.
coherence	Check coherence between two datasets.
compare_units	
compute_index	If obs are not sorted by track, __first_index_of_track will be unusable

continues on next page

Table 40 – continued from previous page

concatenate	
copy	
copy_data_to_zarr	Copy with buffer for zarr.
cost_function	Return the cost function between two obs.
cost_function_common_area	How does it work on x bound ?
count_by_track	Count by track
create_variable	
create_variable_zarr	
display	Plot the speed and effective (dashed) contour of the eddies
display_shape	This function will draw shape of each track
distance	Use haversine distance for distance matrix between every self and other eddies.
distance_to_next	**return** array of distance in m, 0 when next obs if from another track
empty_dataset	
extract_first_obs_in_box	
extract_ids	
extract_in_direction	
extract_longer_eddies	Select eddies which are longer than nb_min
extract_toward_direction	Get eddy which go in same direction
extract_with_area	Extract geographically with a bounding box.
extract_with_length	Return all observations in [b0:b1]
extract_with_mask	Extract a subset of observations
extract_with_period	Extract with a period
filled	**param matplotlib.axes.Axes ax** matplotlib axe used to draw
filled_by_interpolation	Filled selected values by interpolation
first_obs	Get first obs of each trajectory.
fixed_ellipsoid_mask	
follow_obs	
format_label	
from_netcdf	
from_zarr	
get_azimuth	Return azimuth for each tracks.
get_infos	
get_mask_from_id	
grid_box_stat	Compute mean of eddies in each bin
grid_count	Count the eddies in each bin (use all pixels in each contour)
grid_stat	Return the mean of the eddies' variable in each bin
hist	Build histograms.
index	Return obs from self at the index.
insert_observations	Insert other obs in self at the index.
inside	True for each point inside the effective contour of an eddy
intern	

continues on next page

Table 40 – continued from previous page

interp_grid	Interpolate a grid on a center or contour with mean, min or max method
is_convex	Get flag of the eddy's convexity
iter_on	Yield observation group for each bin.
iter_track	Yield track
last_obs	Get Last obs of each trajectory.
load_file	Load the netcdf or the zarr file.
load_from_netcdf	Load data from netcdf.
load_from_zarr	Load data from zarr.
loess_filter	
mask_function	
match	Return index and score computed on the effective contour.
median_filter	
merge	Merge two datasets.
merge_filters	Compute an intersection between all filters after to evaluate each of them
needed_variable	
netcdf_create_dimensions	
new_like	
next_obs	
obs_dimension	
plot	This function will draw path of each track
position_filter	
post_process_link	
propagate	Filled virtual obs (C).
re_reference_index	
reset	
scatter	Scatter data.
set_global_attr_netcdf	Set global attr
set_global_attr_zarr	
set_tracks	Will split one group in tracks
shape_polygon	Get polygons which enclosed each track
shifted_ellipsoid_degrees_mask	
solve_conflict	
solve_first	
solve_function	
solve_simultaneous	Write something (TODO)
split_network	Divide each group in track
to_netcdf	
to_zarr	
tracking	Track obs between self and other
write_file	Write a netcdf or zarr with eddy obs.
zarr_dimension	

Attributes

ELEMENTS	
NOGROUP	
age	Return for each observation age in %, will be [0:100]
array_variables	
dtype	Return dtype to build numpy array.
elements	Return all the names of the variables.
global_attr	
index_from_track	
lifetime	Return for each observation lifetime
nb_days	Return period days cover by dataset
nb_obs_by_track	
nb_tracks	Will count and send number of track
obs	Return observations.
observations	
only_variables	
period	Give the time coverage
period_	
raw_data	
shape	
sign_legend	
sign_type	
track_array_variables	
track_extra_variables	
tracks	

ELEMENTS = ['lon', 'lat', 'radius_s', 'radius_e', 'speed_area', 'effective_area', 'amp

NOGROUP = 0

add_distance()
Add a field of distance (m) between to consecutive observation, 0 for the last observation of each track

property age
Return for each observation age in %, will be [0:100]

close_tracks(*other*, *nb_obs_min=10*, ***kwargs*)
Get close from another atlas.

Parameters

- **other** (*self*) – Atlas to compare
- **nb_obs_min** (*int*) – Minimal number of overlap for one track
- **kwargs** (*dict*) – keyword arguments for match function

Returns return other atlas reduce to common track with self

> **Warning:** It could be a costly operation for huge dataset

compute_index()
If obs are not sorted by track, __first_index_of_track will be unusable

classmethod concatenate(*observations*)

count_by_track(*mask*)
Count by track

Parameters **mask** (*array[bool]*) – Mask of boolean count +1 if true

Returns Return count by track

Return type array

display_shape(*ax*, *ref=None*, *intern=False*, ***kwargs*)
This function will draw shape of each track

Parameters

- **ax** (*matplotlib.axes.Axes*) – ax where drawed
- **ref** (*float, int*) – if defined all coordinates will be wrapped with ref like west boundary
- **intern** (*bool*) – If True use speed contour instead of effective contour
- **kwargs** (*dict*) – keyword arguments for Axes.plot

Returns matplotlib mappable

distance_to_next()

Returns array of distance in m, 0 when next obs if from another track

Return type array

property elements
Return all the names of the variables.

empty_dataset()

extract_first_obs_in_box(*res*)

extract_ids(*tracks*)

extract_in_direction(*direction*, *value=0*)

extract_longer_eddies(*nb_min*, *nb_obs*, *compress_id=True*)
Select eddies which are longer than nb_min

extract_toward_direction(*west=True*, *delta_lon=None*)
Get eddy which go in same direction

Parameters

- **west** (*bool*) – Only eastward eddy if True return westward
- **delta_lon** (*None, float*) – Only eddy with more than delta_lon span in longitude

Returns Only eastern eddy

Return type __class__

extract_with_length(*bounds*)
Return all observations in [b0:b1]

Parameters **bounds** (*(int, int)*) – length min and max of selected eddies, if use of -1 this bound is not used

Returns Return all eddy tracks which have length between bounds

Return type *TrackEddiesObservations*

- *Display fields*
- *Display Tracks*

extract_with_mask(*mask*, *full_path=False*, *remove_incomplete=False*, *compress_id=False*, *reject_virtual=False*)
Extract a subset of observations

Parameters

- **mask** (*array (bool)*) – mask to select observations
- **full_path** (*bool*) – extract full path if only one part is selected
- **remove_incomplete** (*bool*) – delete path which are not fully selected
- **compress_id** (*bool*) – resample track number to use a little range
- **reject_virtual** (*bool*) – if track are only virtual in selection we remove track

Returns same object with selected observations

Return type self.__class__

extract_with_period(*period*, ***kwargs*)
Extract with a period

Parameters

- **period** (*(int, int)*) – two date to define period, must be specify from 1/1/1950
- **kwargs** (*dict*) – look at *extract_with_mask()*

Returns Return all eddy tracks which are in bounds

Return type *TrackEddiesObservations*

filled_by_interpolation(*mask*)
Filled selected values by interpolation

Parameters **mask** (*array (bool)*) – True if must be filled by interpolation

- *Track in python*

classmethod follow_obs(*i_next*, *track_id*, *used*, *ids*, **args*)

format_label(*label*)

get_azimuth(*equatorward=False*)
Return azimuth for each tracks.

Azimuth is compute with first and last observation

Parameters **equatorward** (*bool*) – If True, Poleward are positive and equatorward negative

Return type array

get_mask_from_id(*tracks*)

property index_from_track

iter_track()
Yield track

property lifetime
Return for each observation lifetime

loess_filter(*half_window*, *xfield*, *yfield*, *inplace=True*)

median_filter(*half_window*, *xfield*, *yfield*, *inplace=True*)

property nb_obs_by_track

property nb_tracks
Will count and send number of track

static next_obs(*i_current*, *ids*, *polygons*, *time_s*, *time_e*, *time_ref*, *window*)

plot(*ax*, *ref=None*, ***kwargs*)
This function will draw path of each track

Parameters

- **ax** (*matplotlib.axes.Axes*) – ax where drawed
- **ref** (*float*, *int*) – if defined all coordinates will be wrapped with ref like west boundary
- **kwargs** (*dict*) – keyword arguments for Axes.plot

Returns matplotlib mappable

position_filter(*median_half_window*, *loess_half_window*)

static re_reference_index(*index*, *ref*)

set_global_attr_netcdf(*h_nc*)
Set global attr

set_tracks(*x*, *y*, *ids*, *window*)
Will split one group in tracks

Parameters

- **x** (*array*) – coordinates of group
- **y** (*array*) – coordinates of group
- **ids** (*ndarray*) – several fields like time, group, ...
- **windows** (*int*) – number of days where observations could missed

shape_polygon(*intern=False*)
Get polygons which enclosed each track

Parameters **intern** (*bool*) – If True use speed contour instead of effective contour

Return type list(array, array)

split_network(*intern=True*, ***kwargs*)
Divide each group in track

13.7 py_eddy_tracker.eddy_feature

Class to compute Amplitude and average speed profile

Functions

*detect_local_minima_*	Take an array and detect the troughs using the local maximum filter.
index_from_nearest_path_with_pt_in_bbox	Get index from nearest path in edge bbox contain pt

13.7.1 py_eddy_tracker.eddy_feature.detect_local_minima_

py_eddy_tracker.eddy_feature.**detect_local_minima_**(*grid*, *general_mask*, *pixel_mask*, *maximum_local_extremum*, *sign*)

Take an array and detect the troughs using the local maximum filter. Returns a boolean mask of the troughs (i.e., 1 when the pixel's value is the neighborhood maximum, 0 otherwise) http://stackoverflow.com/questions/3684484/peak-detection-in-a-2d-array/3689710#3689710

13.7.2 py_eddy_tracker.eddy_feature.index_from_nearest_path_with_pt_in_bbox_

py_eddy_tracker.eddy_feature.**index_from_nearest_path_with_pt_in_bbox_**(*level_index*, *l_i*, *nb_c_per_l*, *nb_pt_per_c*, *indices_of_first_pts*, *x_value*, *y_value*, *x_min_per_c*, *y_min_per_c*, *x_max_per_c*, *y_max_per_c*, *xpt*, *ypt*)

Get index from nearest path in edge bbox contain pt

Classes

Amplitude	Class to calculate *amplitude* and counts of *local maxima/minima* within a closed region of a sea level anomaly field.
Contours	Class to calculate average geostrophic velocity along a contour, *uavg*, and return index to contour with maximum *uavg* within a series of closed contours.

13.7.3 py_eddy_tracker.eddy_feature.Amplitude

class py_eddy_tracker.eddy_feature.**Amplitude**(*contour*, *contour_height*, *data*, *interval*, *mle=1*, *nb_step_min=2*, *nb_step_to_be_mle=2*)

Bases: object

Class to calculate *amplitude* and counts of *local maxima/minima* within a closed region of a sea level anomaly field.

Create amplitude object

Parameters

- **contour** (Contours) –
- **contour_height** (*float*) –
- **data** (*array*) –
- **interval** (*float*) –
- **mle** (*int*) – maximum number of local maxima in contour
- **nb_step_min** (*int*) – number of interval to consider like an eddy
- **nb_step_to_be_mle** (*int*) – number of interval to be consider like another maxima

Methods

all_pixels_above_h0	Check CSS11 criterion 1: The SSH values of all of the pixels are above a given SSH threshold for anticyclonic eddies.
all_pixels_below_h0	Check CSS11 criterion 1: The SSH values of all of the pixels are below a given SSH threshold for cyclonic eddies.
within_amplitude_limits	Need update

Attributes

EPSILON
amplitude
contour
grid_extract
h_0
interval_min
interval_min_secondary
mle
nb_pixel
pixel_mask
sla

EPSILON = 1e-08

all_pixels_above_h0(*level*)

Check CSS11 criterion 1: The SSH values of all of the pixels are above a given SSH threshold for anticy-

clonic eddies.

all_pixels_below_h0(*level*)
Check CSS11 criterion 1: The SSH values of all of the pixels are below a given SSH threshold for cyclonic eddies.

amplitude

contour

grid_extract

h_0

interval_min

interval_min_secondary

mle

nb_pixel

pixel_mask

sla

within_amplitude_limits()
Need update

13.7.4 py_eddy_tracker.eddy_feature.Contours

class py_eddy_tracker.eddy_feature.**Contours**(*x*, *y*, *z*, *levels*, *wrap_x=False*, *keep_unclose=False*)

Bases: object

Class to calculate average geostrophic velocity along a contour, *uavg*, and return index to contour with maximum *uavg* within a series of closed contours.

Attributes:

contour: A matplotlib contour object of high-pass filtered SLA

eddy: A tracklist object holding the SLA data

grd: A grid object

c_i : index to contours l_i : index to levels

Methods

check_closing	
display	Display contour
find_wrapcut_path_and_join	
get_index_nearest_path_bbox_contain_p	Get index from the nearest path in the level, if the bbox of the path contain pt
get_next	
iter	
label_contour_unused_which_contain_eddies	Select contour which contain several eddies

Attributes

DELTA_PREC
DELTA_SUP
contour_index
contours
cvalues
level_index
levels
nb_contour_per_level
nb_pt_per_contour
x_max_per_contour
x_min_per_contour
x_value
y_max_per_contour
y_min_per_contour
y_value

DELTA_PREC = 1e-10

DELTA_SUP = 0.01

check_closing(*path*)

contour_index

contours

property cvalues

display(*ax*, *step=1*, *only_used=False*, *only_unused=False*, *only_contain_eddies=False*, *display_criterion=False*, *field=None*, *bins=None*, *cmap='Spectral_r'*, ***kwargs*)

Display contour

Parameters

- **ax** (*matplotlib.axes.Axes*) –
- **step** (*int*) – display only contour every step
- **only_used** (*bool*) – display only contour used in an eddy
- **only_unused** (*bool*) – display only contour unused in an eddy
- **only_contain_eddies** (*bool*) – display only contour which enclosed an eddiy
- **display_criterion** (*bool*) – display only unused contour with criterion color
  0. – Accepted (green)
  1. – Reject for shape error (red)
  2. – Masked value in contour (blue)
  3. – Under or over pixel limit bound (black)
  4. – Amplitude criterion (yellow)
- **field** (*str*) – Must be ‘shape_error’, ‘x’, ‘y’ or ‘radius’. If define display_criterion is not use. bins argument must be define
- **bins** (*array*) – bins use to colorize contour

- **cmap** (*str*) – Name of cmap to use for field display
- **kwargs** (*dict*) – look at `matplotlib.collections.LineCollection()`

- *Eddy detection : Med*
- *Eddy detection : Gulf stream*

find_wrapcut_path_and_join (*x0*, *x1*)

get_index_nearest_path_bbox_contain_pt (*level*, *xpt*, *ypt*)
Get index from the nearest path in the level, if the bbox of the path contain pt

overhead of python is huge with numba, cython little bit best??

get_next (*origin*, *paths_left*, *paths_right*)

iter (*start=None*, *stop=None*, *step=None*)

label_contour_unused_which_contain_eddies (*eddies*)
Select contour which contain several eddies

level_index

property levels

nb_contour_per_level

nb_pt_per_contour

x_max_per_contour

x_min_per_contour

x_value

y_max_per_contour

y_min_per_contour

y_value

13.8 py_eddy_tracker.generic

Tool method which use mostly numba

Functions

bbox_indice_regular	Get bbox indice of a contour in a regular grid.
build_circle	Build circle from center coordinates.
build_index	We expected that variable is monotonous, and return index for each step change.
coordinates_to_local	Take latlong coordinates to transform in local coordinates (in m).
count_consecutive	Count consecutive event every False flag count restart
cumsum_by_track	Cumsum by track.
distance	Compute distance between points from each line.

continues on next page

Table 48 – continued from previous page

distance_grid	Get distance for every couple of point.
flatten_line_matrix	Flat matrix and add on between each line.
hist_numba	Call numba histogram to speed up.
interp2d_geo	For geographic grid, test of cicularity.
local_to_coordinates	Take local coordinates (in m) to transform to latlong.
nearest_grd_indice	Get nearest grid indice from a position.
reverse_index	Compute a list of index, which are not in index.
simplify	Will remove all middle/end point which are closer than precision.
split_line	Split x and y at each i change.
uniform_resample	Resample contours to have (nearly) equal spacing.
wrap_longitude	Will wrap contiguous longitude with reference as west bound.

13.8.1 py_eddy_tracker.generic.bbox_indice_regular

py_eddy_tracker.generic.**bbox_indice_regular**(*vertices*, *x0*, *y0*, *xstep*, *ystep*, *N*, *circular*, *x_size*)

Get bbox indice of a contour in a regular grid.

Parameters

- **vertices** – vertice of contour
- **x0** (*float*) – first grid longitude
- **y0** (*float*) – first grid latitude
- **xstep** (*float*) – step between two longitude
- **ystep** (*float*) – step between two latitude
- **N** (*int*) – shift of index to enlarge window
- **circular** (*bool*) – To know if grid is wrappable
- **x_size** (*int*) – Number of longitude

13.8.2 py_eddy_tracker.generic.build_circle

py_eddy_tracker.generic.**build_circle**(*x0*, *y0*, *r*)

Build circle from center coordinates.

Parameters

- **x0** (*float*) – center coordinate
- **y0** (*float*) – center coordinate
- **r** (*float*) – radius i meter

Returns x,y

Return type (array,array)

13.8.3 py_eddy_tracker.generic.build_index

py_eddy_tracker.generic.**build_index**(*groups*)

We expected that variable is monotonous, and return index for each step change.

Parameters **groups** (*array*) – array which contain group to be separated

Returns (first_index of each group, last_index of each group, value to shift group)

Return type (array, array, int)

13.8.4 py_eddy_tracker.generic.coordinates_to_local

py_eddy_tracker.generic.**coordinates_to_local**(*lon*, *lat*, *lon0*, *lat0*)

Take latlong coordinates to transform in local coordinates (in m).

Parameters

- **x** (*array*) – coordinates to transform
- **y** (*array*) – coordinates to transform
- **lon0** (*float*) – longitude of local reference
- **lat0** (*float*) – latitude of local reference

Returns x,y

Retype (array, array)

13.8.5 py_eddy_tracker.generic.count_consecutive

py_eddy_tracker.generic.**count_consecutive**(*mask*)

Count consecutive event every False flag count restart

Parameters **mask** (*array[bool]*) – event to count

Returns count when consecutive event

Return type array

13.8.6 py_eddy_tracker.generic.cumsum_by_track

py_eddy_tracker.generic.**cumsum_by_track**(*field*, *track*)

Cumsum by track.

Parameters **field** (*array*) – data to sum

Pram array(int) track id of track to separate data

Returns cumsum with a reset at each start of track

Return type array

13.8.7 py_eddy_tracker.generic.distance

py_eddy_tracker.generic.**distance**(*lon0*, *lat0*, *lon1*, *lat1*)
Compute distance between points from each line.

Parameters

- **lon0** (*float*) –
- **lat0** (*float*) –
- **lon1** (*float*) –
- **lat1** (*float*) –

Returns distance (in m)

Return type array

13.8.8 py_eddy_tracker.generic.distance_grid

py_eddy_tracker.generic.**distance_grid**(*lon0*, *lat0*, *lon1*, *lat1*)
Get distance for every couple of point.

Parameters

- **lon0** (*array*) –
- **lat0** (*array*) –
- **lon1** (*array*) –
- **lat1** (*array*) –

Returns nan value for far away point, and km for other

Return type array

13.8.9 py_eddy_tracker.generic.flatten_line_matrix

py_eddy_tracker.generic.**flatten_line_matrix**(*l_matrix*)
Flat matrix and add on between each line.

Parameters **l_matrix** – matrix of position

Returns array with nan between line

13.8.10 py_eddy_tracker.generic.hist_numba

py_eddy_tracker.generic.**hist_numba**(*x*, *bins*)
Call numba histogram to speed up.

13.8.11 py_eddy_tracker.generic.interp2d_geo

py_eddy_tracker.generic.**interp2d_geo**(*x_g*, *y_g*, *z_g*, *m_g*, *x*, *y*, *nearest=False*)

For geographic grid, test of cicularity.

Parameters

- **x_g** (*array*) – coordinates of grid
- **y_g** (*array*) – coordinates of grid
- **z_g** (*array*) – Grid value
- **m_g** (*array*) – Boolean grid, True if value is masked
- **x** (*array*) – coordinate where interpolate z
- **y** (*array*) – coordinate where interpolate z
- **nearest** (*bool*) – if true we will take nearest pixel

Returns z interpolated

Return type array

13.8.12 py_eddy_tracker.generic.local_to_coordinates

py_eddy_tracker.generic.**local_to_coordinates**(*x*, *y*, *lon0*, *lat0*)

Take local coordinates (in m) to transform to latlong.

Parameters

- **x** (*array*) – coordinates to transform
- **y** (*array*) – coordinates to transform
- **lon0** (*float*) – longitude of local reference
- **lat0** (*float*) – latitude of local reference

Returns lon,lat

Retype (array, array)

13.8.13 py_eddy_tracker.generic.nearest_grd_indice

py_eddy_tracker.generic.**nearest_grd_indice**(*x*, *y*, *x0*, *y0*, *xstep*, *ystep*)

Get nearest grid indice from a position.

Parameters

- **x** – longitude
- **y** – latitude
- **x0** (*float*) – first grid longitude
- **y0** (*float*) – first grid latitude
- **xstep** (*float*) – step between two longitude
- **ystep** (*float*) – step between two latitude

13.8.14 py_eddy_tracker.generic.reverse_index

py_eddy_tracker.generic.**reverse_index**(*index*, *nb*)
Compute a list of index, which are not in index.

Parameters

- **index** (*array*) – index of group which will be set to False
- **nb** (*array*) – Count for each group

Returns mask of value selected

Return type array

13.8.15 py_eddy_tracker.generic.simplify

py_eddy_tracker.generic.**simplify**(*x*, *y*, *precision=0.1*)
Will remove all middle/end point which are closer than precision.

Parameters

- **x** (*array*) –
- **y** (*array*) –
- **precision** (*float*) – if two points have distance inferior to precision with remove next point

Returns (x,y)

Return type (array,array)

13.8.16 py_eddy_tracker.generic.split_line

py_eddy_tracker.generic.**split_line**(*x*, *y*, *i*)
Split x and y at each i change.

Parameters

- **x** – array
- **y** – array
- **i** – array of int at each i change, we cut x, y

Returns x and y separate by nan at each i jump

13.8.17 py_eddy_tracker.generic.uniform_resample

py_eddy_tracker.generic.**uniform_resample**(*x_val*, *y_val*, *num_fac=2*, *fixed_size=None*)
Resample contours to have (nearly) equal spacing.

Parameters

- **x_val** (*array_like*) – input x contour coordinates
- **y_val** (*array_like*) – input y contour coordinates
- **num_fac** (*int*) – factor to increase lengths of output coordinates
- **fixed_size** (*int*, *None*) – if define, it will used to set sampling

13.8.18 py_eddy_tracker.generic.wrap_longitude

py_eddy_tracker.generic.**wrap_longitude**(*x*, *y*, *ref*, *cut=False*)

Will wrap contiguous longitude with reference as west bound.

Parameters

- **x** (*array*) –
- **y** (*array*) –
- **ref** (*float*) – longitude of reference, all the new value will be between ref and ref + 360
- **cut** (*bool*) – if True line will be cut at the bounds

Returns lon,lat

Return type (array,array)

13.9 py_eddy_tracker.gui

GUI class

Functions

no

13.9.1 py_eddy_tracker.gui.no

py_eddy_tracker.gui.**no**(**args*, ***kwargs*)

Classes

GUI	
GUIAxes	Axes which will use full space available
PlatCarreAxes	Class to replace missing pylook class

13.9.2 py_eddy_tracker.gui.GUI

class py_eddy_tracker.gui.**GUI**(***datasets*)

Bases: object

Methods

adjust
draw
event
get_infos
indexs
keyboard
med
move
press
release
scroll
set_initial_values
setup
show
update

Attributes

COLORS
KEYTIME
bbox
d_indexs
datasets
figure
last_event
m
map
now
param_ax
period
settings
time_ax

COLORS = ('r', 'g', 'b', 'y', 'k')

KEYTIME = {'down': -1, 'pagedown': -5, 'pageup': 5, 'up': 1}

adjust(*event=None*)

property bbox

d_indexs

datasets

draw()

event()

figure

get_infos(*name*, *index*)

indexs(*dataset*)

keyboard(*event*)

last_event

m

map

med()

move(*event*)

property now

param_ax

property period

press(*event*)

release(*event*)

scroll(*event*)

set_initial_values()

settings

setup()

show()

time_ax

update()

13.9.3 py_eddy_tracker.gui.GUIAxes

class py_eddy_tracker.gui.**GUIAxes**(**args*, ***kwargs*)

Bases: *py_eddy_tracker.gui.PlatCarreAxes*

Axes which will use full space available

Build an axes in a figure.

fig [*~matplotlib.figure.Figure*] The axes is build in the *.Figure fig*.

rect [[left, bottom, width, height]] The axes is build in the rectangle *rect*. *rect* is in *.Figure* coordinates.

sharex, sharey [*~.axes.Axes*, optional] The x or y *~.matplotlib.axis* is shared with the x or y axis in the input *~.axes.Axes*.

frameon [bool, default: True] Whether the axes frame is visible.

box_aspect [None, or a number, optional] Sets the aspect of the axes box. See *~.axes.Axes.set_box_aspect* for details.

****kwargs** Other optional keyword arguments:

Properties: adjustable: {'box', 'datalim'} agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array alpha: float or None anchor: 2-tuple of floats or {'C', 'SW', 'S', 'SE', ...} animated: bool aspect: {'auto'} or num autoscale_on: bool autoscalex_on: bool autoscaley_on: bool axes_locator: Callable[[Axes, Renderer], Bbox] axisbelow: bool or 'line' box_aspect: None, or a number clip_box: *.Bbox* clip_on: bool clip_path: Patch or (Path, Transform) or None contains: unknown facecolor or fc: color figure: *.Figure* frame_on: bool gid: str in_layout: bool label:

object navigate: bool navigate_mode: unknown path_effects: *.AbstractPathEffect* picker: None or bool or callable position: [left, bottom, width, height] or *~matplotlib.transforms.Bbox* prop_cycle: unknown rasterization_zorder: float or None rasterized: bool or None sketch_params: (scale: float, length: float, randomness: float) snap: bool or None title: str transform: *.Transform* url: str visible: bool xbound: unknown xlabel: str xlim: (bottom: float, top: float) xmargin: float greater than -0.5 xscale: {"linear", "log", "symlog", "logit", ... } xticklabels: unknown xticks: unknown ybound: unknown ylabel: str ylim: (bottom: float, top: float) ymargin: float greater than -0.5 yscale: {"linear", "log", "symlog", "logit", ... } yticklabels: unknown yticks: unknown zorder: float

~.axes.Axes The new *~.axes.Axes* object.

Methods

`acorr`	Plot the autocorrelation of *x*.
`add_artist`	Add an *~.Artist* to the axes, and return the artist.
`add_callback`	Add a callback function that will be called whenever one of the *.Artist*'s properties changes.
`add_child_axes`	Add an *~.AxesBase* to the axes' children; return the child axes.
`add_collection`	Add a *~.Collection* to the axes' collections; return the collection.
`add_container`	Add a *~.Container* to the axes' containers; return the container.
`add_image`	Add an *~.AxesImage* to the axes' images; return the image.
`add_line`	Add a *.Line2D* to the axes' lines; return the line.
`add_patch`	Add a *~.Patch* to the axes' patches; return the patch.
`add_table`	Add a *~.Table* to the axes' tables; return the table.
`angle_spectrum`	Plot the angle spectrum.
`annotate`	Annotate the point *xy* with text *text*.
`apply_aspect`	Adjust the Axes for a specified data aspect ratio.
`arrow`	Add an arrow to the axes.
`autoscale`	Autoscale the axis view to the data (toggle).
`autoscale_view`	Autoscale the view limits using the data limits.
`axhline`	Add a horizontal line across the axis.
`axhspan`	Add a horizontal span (rectangle) across the axis.
`axis`	Convenience method to get or set some axis properties.
`axline`	Add an infinitely long straight line.
`axvline`	Add a vertical line across the axes.
`axvspan`	Add a vertical span (rectangle) across the axes.
`bar`	Make a bar plot.
`barbs`	Plot a 2D field of barbs.
`barh`	Make a horizontal bar plot.
`boxplot`	Make a box and whisker plot.
`broken_barh`	Plot a horizontal sequence of rectangles.
`bxp`	Drawing function for box and whisker plots.
`can_pan`	Return *True* if this axes supports any pan/zoom button functionality.

continues on next page

Table 53 – continued from previous page

can_zoom	Return *True* if this axes supports the zoom box button functionality.
cla	Clear the current axes.
clabel	Label a contour plot.
clear	Clear the axes.
cohere	Plot the coherence between *x* and *y*.
contains	Test whether the artist contains the mouse event.
contains_point	Return whether *point* (pair of pixel coordinates) is inside the axes patch.
contour	Plot contours.
contourf	Plot contours.
convert_xunits	Convert *x* using the unit type of the xaxis.
convert_yunits	Convert *y* using the unit type of the yaxis.
csd	Plot the cross-spectral density.
drag_pan	Called when the mouse moves during a pan operation.
draw	Draw the Artist (and its children) using the given renderer.
draw_artist	Efficiently redraw a single artist.
end_pan	Called when a pan operation completes (when the mouse button is up.)
errorbar	Plot y versus x as lines and/or markers with attached errorbars.
eventplot	Plot identical parallel lines at the given positions.
fill	Plot filled polygons.
fill_between	Fill the area between two horizontal curves.
fill_betweenx	Fill the area between two vertical curves.
findobj	Find artist objects.
format_coord	Return a format string formatting the *x*, *y* coordinates.
format_cursor_data	Return a string representation of *data*.
format_xdata	Return *x* formatted as an x-value.
format_ydata	Return *y* formatted as an y-value.
get_adjustable	Return whether the Axes will adjust its physical dimension ('box') or its data limits ('datalim') to achieve the desired aspect ratio.
get_agg_filter	Return filter function to be used for agg filter.
get_alpha	Return the alpha value used for blending - not supported on all backends.
get_anchor	Get the anchor location.
get_animated	Return whether the artist is animated.
get_aspect	
get_autoscale_on	Get whether autoscaling is applied for both axes on plot commands
get_autoscalex_on	Get whether autoscaling for the x-axis is applied on plot commands
get_autoscaley_on	Get whether autoscaling for the y-axis is applied on plot commands
get_axes_locator	Return the axes_locator.
get_axisbelow	Get whether axis ticks and gridlines are above or below most artists.

continues on next page

Table 53 – continued from previous page

get_box_aspect	Get the axes box aspect.
get_children	Return a list of the child *.Artist*s of this *.Artist*.
get_clip_box	Return the clipbox.
get_clip_on	Return whether the artist uses clipping.
get_clip_path	Return the clip path.
get_contains	[*Deprecated*] Return the custom contains function of the artist if set, or *None*.
get_cursor_data	Return the cursor data for a given event.
get_data_ratio	Return the aspect ratio of the scaled data.
get_data_ratio_log	[*Deprecated*] Return the aspect ratio of the raw data in log scale.
get_default_bbox_extra_artists	Return a default list of artists that are used for the bounding box calculation.
get_facecolor	Get the facecolor of the Axes.
get_fc	Alias for *get_facecolor*.
get_figure	Return the *.Figure* instance the artist belongs to.
get_frame_on	Get whether the axes rectangle patch is drawn.
get_gid	Return the group id.
get_images	Return a list of *.AxesImage*s contained by the Axes.
get_in_layout	Return boolean flag, `True` if artist is included in layout calculations.
get_label	Return the label used for this artist in the legend.
get_legend	Return the *.Legend* instance, or None if no legend is defined.
get_legend_handles_labels	Return handles and labels for legend
get_lines	Return a list of lines contained by the Axes.
get_navigate	Get whether the axes responds to navigation commands
get_navigate_mode	Get the navigation toolbar button status: 'PAN', 'ZOOM', or None
get_path_effects	
get_picker	Return the picking behavior of the artist.
get_position	Get a copy of the axes rectangle as a *.Bbox*.
get_rasterization_zorder	Return the zorder value below which artists will be rasterized.
get_rasterized	Return whether the artist is to be rasterized.
get_renderer_cache	
get_shared_x_axes	Return a reference to the shared axes Grouper object for x axes.
get_shared_y_axes	Return a reference to the shared axes Grouper object for y axes.
get_sketch_params	Return the sketch parameters for the artist.
get_snap	Return the snap setting.
get_tightbbox	Return the tight bounding box of the axes, including axis and their decorators (xlabel, title, etc).
get_title	Get an axes title.
get_transform	Return the *.Transform* instance used by this artist.
get_transformed_clip_path_and_affine	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.
get_url	Return the url.

continues on next page

Table 53 – continued from previous page

get_visible	Return the visibility.
get_window_extent	Return the axes bounding box in display space; *args* and *kwargs* are empty.
get_xaxis	Return the XAxis instance.
get_xaxis_text1_transform	Returns
get_xaxis_text2_transform	Returns
get_xaxis_transform	Get the transformation used for drawing x-axis labels, ticks and gridlines.
get_xbound	Return the lower and upper x-axis bounds, in increasing order.
get_xgridlines	Return the xaxis' grid lines as a list of *.Line2D*s.
get_xlabel	Get the xlabel text string.
get_xlim	Return the x-axis view limits.
get_xmajorticklabels	Return the xaxis' major tick labels, as a list of *~.text.Text*.
get_xminorticklabels	Return the xaxis' minor tick labels, as a list of *~.text.Text*.
get_xscale	Return the xaxis' scale (as a str).
get_xticklabels	Get the xaxis' tick labels.
get_xticklines	Return the xaxis' tick lines as a list of *.Line2D*s.
get_xticks	Return the xaxis' tick locations in data coordinates.
get_yaxis	Return the YAxis instance.
get_yaxis_text1_transform	Returns
get_yaxis_text2_transform	Returns
get_yaxis_transform	Get the transformation used for drawing y-axis labels, ticks and gridlines.
get_ybound	Return the lower and upper y-axis bounds, in increasing order.
get_ygridlines	Return the yaxis' grid lines as a list of *.Line2D*s.
get_ylabel	Get the ylabel text string.
get_ylim	Return the y-axis view limits.
get_ymajorticklabels	Return the yaxis' major tick labels, as a list of *~.text.Text*.
get_yminorticklabels	Return the yaxis' minor tick labels, as a list of *~.text.Text*.
get_yscale	Return the yaxis' scale (as a str).
get_yticklabels	Get the yaxis' tick labels.
get_yticklines	Return the yaxis' tick lines as a list of *.Line2D*s.
get_yticks	Return the yaxis' tick locations in data coordinates.
get_zorder	Return the artist's zorder.
grid	Configure the grid lines.
has_data	Return *True* if any artists have been added to axes.
have_units	Return *True* if units are set on any axis.
hexbin	Make a 2D hexagonal binning plot of points *x*, *y*.
hist	Plot a histogram.
hist2d	Make a 2D histogram plot.
hlines	Plot horizontal lines at each *y* from *xmin* to *xmax*.
imshow	Display data as an image, i.e., on a 2D regular raster.
in_axes	Return *True* if the given *mouseevent* (in display coords) is in the Axes
indicate_inset	Add an inset indicator to the axes.

continues on next page

Table 53 – continued from previous page

indicate_inset_zoom	Add an inset indicator rectangle to the axes based on the axis limits for an *inset_ax* and draw connectors between *inset_ax* and the rectangle.
inset_axes	Add a child inset axes to this existing axes.
invert_xaxis	Invert the x-axis.
invert_yaxis	Invert the y-axis.
is_transform_set	Return whether the Artist has an explicitly set transform.
legend	Place a legend on the axes.
locator_params	Control behavior of major tick locators.
loglog	Make a plot with log scaling on both the x and y axis.
magnitude_spectrum	Plot the magnitude spectrum.
margins	Set or retrieve autoscaling margins.
matshow	Plot the values of a 2D matrix or array as color-coded image.
minorticks_off	Remove minor ticks from the axes.
minorticks_on	Display minor ticks on the axes.
pchanged	Call all of the registered callbacks.
pcolor	Create a pseudocolor plot with a non-regular rectangular grid.
pcolorfast	Create a pseudocolor plot with a non-regular rectangular grid.
pcolormesh	Create a pseudocolor plot with a non-regular rectangular grid.
phase_spectrum	Plot the phase spectrum.
pick	Process a pick event.
pickable	Return whether the artist is pickable.
pie	Plot a pie chart.
plot	Plot y versus x as lines and/or markers.
plot_date	Plot data that contains dates.
properties	Return a dictionary of all the properties of the artist.
psd	Plot the power spectral density.
quiver	Plot a 2D field of arrows.
quiverkey	Add a key to a quiver plot.
redraw_in_frame	Efficiently redraw Axes data, but not axis ticks, labels, etc.
relim	Recompute the data limits based on current artists.
remove	Remove the artist from the figure if possible.
remove_callback	Remove a callback based on its observer id.
reset_position	Reset the active position to the original position.
scatter	A scatter plot of *y* vs.
secondary_xaxis	Add a second x-axis to this axes.
secondary_yaxis	Add a second y-axis to this axes.
semilogx	Make a plot with log scaling on the x axis.
semilogy	Make a plot with log scaling on the y axis.
set	A property batch setter.
set_adjustable	Set how the Axes adjusts to achieve the required aspect ratio.
set_agg_filter	Set the agg filter.
set_alpha	Set the alpha value used for blending - not supported on all backends.

continues on next page

Table 53 – continued from previous page

set_anchor	Define the anchor location.
set_animated	Set the artist's animation state.
set_aspect	Set the aspect of the axis scaling, i.e. the ratio of y-unit to x-unit.
set_autoscale_on	Set whether autoscaling is applied on plot commands
set_autoscalex_on	Set whether autoscaling for the x-axis is applied on plot commands
set_autoscaley_on	Set whether autoscaling for the y-axis is applied on plot commands
set_axes_locator	Set the axes locator.
set_axis_off	Turn the x- and y-axis off.
set_axis_on	Turn the x- and y-axis on.
set_axisbelow	Set whether axis ticks and gridlines are above or below most artists.
set_box_aspect	Set the axes box aspect.
set_clip_box	Set the artist's clip *.Bbox*.
set_clip_on	Set whether the artist uses clipping.
set_clip_path	Set the artist's clip path.
set_contains	[*Deprecated*] Define a custom contains test for the artist.
set_facecolor	Set the facecolor of the Axes.
set_fc	Alias for *set_facecolor*.
set_figure	Set the *.Figure* instance the artist belongs to.
set_frame_on	Set whether the axes rectangle patch is drawn.
set_gid	Set the (group) id for the artist.
set_in_layout	Set if artist is to be included in layout calculations, E.g.
set_label	Set a label that will be displayed in the legend.
set_navigate	Set whether the axes responds to navigation toolbar commands
set_navigate_mode	Set the navigation toolbar button status;
set_path_effects	Set the path effects.
set_picker	Define the picking behavior of the artist.
set_position	Set the axes position.
set_prop_cycle	Set the property cycle of the Axes.
set_rasterization_zorder	Parameters
set_rasterized	Force rasterized (bitmap) drawing in vector backend output.
set_sketch_params	Sets the sketch parameters.
set_snap	Set the snapping behavior.
set_title	Set a title for the axes.
set_transform	Set the artist transform.
set_url	Set the url for the artist.
set_visible	Set the artist's visibility.
set_xbound	Set the lower and upper numerical bounds of the x-axis.
set_xlabel	Set the label for the x-axis.
set_xlim	Set the x-axis view limits.
set_xmargin	Set padding of X data limits prior to autoscaling.
set_xscale	Set the x-axis scale.
set_xticklabels	Set the xaxis' labels with list of string labels.

continues on next page

Table 53 – continued from previous page

set_xticks	Set the xaxis' tick locations.
set_ybound	Set the lower and upper numerical bounds of the y-axis.
set_ylabel	Set the label for the y-axis.
set_ylim	Set the y-axis view limits.
set_ymargin	Set padding of Y data limits prior to autoscaling.
set_yscale	Set the y-axis scale.
set_yticklabels	Set the yaxis' labels with list of string labels.
set_yticks	Set the yaxis' tick locations.
set_zorder	Set the zorder for the artist.
sharex	Share the x-axis with *other*.
sharey	Share the y-axis with *other*.
specgram	Plot a spectrogram.
spy	Plot the sparsity pattern of a 2D array.
stackplot	Draw a stacked area plot.
start_pan	Called when a pan operation has started.
stem	Create a stem plot.
step	Make a step plot.
streamplot	Draw streamlines of a vector flow.
table	Add a table to an *~.axes.Axes*.
text	Add text to the axes.
tick_params	Change the appearance of ticks, tick labels, and grid-lines.
ticklabel_format	Configure the *.ScalarFormatter* used by default for linear axes.
tricontour	Draw contour lines on an unstructured triangular grid.
tricontourf	Draw contour regions on an unstructured triangular grid.
tripcolor	Create a pseudocolor plot of an unstructured triangular grid.
triplot	Draw a unstructured triangular grid as lines and/or markers.
twinx	Create a twin Axes sharing the xaxis.
twiny	Create a twin Axes sharing the yaxis.
update	Update this artist's properties from the dict *props*.
update_datalim	Extend the *~.Axes.dataLim* Bbox to include the given points.
update_datalim_bounds	[*Deprecated*] Extend the *~.Axes.datalim* Bbox to include the given *~matplotlib.transforms.Bbox*.
update_from	Copy properties from *other* to *self*.
violin	Drawing function for violin plots.
violinplot	Make a violin plot.
vlines	Plot vertical lines.
xaxis_date	Sets up axis ticks and labels to treat data along the xaxis as dates.
xaxis_inverted	Return whether the xaxis is oriented in the "inverse" direction.
xcorr	Plot the cross correlation between *x* and *y*.
yaxis_date	Sets up axis ticks and labels to treat data along the yaxis as dates.

continues on next page

Table 53 – continued from previous page

yaxis_inverted	Return whether the yaxis is oriented in the "inverse" direction.

Attributes

axes	The ~*.axes.Axes* instance the artist resides in, or *None*.
mouseover	If this property is set to *True*, the artist will be queried for custom context information when the mouse cursor moves over it.
name	
stale	Whether the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.
sticky_edges	x and y sticky edge lists for autoscaling.
use_sticky_edges	When autoscaling, whether to obey all *Artist.sticky_edges*.
viewLim	
zorder	

end_pan(**args*, ***kwargs*)

Called when a pan operation completes (when the mouse button is up.)

This is intended to be overridden by new projection types.

name = 'full_axes'

13.9.4 py_eddy_tracker.gui.PlatCarreAxes

class py_eddy_tracker.gui.**PlatCarreAxes**(**args*, ***kwargs*)

Bases: matplotlib.axes._axes.Axes

Class to replace missing pylook class

Build an axes in a figure.

fig [~*matplotlib.figure.Figure*] The axes is build in the *.Figure fig*.

rect [[left, bottom, width, height]] The axes is build in the rectangle *rect*. *rect* is in *.Figure* coordinates.

sharex, sharey [~*.axes.Axes*, optional] The x or y ~*.matplotlib.axis* is shared with the x or y axis in the input ~*.axes.Axes*.

frameon [bool, default: True] Whether the axes frame is visible.

box_aspect [None, or a number, optional] Sets the aspect of the axes box. See ~*.axes.Axes.set_box_aspect* for details.

****kwargs** Other optional keyword arguments:

Properties: adjustable: {'box', 'datalim'} agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array alpha: float or None anchor: 2-tuple of floats or {'C', 'SW', 'S', 'SE', ...} animated: bool aspect: {'auto'} or num autoscale_on: bool autoscalex_on: bool autoscaley_on: bool axes_locator: Callable[[Axes, Renderer], Bbox] axisbelow: bool or 'line' box_aspect: None, or a number clip_box: *.Bbox* clip_on: bool clip_path: Patch or (Path, Transform) or None contains: unknown facecolor or fc: color figure: *.Figure* frame_on: bool gid: str in_layout: bool label:

object navigate: bool navigate_mode: unknown path_effects: *.AbstractPathEffect* picker: None or bool or callable position: [left, bottom, width, height] or *~matplotlib.transforms.Bbox* prop_cycle: unknown rasterization_zorder: float or None rasterized: bool or None sketch_params: (scale: float, length: float, randomness: float) snap: bool or None title: str transform: *.Transform* url: str visible: bool xbound: unknown xlabel: str xlim: (bottom: float, top: float) xmargin: float greater than -0.5 xscale: {"linear", "log", "symlog", "logit", ... } xticklabels: unknown xticks: unknown ybound: unknown ylabel: str ylim: (bottom: float, top: float) ymargin: float greater than -0.5 yscale: {"linear", "log", "symlog", "logit", ... } yticklabels: unknown yticks: unknown zorder: float

~.axes.Axes The new *~.axes.Axes* object.

Methods

`acorr`	Plot the autocorrelation of *x*.
`add_artist`	Add an *~.Artist* to the axes, and return the artist.
`add_callback`	Add a callback function that will be called whenever one of the *.Artist*'s properties changes.
`add_child_axes`	Add an *~.AxesBase* to the axes' children; return the child axes.
`add_collection`	Add a *~.Collection* to the axes' collections; return the collection.
`add_container`	Add a *~.Container* to the axes' containers; return the container.
`add_image`	Add an *~.AxesImage* to the axes' images; return the image.
`add_line`	Add a *.Line2D* to the axes' lines; return the line.
`add_patch`	Add a *~.Patch* to the axes' patches; return the patch.
`add_table`	Add a *~.Table* to the axes' tables; return the table.
`angle_spectrum`	Plot the angle spectrum.
`annotate`	Annotate the point *xy* with text *text*.
`apply_aspect`	Adjust the Axes for a specified data aspect ratio.
`arrow`	Add an arrow to the axes.
`autoscale`	Autoscale the axis view to the data (toggle).
`autoscale_view`	Autoscale the view limits using the data limits.
`axhline`	Add a horizontal line across the axis.
`axhspan`	Add a horizontal span (rectangle) across the axis.
`axis`	Convenience method to get or set some axis properties.
`axline`	Add an infinitely long straight line.
`axvline`	Add a vertical line across the axes.
`axvspan`	Add a vertical span (rectangle) across the axes.
`bar`	Make a bar plot.
`barbs`	Plot a 2D field of barbs.
`barh`	Make a horizontal bar plot.
`boxplot`	Make a box and whisker plot.
`broken_barh`	Plot a horizontal sequence of rectangles.
`bxp`	Drawing function for box and whisker plots.
`can_pan`	Return *True* if this axes supports any pan/zoom button functionality.

continues on next page

Table 55 – continued from previous page

can_zoom	Return *True* if this axes supports the zoom box button functionality.
cla	Clear the current axes.
clabel	Label a contour plot.
clear	Clear the axes.
cohere	Plot the coherence between *x* and *y*.
contains	Test whether the artist contains the mouse event.
contains_point	Return whether *point* (pair of pixel coordinates) is inside the axes patch.
contour	Plot contours.
contourf	Plot contours.
convert_xunits	Convert *x* using the unit type of the xaxis.
convert_yunits	Convert *y* using the unit type of the yaxis.
csd	Plot the cross-spectral density.
drag_pan	Called when the mouse moves during a pan operation.
draw	Draw the Artist (and its children) using the given renderer.
draw_artist	Efficiently redraw a single artist.
end_pan	Called when a pan operation completes (when the mouse button is up.)
errorbar	Plot y versus x as lines and/or markers with attached errorbars.
eventplot	Plot identical parallel lines at the given positions.
fill	Plot filled polygons.
fill_between	Fill the area between two horizontal curves.
fill_betweenx	Fill the area between two vertical curves.
findobj	Find artist objects.
format_coord	Return a format string formatting the *x*, *y* coordinates.
format_cursor_data	Return a string representation of *data*.
format_xdata	Return *x* formatted as an x-value.
format_ydata	Return *y* formatted as an y-value.
get_adjustable	Return whether the Axes will adjust its physical dimension (‘box’) or its data limits (‘datalim’) to achieve the desired aspect ratio.
get_agg_filter	Return filter function to be used for agg filter.
get_alpha	Return the alpha value used for blending - not supported on all backends.
get_anchor	Get the anchor location.
get_animated	Return whether the artist is animated.
get_aspect	
get_autoscale_on	Get whether autoscaling is applied for both axes on plot commands
get_autoscalex_on	Get whether autoscaling for the x-axis is applied on plot commands
get_autoscaley_on	Get whether autoscaling for the y-axis is applied on plot commands
get_axes_locator	Return the axes_locator.
get_axisbelow	Get whether axis ticks and gridlines are above or below most artists.

continues on next page

Table 55 – continued from previous page

get_box_aspect	Get the axes box aspect.
get_children	Return a list of the child *.Artist*s of this *.Artist*.
get_clip_box	Return the clipbox.
get_clip_on	Return whether the artist uses clipping.
get_clip_path	Return the clip path.
get_contains	[*Deprecated*] Return the custom contains function of the artist if set, or *None*.
get_cursor_data	Return the cursor data for a given event.
get_data_ratio	Return the aspect ratio of the scaled data.
get_data_ratio_log	[*Deprecated*] Return the aspect ratio of the raw data in log scale.
get_default_bbox_extra_artists	Return a default list of artists that are used for the bounding box calculation.
get_facecolor	Get the facecolor of the Axes.
get_fc	Alias for *get_facecolor*.
get_figure	Return the *.Figure* instance the artist belongs to.
get_frame_on	Get whether the axes rectangle patch is drawn.
get_gid	Return the group id.
get_images	Return a list of *.AxesImage*s contained by the Axes.
get_in_layout	Return boolean flag, `True` if artist is included in layout calculations.
get_label	Return the label used for this artist in the legend.
get_legend	Return the *.Legend* instance, or None if no legend is defined.
get_legend_handles_labels	Return handles and labels for legend
get_lines	Return a list of lines contained by the Axes.
get_navigate	Get whether the axes responds to navigation commands
get_navigate_mode	Get the navigation toolbar button status: 'PAN', 'ZOOM', or None
get_path_effects	
get_picker	Return the picking behavior of the artist.
get_position	Get a copy of the axes rectangle as a *.Bbox*.
get_rasterization_zorder	Return the zorder value below which artists will be rasterized.
get_rasterized	Return whether the artist is to be rasterized.
get_renderer_cache	
get_shared_x_axes	Return a reference to the shared axes Grouper object for x axes.
get_shared_y_axes	Return a reference to the shared axes Grouper object for y axes.
get_sketch_params	Return the sketch parameters for the artist.
get_snap	Return the snap setting.
get_tightbbox	Return the tight bounding box of the axes, including axis and their decorators (xlabel, title, etc).
get_title	Get an axes title.
get_transform	Return the *.Transform* instance used by this artist.
get_transformed_clip_path_and_affine	Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.
get_url	Return the url.

continues on next page

Table 55 – continued from previous page

get_visible	Return the visibility.
get_window_extent	Return the axes bounding box in display space; *args* and *kwargs* are empty.
get_xaxis	Return the XAxis instance.
get_xaxis_text1_transform	Returns
get_xaxis_text2_transform	Returns
get_xaxis_transform	Get the transformation used for drawing x-axis labels, ticks and gridlines.
get_xbound	Return the lower and upper x-axis bounds, in increasing order.
get_xgridlines	Return the xaxis' grid lines as a list of *.Line2D*s.
get_xlabel	Get the xlabel text string.
get_xlim	Return the x-axis view limits.
get_xmajorticklabels	Return the xaxis' major tick labels, as a list of *~.text.Text*.
get_xminorticklabels	Return the xaxis' minor tick labels, as a list of *~.text.Text*.
get_xscale	Return the xaxis' scale (as a str).
get_xticklabels	Get the xaxis' tick labels.
get_xticklines	Return the xaxis' tick lines as a list of *.Line2D*s.
get_xticks	Return the xaxis' tick locations in data coordinates.
get_yaxis	Return the YAxis instance.
get_yaxis_text1_transform	Returns
get_yaxis_text2_transform	Returns
get_yaxis_transform	Get the transformation used for drawing y-axis labels, ticks and gridlines.
get_ybound	Return the lower and upper y-axis bounds, in increasing order.
get_ygridlines	Return the yaxis' grid lines as a list of *.Line2D*s.
get_ylabel	Get the ylabel text string.
get_ylim	Return the y-axis view limits.
get_ymajorticklabels	Return the yaxis' major tick labels, as a list of *~.text.Text*.
get_yminorticklabels	Return the yaxis' minor tick labels, as a list of *~.text.Text*.
get_yscale	Return the yaxis' scale (as a str).
get_yticklabels	Get the yaxis' tick labels.
get_yticklines	Return the yaxis' tick lines as a list of *.Line2D*s.
get_yticks	Return the yaxis' tick locations in data coordinates.
get_zorder	Return the artist's zorder.
grid	Configure the grid lines.
has_data	Return *True* if any artists have been added to axes.
have_units	Return *True* if units are set on any axis.
hexbin	Make a 2D hexagonal binning plot of points *x*, *y*.
hist	Plot a histogram.
hist2d	Make a 2D histogram plot.
hlines	Plot horizontal lines at each *y* from *xmin* to *xmax*.
imshow	Display data as an image, i.e., on a 2D regular raster.
in_axes	Return *True* if the given *mouseevent* (in display coords) is in the Axes
indicate_inset	Add an inset indicator to the axes.

continues on next page

Table 55 – continued from previous page

indicate_inset_zoom	Add an inset indicator rectangle to the axes based on the axis limits for an *inset_ax* and draw connectors between *inset_ax* and the rectangle.
inset_axes	Add a child inset axes to this existing axes.
invert_xaxis	Invert the x-axis.
invert_yaxis	Invert the y-axis.
is_transform_set	Return whether the Artist has an explicitly set transform.
legend	Place a legend on the axes.
locator_params	Control behavior of major tick locators.
loglog	Make a plot with log scaling on both the x and y axis.
magnitude_spectrum	Plot the magnitude spectrum.
margins	Set or retrieve autoscaling margins.
matshow	Plot the values of a 2D matrix or array as color-coded image.
minorticks_off	Remove minor ticks from the axes.
minorticks_on	Display minor ticks on the axes.
pchanged	Call all of the registered callbacks.
pcolor	Create a pseudocolor plot with a non-regular rectangular grid.
pcolorfast	Create a pseudocolor plot with a non-regular rectangular grid.
pcolormesh	Create a pseudocolor plot with a non-regular rectangular grid.
phase_spectrum	Plot the phase spectrum.
pick	Process a pick event.
pickable	Return whether the artist is pickable.
pie	Plot a pie chart.
plot	Plot y versus x as lines and/or markers.
plot_date	Plot data that contains dates.
properties	Return a dictionary of all the properties of the artist.
psd	Plot the power spectral density.
quiver	Plot a 2D field of arrows.
quiverkey	Add a key to a quiver plot.
redraw_in_frame	Efficiently redraw Axes data, but not axis ticks, labels, etc.
relim	Recompute the data limits based on current artists.
remove	Remove the artist from the figure if possible.
remove_callback	Remove a callback based on its observer id.
reset_position	Reset the active position to the original position.
scatter	A scatter plot of *y* vs.
secondary_xaxis	Add a second x-axis to this axes.
secondary_yaxis	Add a second y-axis to this axes.
semilogx	Make a plot with log scaling on the x axis.
semilogy	Make a plot with log scaling on the y axis.
set	A property batch setter.
set_adjustable	Set how the Axes adjusts to achieve the required aspect ratio.
set_agg_filter	Set the agg filter.
set_alpha	Set the alpha value used for blending - not supported on all backends.

continues on next page

Table 55 – continued from previous page

set_anchor	Define the anchor location.
set_animated	Set the artist's animation state.
set_aspect	Set the aspect of the axis scaling, i.e. the ratio of y-unit to x-unit.
set_autoscale_on	Set whether autoscaling is applied on plot commands
set_autoscalex_on	Set whether autoscaling for the x-axis is applied on plot commands
set_autoscaley_on	Set whether autoscaling for the y-axis is applied on plot commands
set_axes_locator	Set the axes locator.
set_axis_off	Turn the x- and y-axis off.
set_axis_on	Turn the x- and y-axis on.
set_axisbelow	Set whether axis ticks and gridlines are above or below most artists.
set_box_aspect	Set the axes box aspect.
set_clip_box	Set the artist's clip *.Bbox*.
set_clip_on	Set whether the artist uses clipping.
set_clip_path	Set the artist's clip path.
set_contains	[*Deprecated*] Define a custom contains test for the artist.
set_facecolor	Set the facecolor of the Axes.
set_fc	Alias for *set_facecolor*.
set_figure	Set the *.Figure* instance the artist belongs to.
set_frame_on	Set whether the axes rectangle patch is drawn.
set_gid	Set the (group) id for the artist.
set_in_layout	Set if artist is to be included in layout calculations, E.g.
set_label	Set a label that will be displayed in the legend.
set_navigate	Set whether the axes responds to navigation toolbar commands
set_navigate_mode	Set the navigation toolbar button status;
set_path_effects	Set the path effects.
set_picker	Define the picking behavior of the artist.
set_position	Set the axes position.
set_prop_cycle	Set the property cycle of the Axes.
set_rasterization_zorder	Parameters
set_rasterized	Force rasterized (bitmap) drawing in vector backend output.
set_sketch_params	Sets the sketch parameters.
set_snap	Set the snapping behavior.
set_title	Set a title for the axes.
set_transform	Set the artist transform.
set_url	Set the url for the artist.
set_visible	Set the artist's visibility.
set_xbound	Set the lower and upper numerical bounds of the x-axis.
set_xlabel	Set the label for the x-axis.
set_xlim	Set the x-axis view limits.
set_xmargin	Set padding of X data limits prior to autoscaling.
set_xscale	Set the x-axis scale.
set_xticklabels	Set the xaxis' labels with list of string labels.

continues on next page

Table 55 – continued from previous page

set_xticks	Set the xaxis' tick locations.
set_ybound	Set the lower and upper numerical bounds of the y-axis.
set_ylabel	Set the label for the y-axis.
set_ylim	Set the y-axis view limits.
set_ymargin	Set padding of Y data limits prior to autoscaling.
set_yscale	Set the y-axis scale.
set_yticklabels	Set the yaxis' labels with list of string labels.
set_yticks	Set the yaxis' tick locations.
set_zorder	Set the zorder for the artist.
sharex	Share the x-axis with *other*.
sharey	Share the y-axis with *other*.
specgram	Plot a spectrogram.
spy	Plot the sparsity pattern of a 2D array.
stackplot	Draw a stacked area plot.
start_pan	Called when a pan operation has started.
stem	Create a stem plot.
step	Make a step plot.
streamplot	Draw streamlines of a vector flow.
table	Add a table to an *~.axes.Axes*.
text	Add text to the axes.
tick_params	Change the appearance of ticks, tick labels, and grid-lines.
ticklabel_format	Configure the *.ScalarFormatter* used by default for linear axes.
tricontour	Draw contour lines on an unstructured triangular grid.
tricontourf	Draw contour regions on an unstructured triangular grid.
tripcolor	Create a pseudocolor plot of an unstructured triangular grid.
triplot	Draw a unstructured triangular grid as lines and/or markers.
twinx	Create a twin Axes sharing the xaxis.
twiny	Create a twin Axes sharing the yaxis.
update	Update this artist's properties from the dict *props*.
update_datalim	Extend the *~.Axes.dataLim* Bbox to include the given points.
update_datalim_bounds	[*Deprecated*] Extend the *~.Axes.datalim* Bbox to include the given *~matplotlib.transforms.Bbox*.
update_from	Copy properties from *other* to *self*.
violin	Drawing function for violin plots.
violinplot	Make a violin plot.
vlines	Plot vertical lines.
xaxis_date	Sets up axis ticks and labels to treat data along the xaxis as dates.
xaxis_inverted	Return whether the xaxis is oriented in the "inverse" direction.
xcorr	Plot the cross correlation between *x* and *y*.
yaxis_date	Sets up axis ticks and labels to treat data along the yaxis as dates.

continues on next page

Table 55 – continued from previous page

yaxis_inverted	Return whether the yaxis is oriented in the "inverse" direction.

Attributes

axes	The ~.*axes.Axes* instance the artist resides in, or *None*.
mouseover	If this property is set to *True*, the artist will be queried for custom context information when the mouse cursor moves over it.
name	
stale	Whether the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.
sticky_edges	x and y sticky edge lists for autoscaling.
use_sticky_edges	When autoscaling, whether to obey all *Artist.sticky_edges*.
viewLim	
zorder	

13.10 py_eddy_tracker.poly

Method for polygon

Functions

bbox_intersection	Compute bbox to check if there are a bbox intersection.
close_center	Compute an overlap with circle parameter and return a percentage
convex	Check if polygon is convex
convexs	Check if polygons are convex
create_vertice	Return polygon vertice.
create_vertice_from_2darray	Choose a polygon in x,y list and return vertice.
fit_circle	From a polygon, function will fit a circle.
*fit_circle_*	From a polygon, function will fit a circle.
get_convex_hull	Get convex polygon which enclosed current polygon
get_pixel_in_regular	Get a pixel list of a regular grid contain in a contour.
get_wrap_vertice	Return a vertice for each polygon and check that use same reference coordinates.
is_left	Test if point is left of an infinit line.
merge	Merge all polygon of the list
poly_area	Must be call with local coordinates (in m, to get an area in m^2).
poly_area_vertice	Compute area from vertice.
poly_contain_poly	Check if poly_in is include in poly_out.
polygon_overlap	Return percent of overlap for each item.
shape_error	With a polygon(x,y) in local coordinates.

continues on next page

Table 57 – continued from previous page

tri_area2	Double area of triangle
vertice_overlap	Return percent of overlap for each item.
visvalingam	Polygon simplification with visvalingam algorithm
winding_number_grid_in_poly	Return index for each grid coordinates within contour.
winding_number_poly	Check if x,y is in poly.

13.10.1 py_eddy_tracker.poly.bbox_intersection

py_eddy_tracker.poly.**bbox_intersection**(*x0*, *y0*, *x1*, *y1*)

Compute bbox to check if there are a bbox intersection.

Parameters

- **x0** (*array*) – x for polygon list 0
- **y0** (*array*) – y for polygon list 0
- **x1** (*array*) – x for polygon list 1
- **y1** (*array*) – y for polygon list 1

Returns index of each polygon bbox which have an intersection

Return type (int, int)

13.10.2 py_eddy_tracker.poly.close_center

py_eddy_tracker.poly.**close_center**(*x0*, *y0*, *x1*, *y1*, *delta=0.1*)

Compute an overlap with circle parameter and return a percentage

Parameters

- **x0** (*array*) – x centers of dataset 0
- **y0** (*array*) – y centers of dataset 0
- **x1** (*array*) – x centers of dataset 1
- **y1** (*array*) – y centers of dataset 1

Returns Result of cost function

Return type array

13.10.3 py_eddy_tracker.poly.convex

py_eddy_tracker.poly.**convex**(*x*, *y*)

Check if polygon is convex

Parameters

- **x** (*array[float]*) –
- **y** (*array[float]*) –

Returns True if convex

Return type bool

13.10.4 py_eddy_tracker.poly.convexs

py_eddy_tracker.poly.**convexs**(*x*, *y*)

Check if polygons are convex

Parameters

- **x** (*array[float]*) –
- **y** (*array[float]*) –

Returns True if convex

Return type array[bool]

13.10.5 py_eddy_tracker.poly.create_vertice

py_eddy_tracker.poly.**create_vertice**(*x*, *y*)

Return polygon vertice.

Parameters

- **x** (*array*) –
- **y** (*array*) –

Returns Return polygon vertice

Return type vertice

13.10.6 py_eddy_tracker.poly.create_vertice_from_2darray

py_eddy_tracker.poly.**create_vertice_from_2darray**(*x*, *y*, *index*)

Choose a polygon in x,y list and return vertice.

Parameters

- **x** (*array*) –
- **y** (*array*) –
- **index** (*int*) –

Returns Return the vertice of polygon

Return type vertice

13.10.7 py_eddy_tracker.poly.fit_circle

py_eddy_tracker.poly.**fit_circle**(*x*, *y*)

From a polygon, function will fit a circle.

Must be call with local coordinates (in m, to get a radius in m).

Parameters

- **x** (*array*) – x of polygon
- **y** (*array*) – y of polygon

Returns x0, y0, radius, shape_error

Return type (float,float,float,float)

13.10.8 py_eddy_tracker.poly.fit_circle_

py_eddy_tracker.poly.**fit_circle_**(*x*, *y*)

From a polygon, function will fit a circle.

Must be call with local coordinates (in m, to get a radius in m).

$$(x_i - x_0)^2 + (y_i - y_0)^2 = r^2$$

$$x_i^2 - 2x_i x_0 + x_0^2 + y_i^2 - 2y_i y_0 + y_0^2 = r^2$$

$$2x_0 x_i + 2y_0 y_i + r^2 - x_0^2 - y_0^2 = x_i^2 + y_i^2$$

we get this linear equation

$$aX + bY + c = Z$$

where :

$$a = 2x_0, b = 2y_0, c = r^2 - x_0^2 - y_0^2$$

$$X = x_i, Y = y_i, Z = x_i^2 + y_i^2$$

Solutions:

$$x_0 = a/2, y_0 = b/2, r = \sqrt{c + x_0^2 + y_0^2}$$

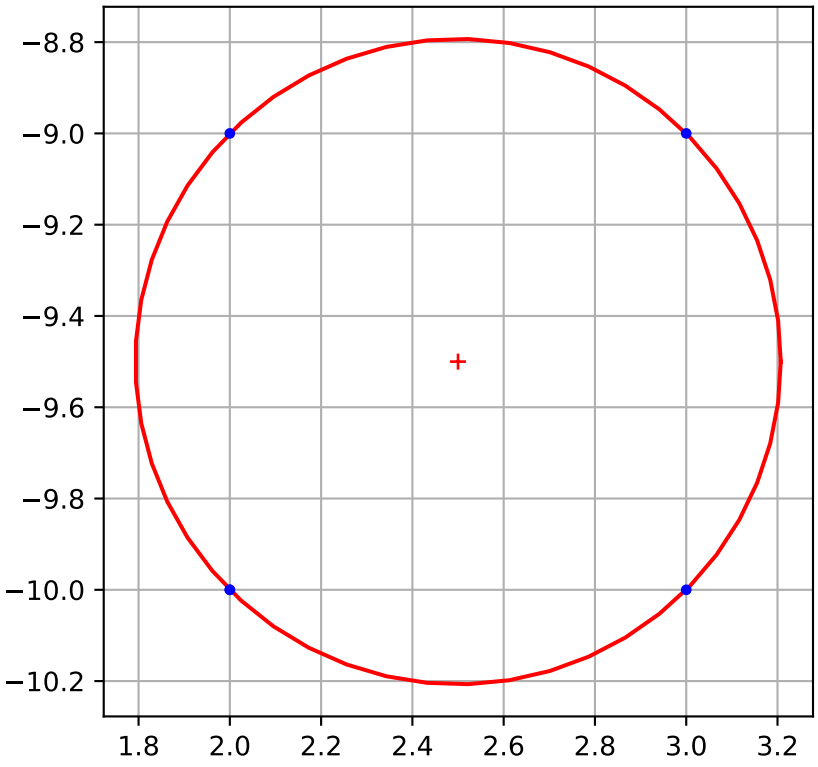
Parameters

- **x** (*array*) – x of polygon
- **y** (*array*) – y of polygon

Returns x0, y0, radius, shape_error

Return type (float,float,float,float)

```
import matplotlib.pyplot as plt
import numpy as np
from py_eddy_tracker.poly import fit_circle_
from py_eddy_tracker.generic import build_circle

V = np.array(((2, 2, 3, 3, 2), (-10, -9, -9, -10, -10)), dtype="f4")
x0, y0, radius, err = fit_circle_(V[0], V[1])
ax = plt.subplot(111)
ax.set_aspect("equal")
ax.grid(True)
ax.plot(*build_circle(x0, y0, radius), "r")
ax.plot(x0, y0, "r+")
ax.plot(*V, "b.")
plt.show()
```


−8.8
−9.0
−9.2
−9.4
−9.6
−9.8
−10.0
−10.2
1.8
2.0
2.2
2.4
2.6
2.8
3.0
3.2

13.10.9 py_eddy_tracker.poly.get_convex_hull

py_eddy_tracker.poly.**get_convex_hull**(*x*, *y*)

Get convex polygon which enclosed current polygon

Work only if contour is describe anti-clockwise

Parameters

- **x** (*array[float]*) –
- **y** (*array[float]*) –

Returns a convex polygon

Return type array,array

13.10.10 py_eddy_tracker.poly.get_pixel_in_regular

py_eddy_tracker.poly.**get_pixel_in_regular**(*vertices*, *x_c*, *y_c*, *x_start*, *x_stop*, *y_start*, *y_stop*)

Get a pixel list of a regular grid contain in a contour.

Parameters

- **vertices** (*array_like*) – contour vertice (N,2)
- **x_c** (*array_like*) – longitude coordinate of grid
- **y_c** (*array_like*) – latitude coordinate of grid
- **x_start** (*int*) – west index of contour
- **y_start** (*int*) – east index of contour
- **x_stop** (*int*) – south index of contour
- **y_stop** (*int*) – north index of contour

13.10.11 py_eddy_tracker.poly.get_wrap_vertice

py_eddy_tracker.poly.**get_wrap_vertice**(*x0*, *y0*, *x1*, *y1*, *i*)

Return a vertice for each polygon and check that use same reference coordinates.

Parameters

- **x0** (*array*) – x for polygon list 0
- **y0** (*array*) – y for polygon list 0
- **x1** (*array*) – x for polygon list 1
- **y1** (*array*) – y for polygon list 1
- **i** (*int*) – index to use fot the 2 list

Returns return two compatible vertice

Return type (vertice, vertice)

13.10.12 py_eddy_tracker.poly.is_left

py_eddy_tracker.poly.**is_left**(*x_line_0: float*, *y_line_0: float*, *x_line_1: float*, *y_line_1: float*, *x_test: float*, *y_test: float*) → bool

Test if point is left of an infinit line.

http://geomalgorithms.com/a03-_inclusion.html See: Algorithm 1 "Area of Triangles and Polygons"

Parameters

- **x_line_0** (*float*) –
- **y_line_0** (*float*) –
- **x_line_1** (*float*) –
- **y_line_1** (*float*) –
- **x_test** (*float*) –
- **y_test** (*float*) –

Returns > 0 for P2 left of the line through P0 and P1 = 0 for P2 on the line < 0 for P2 right of the line

Return type bool

13.10.13 py_eddy_tracker.poly.merge

py_eddy_tracker.poly.**merge**(*x*, *y*)

Merge all polygon of the list

Parameters

- **x** (*array*) – 2D array for a list of polygon
- **y** (*array*) – 2D array for a list of polygon

Returns Polygons which enclosed all

Return type array, array

13.10.14 py_eddy_tracker.poly.poly_area

py_eddy_tracker.poly.**poly_area**(*x*, *y*)

Must be call with local coordinates (in m, to get an area in m^2).

Parameters

- **x** (*array*) –
- **y** (*array*) –

Returns area of polygon in coordinates unit

Return type float

13.10.15 py_eddy_tracker.poly.poly_area_vertice

py_eddy_tracker.poly.**poly_area_vertice**(*v*)

Compute area from vertice.

Parameters **v** (*vertice*) – polygon vertice

Returns area of polygon in coordinates unit

Return type float

13.10.16 py_eddy_tracker.poly.poly_contain_poly

py_eddy_tracker.poly.**poly_contain_poly**(*xy_poly_out*, *xy_poly_in*)

Check if poly_in is include in poly_out.

Parameters

- **xy_poly_out** (*vertice*) –
- **xy_poly_in** (*vertice*) –

Returns True if poly_in is in poly_out

Return type bool

13.10.17 py_eddy_tracker.poly.polygon_overlap

py_eddy_tracker.poly.**polygon_overlap**(*p0*, *p1*, *minimal_area=False*)

Return percent of overlap for each item.

Parameters

- **p0** (*list(Polygon)*) – List of polygon to compare with p1 list
- **p1** (*list(Polygon)*) – List of polygon to compare with p0 list
- **minimal_area** (*bool*) – If True, function will compute intersection/little polygon, else intersection/union

Returns Result of cost function

Return type array

13.10.18 py_eddy_tracker.poly.shape_error

py_eddy_tracker.poly.**shape_error**(*x*, *y*, *x0*, *y0*, *r*)

With a polygon(x,y) in local coordinates.

and circle properties(x0, y0, r), function compute a shape error:

$$ShapeError = \frac{Polygon_{area} + Circle_{area} - 2 * Intersection_{area}}{Circle_{area}} * 100$$

When error > 100, area of difference is bigger than circle area

Parameters

- **x** (*array*) – x of polygon
- **y** (*array*) – y of polygon

- **x0** (*float*) – x center of circle
- **y0** (*float*) – y center of circle
- **r** (*float*) – radius of circle

Returns shape error

Return type float

13.10.19 py_eddy_tracker.poly.tri_area2

py_eddy_tracker.poly.**tri_area2**(*x*, *y*, *i0*, *i1*, *i2*)

Double area of triangle

Parameters

- **x** (*array*) –
- **y** (*array*) –
- **i0** (*int*) – indice of first point
- **i1** (*int*) – indice of second point
- **i2** (*int*) – indice of third point

Returns area

Return type float

13.10.20 py_eddy_tracker.poly.vertice_overlap

py_eddy_tracker.poly.**vertice_overlap**(*x0*, *y0*, *x1*, *y1*, *minimal_area=False*)

Return percent of overlap for each item.

Parameters

- **x0** (*array*) – x for polygon list 0
- **y0** (*array*) – y for polygon list 0
- **x1** (*array*) – x for polygon list 1
- **y1** (*array*) – y for polygon list 1
- **minimal_area** (*bool*) – If True, function will compute intersection/little polygon, else intersection/union

Returns Result of cost function

Return type array

By default

$$Score = \frac{Intersection(P_0, P_1)_{area}}{Union(P_0, P_1)_{area}}$$

If minimal area:

$$Score = \frac{Intersection(P_0, P_1)_{area}}{min(P_{0area}, P_{1area})}$$

13.10.21 py_eddy_tracker.poly.visvalingam

py_eddy_tracker.poly.**visvalingam**(*x*, *y*, *nb_pt=18*)

Polygon simplification with visvalingam algorithm

Parameters

- **x** (*array*) –
- **y** (*array*) –
- **nb_pt** (*int*) – array size of out

Returns New (x, y) array

Return type array,array

13.10.22 py_eddy_tracker.poly.winding_number_grid_in_poly

py_eddy_tracker.poly.**winding_number_grid_in_poly**(*x_1d*, *y_1d*, *i_x0*, *i_x1*, *x_size*, *i_y0*, *xy_poly*)

Return index for each grid coordinates within contour.

http://geomalgorithms.com/a03-_inclusion.html

Parameters

- **x_1d** (*array*) – x of local grid
- **y_1d** (*array*) – y of local grid
- **i_x0** (*int*) – int to add at x index to have index in global grid
- **i_x1** (*int*) – last index in global grid
- **x_size** (*int*) – number of x in global grid
- **i_y0** (*int*) – int to add at y index to have index in global grid
- **xy_poly** (*vertice*) – vertices of polygon which must contain pixel

Returns Return index in xy_poly

Return type (int,int)

13.10.23 py_eddy_tracker.poly.winding_number_poly

py_eddy_tracker.poly.**winding_number_poly**(*x*, *y*, *xy_poly*)

Check if x,y is in poly.

Parameters

- **x** (*float*) – x to test
- **y** (*float*) – y to test
- **xy_poly** (*vertice*) – vertice of polygon

Returns wn == 0 if x,y is not in poly

Retype int

13.11 py_eddy_tracker.tracking

Class to store link between observations

Functions

index	

13.11.1 py_eddy_tracker.tracking.index

py_eddy_tracker.tracking.**index**(*ar*, *items*)

Classes

Correspondances	Object to store correspondances And run tracking

13.11.2 py_eddy_tracker.tracking.Correspondances

class py_eddy_tracker.tracking.**Correspondances**(*datasets*, *virtual=0*, *class_method=None*, *class_kw=None*, *previous_correspondance=None*)

Bases: list

Object to store correspondances And run tracking

Initiate tracking

Parameters

- **datasets** (*list(str)*) – A sorted list of filename which contains eddy observations to track
- **class_method** (*class*) – A class which tell how to track
- **class_kw** (*dict*) – keyword argument to setup class
- **previous_correspondance** (Correspondances) – A previous correspondance object if you want continue tracking

Methods

append	Append object to the end of the list.
clear	Remove all items from list.
copy	Return a shallow copy of the list.
count	Return number of occurrences of value.
extend	Extend list by appending elements from the iterable.
from_netcdf	
get_unused_data	Add in track object all the observations which aren't selected Returns: Unused Eddies

continues on next page

Table 60 – continued from previous page

id_generator	Generation id and incrementation
index	Return first index of value.
insert	Insert object before index.
load	
load_compatible	
load_state	
longer_than	Remove from correspondance table all association for shorter eddies than size_min
merge	Merge all the correspondance in one array with all fields
merge_correspondance	
pop	Remove and return item at index (default last).
prepare_merging	
recense_dead_id_to_extend	Recense dead id to extend in virtual observation
remove	Remove first occurrence of value.
reset_dataset_cache	
reverse	Reverse *IN PLACE*.
save	
shorter_than	Remove from correspondance table all association for longer eddies than size_max
sort	Stable sort *IN PLACE*.
store_correspondance	Storing correspondance in an array
swap_dataset	Swap to next dataset
to_netcdf	
track	Run tracking

Attributes

ID_DTYPE	
N_DTYPE	
UINT32_MAX	
VIRTUAL_DTYPE	
period	To rethink

ID_DTYPE = 'u4'

N_DTYPE = 'u2'

UINT32_MAX = 4294967295

VIRTUAL_DTYPE = 'u1'

append(**args*, ***kwargs*)
Append object to the end of the list.

classmethod from_netcdf(*handler*)

get_unused_data(*raw_data=False*)
Add in track object all the observations which aren't selected Returns: Unused Eddies

id_generator(*nb_id*)
Generation id and incrementation

classmethod load(*filename*)

load_compatible(*filename*)

load_state()

longer_than(*size_min*)
Remove from correspondance table all association for shorter eddies than size_min

merge(*until=- 1*, *raw_data=True*)
Merge all the correspondance in one array with all fields

merge_correspondance(*other*)

property period
To rethink

Returns: period coverage by obs

prepare_merging()

recense_dead_id_to_extend()
Recense dead id to extend in virtual observation

reset_dataset_cache()

save(*filename*, *dict_completion=None*)

shorter_than(*size_max*)
Remove from correspondance table all association for longer eddies than size_max

store_correspondance(*i_previous*, *i_current*, *nb_real_obs*, *association_cost*)
Storing correspondance in an array

swap_dataset(*dataset*, **args*, ***kwargs*)
Swap to next dataset

to_netcdf(*handler*)

track()
Run tracking

CHAPTER

FOURTEEN

CHANGELOG

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

14.1 [Unreleased]

14.2 [3.3.0] - 2020-12-03

14.2.1 Added

- Add an implementation of visvalingam algorithm to simplify polygons with low modification
- Add method to found close tracks in an other atlas
- Allow to give a x reference when we display grid to be able to change xlim
- Add option to EddyId to select data index like *–indexs time=5 depth=2*
- Add a method to merge several indexs type for eddy obs
- Get dataset variable like attribute, and lifetime/age are available for all observations
- Add **EddyInfos** application to get general information about eddies dataset
- Add method to inspect contour rejection (which are not in eddies)
- Grid interp could be “nearest” or “bilinear”

14.2.2 Changed

- Now to have object informations in plot label used python `format` style, several key are available :
  - “t0”
  - “t1”
  - “nb_obs”
  - “nb_tracks” (only for tracked eddies)

14.3 [3.2.0] - 2020-09-16

14.4 [3.1.0] - 2020-06-25

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z