py-eddy-tracker Documentation

Release 3.0

A. Delepoulle & E. Mason

Dec 18, 2020

Installation

CHAPTER 1

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 2

Py eddy tracker toolbox

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

CHAPTER 3

Eddy detection

3.1 Display identification

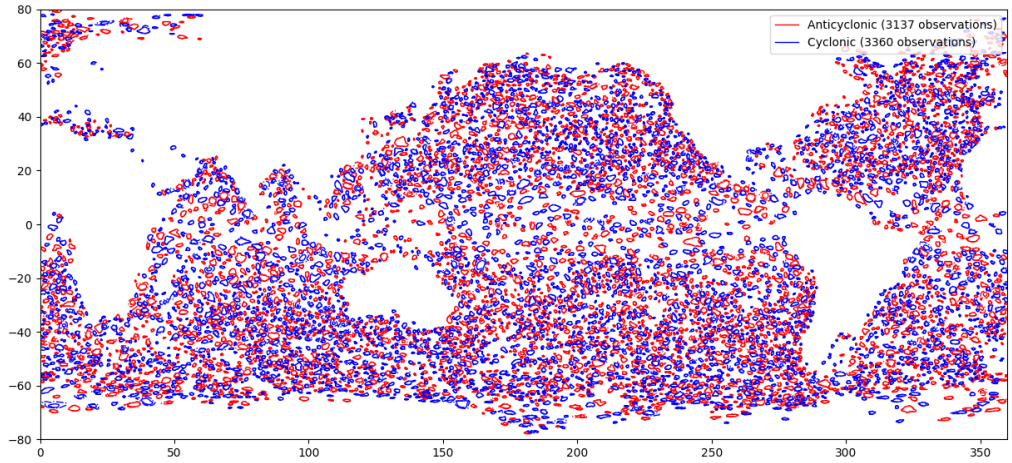
```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.observation import EddiesObservations
from py_eddy_tracker import data
```

Load detection files

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

Plot

```
fig = plt.figure(figsize=(15, 8))
ax = fig.add_subplot(111)
ax.set_aspect("equal")
ax.set_xlim(0, 360)
ax.set_ylim(-80, 80)
a.display(ax, label="Anticyclonic", color="r", lw=1)
c.display(ax, label="Cyclonic", color="b", lw=1)
ax.legend(loc="upper right")
```



Total running time of the script: (0 minutes 1.240 seconds)

3.2 Display contour & circle

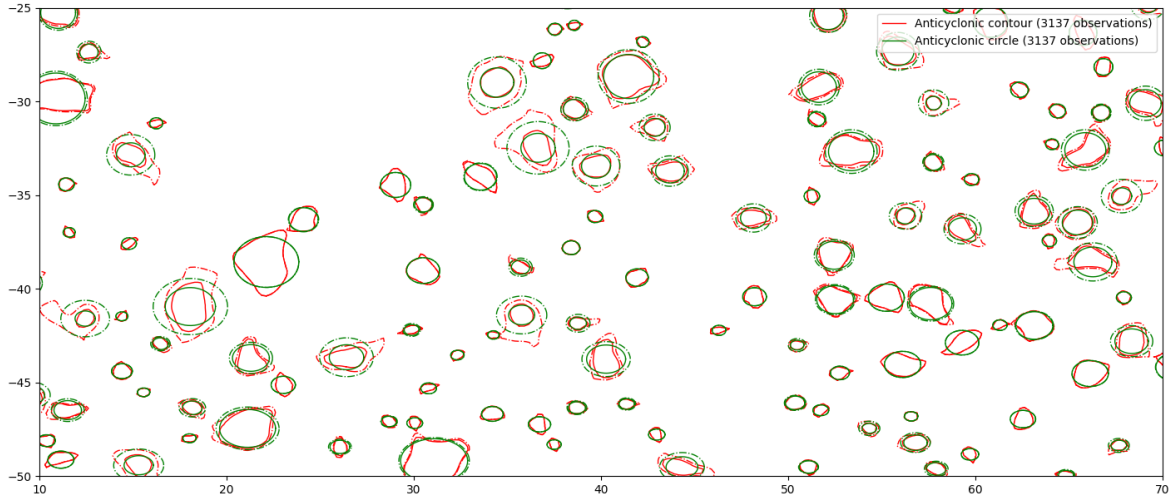
```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.observation import EddiesObservations
from py_eddy_tracker import data
```

Load detection files

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

Plot

```
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 contour by circle
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 1.014 seconds)

3.3 Shape error gallery

Gallery of contours with shape error

```
from matplotlib import pyplot as plt
from numpy import arange, radians, linspace, cos, sin
from py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker import data
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
```

(continues on next page)

(continued from previous page)

```
        if i not in contours:
            contours[i] = list()
        contours[i].append(current_contour)
```

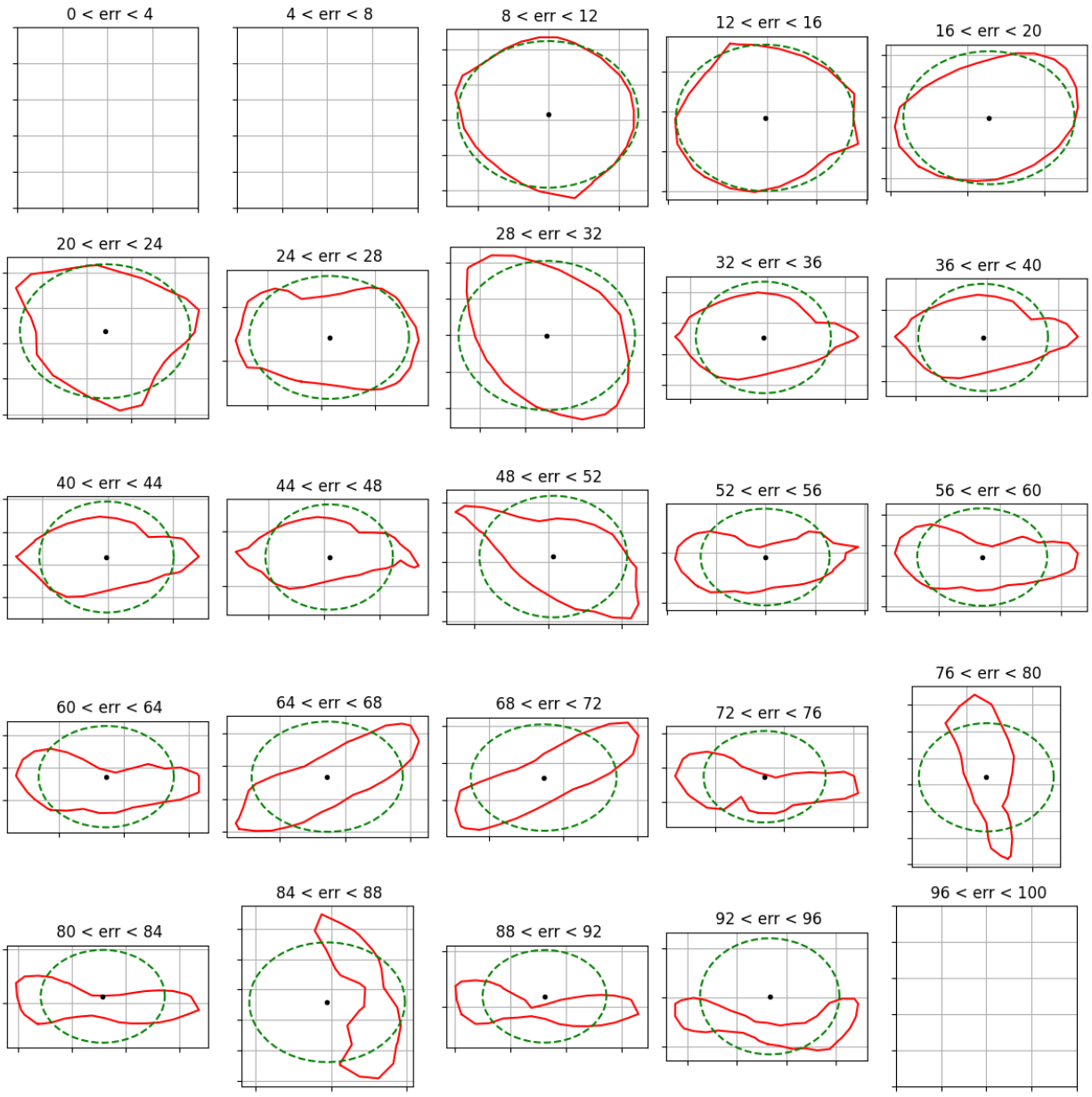
Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.2.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪2.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

3.3.1 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 11.987 seconds)

3.4 Eddy detection

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

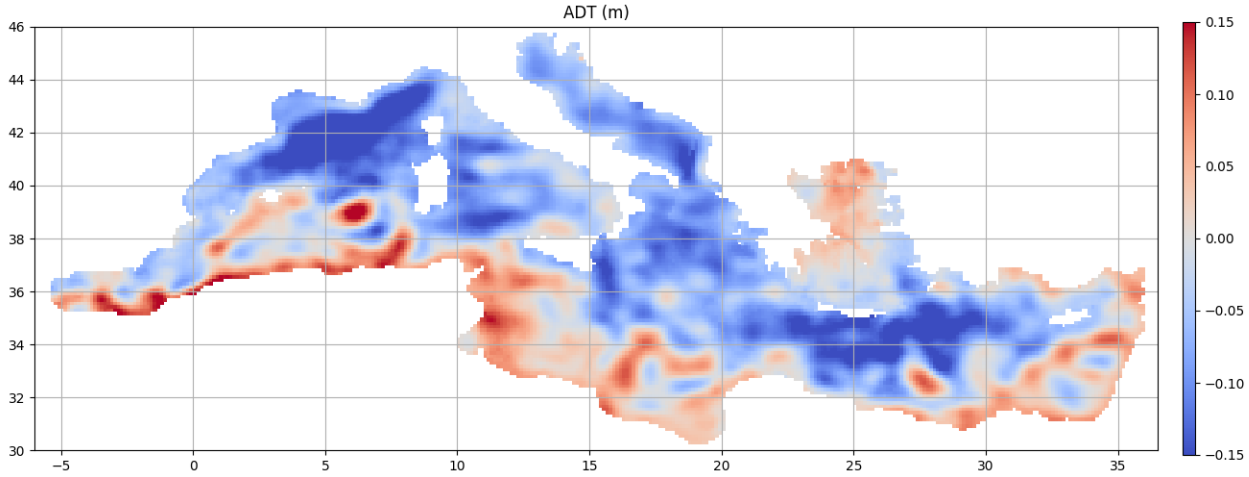
Figures will show different step to detect eddies.

```
from datetime import datetime
from matplotlib import pyplot as plt
from py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker import data
```

```
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(m, 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
↪"
)

ax = start_axes("ADT (m)")
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15)
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/v3.2.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪2.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

3.4.1 Get u/v

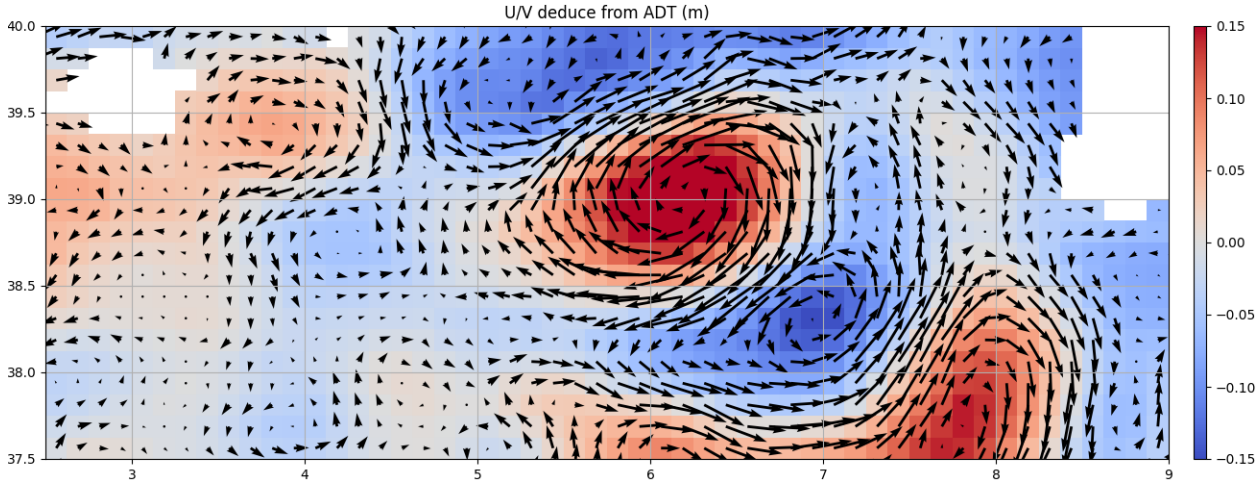
U/V are deduced from ADT, this algortihm are not usable around equator (~+- 2°)

```
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)
```

(continues on next page)

(continued from previous page)

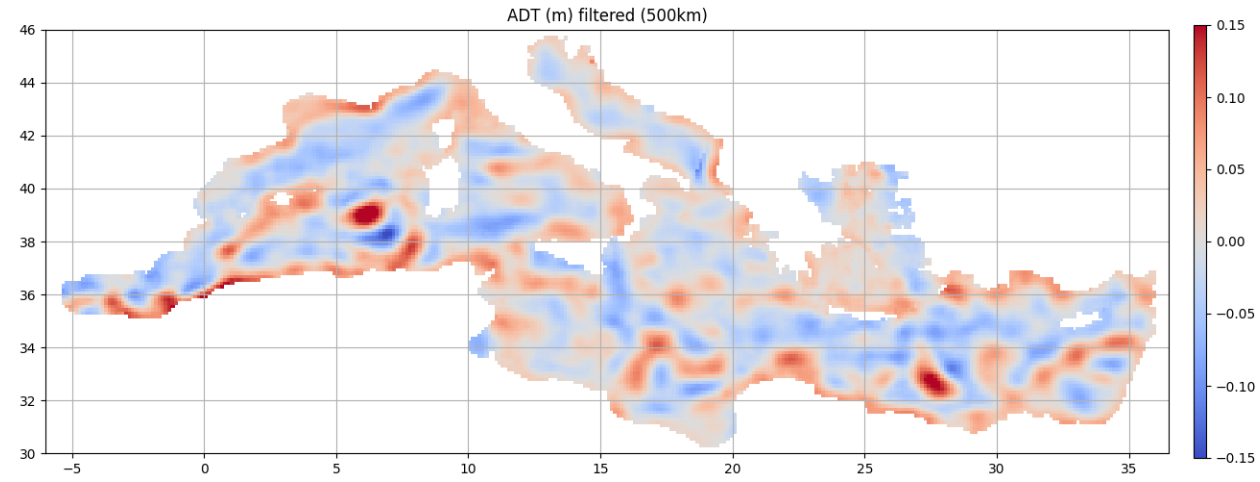
```
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15)
u, v = g.grid("u").T, g.grid("v").T
ax.quiver(g.x_c, g.y_c, u, v, scale=10)
update_axes(ax, m)
```



3.4.2 Pre-processings

Apply high filter to remove long scale to highlight 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)
update_axes(ax, m)
```



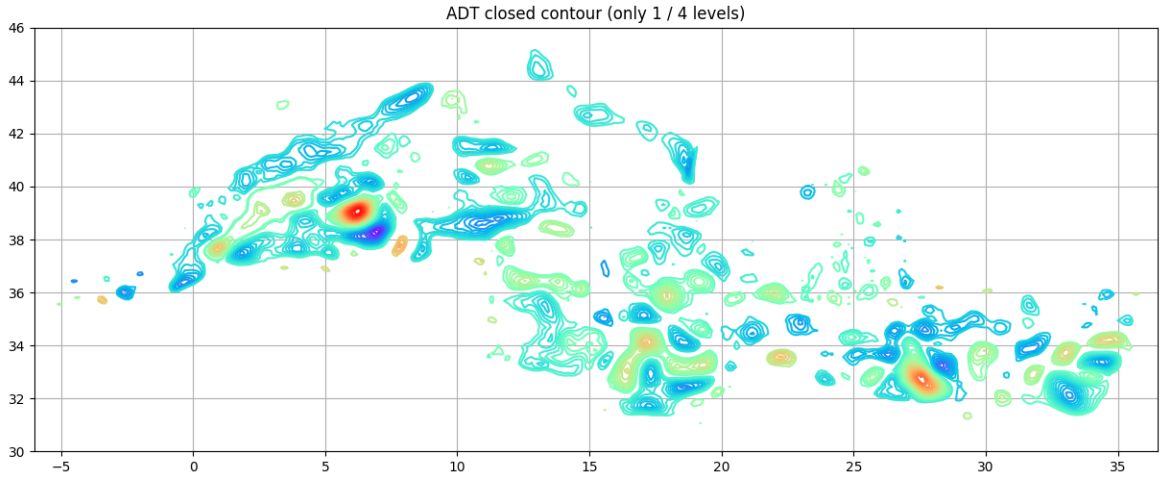
3.4.3 Identification

run identification with slice of 2 mm

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

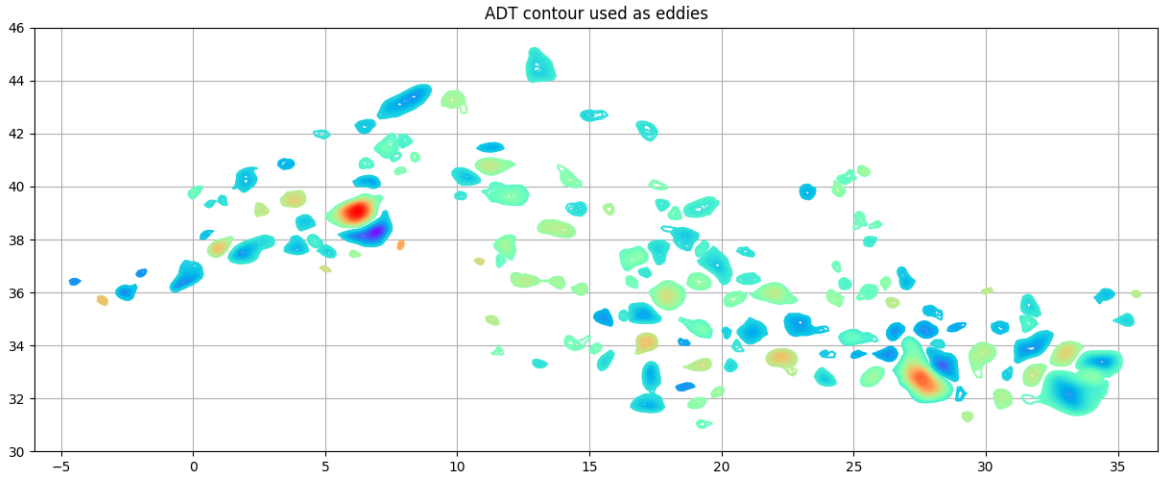
All closed contour found in this input grid (Display only 1 contour every 4)

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



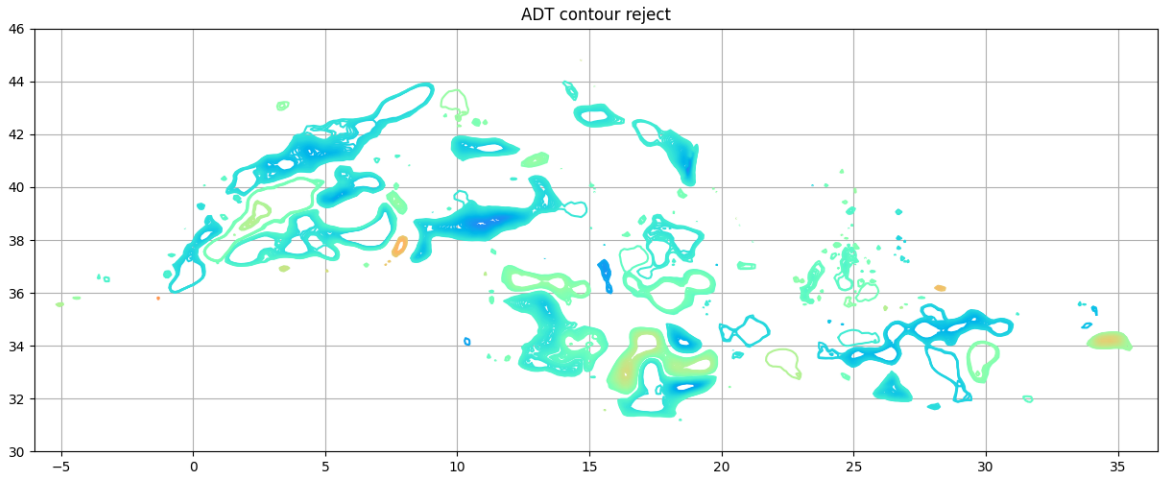
Contours include in eddies

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



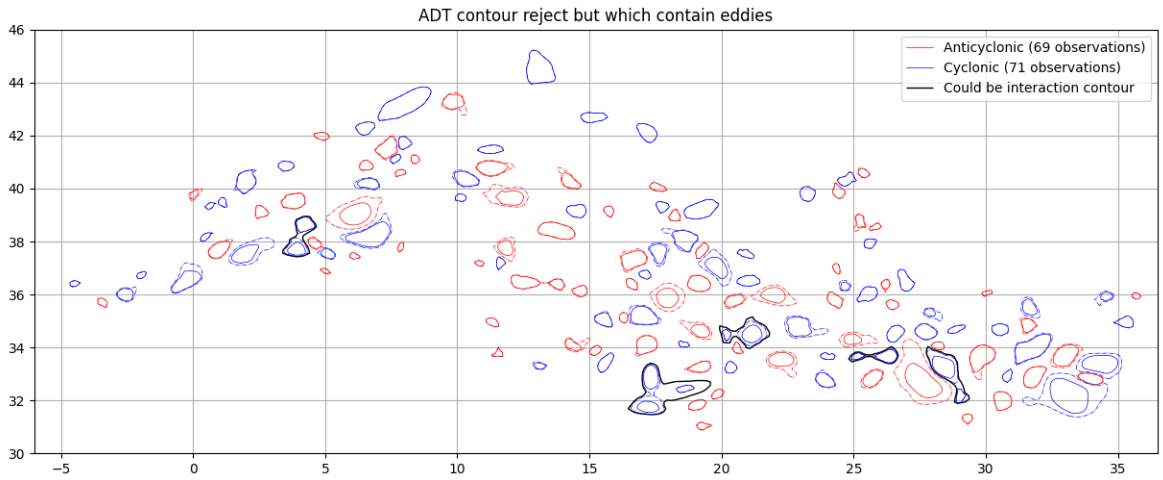
Contours reject from several origin (shape error to high, several extremum in contour, . . .)

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



Contours closed which contains several eddies

```
ax = start_axes("ADT contour reject but which contain 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 interaction contour
↪"
)
a.display(ax, color="r", linewidth=0.5, label="Anticyclonic", ref=-10)
c.display(ax, color="b", linewidth=0.5, label="Cyclonic", ref=-10)
ax.legend()
update_axes(ax)
```



3.4.4 Output

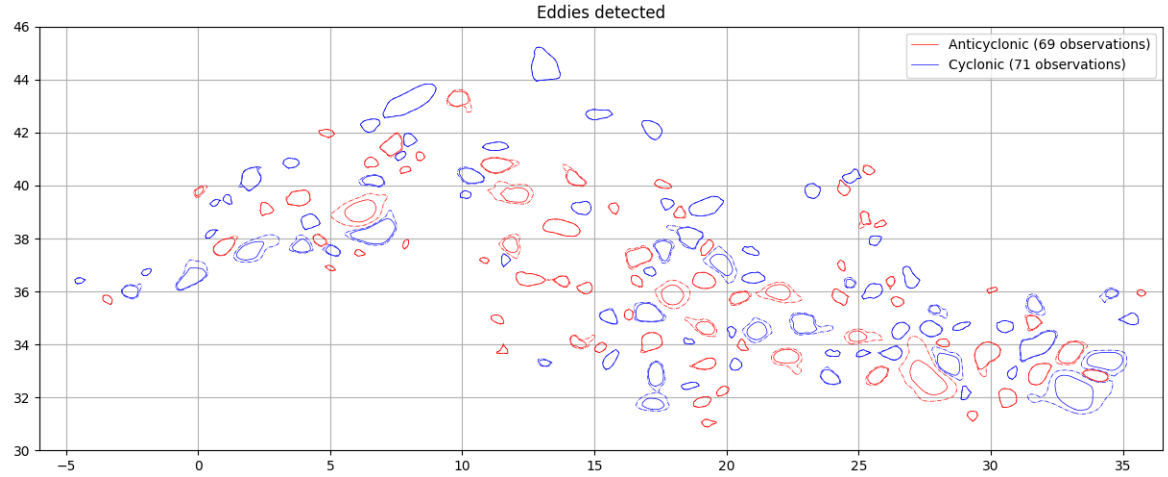
Display detected eddies, dashed lines represent effective contour and solid lines represent contour of maximum of speed. See figure 1 of https://doi.org/10.1175/JTECH-D-14-00019.1

```
ax = start_axes("Eddies detected")
a.display(ax, color="r", linewidth=0.5, label="Anticyclonic", ref=-10)
c.display(ax, color="b", linewidth=0.5, label="Cyclonic", ref=-10)
```

(continues on next page)

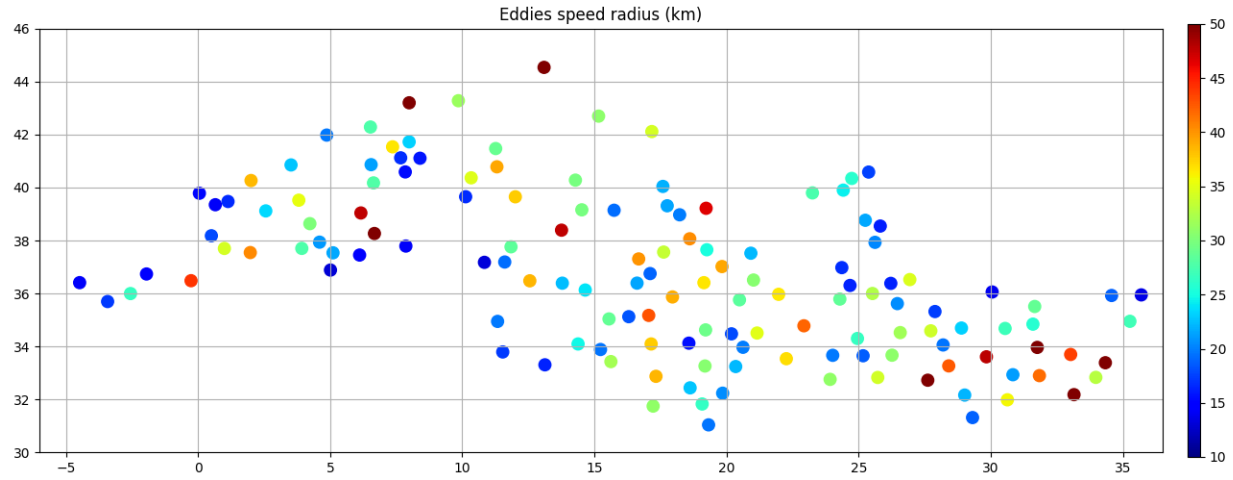
(continued from previous page)

```
ax.legend()
update_axes(ax)
```



Display speed radius of eddies detected

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



Total running time of the script: (0 minutes 17.397 seconds)

3.5 Eddy detection and filter

```
from datetime import datetime
from matplotlib import pyplot as plt
from py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker import data
from numpy import arange
```

```
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")
g.copy("adt", "adt_high")
wavelength = 400
g.bessel_high_filter("adt_high", wavelength)
date = datetime(2016, 5, 15)
```

Out:

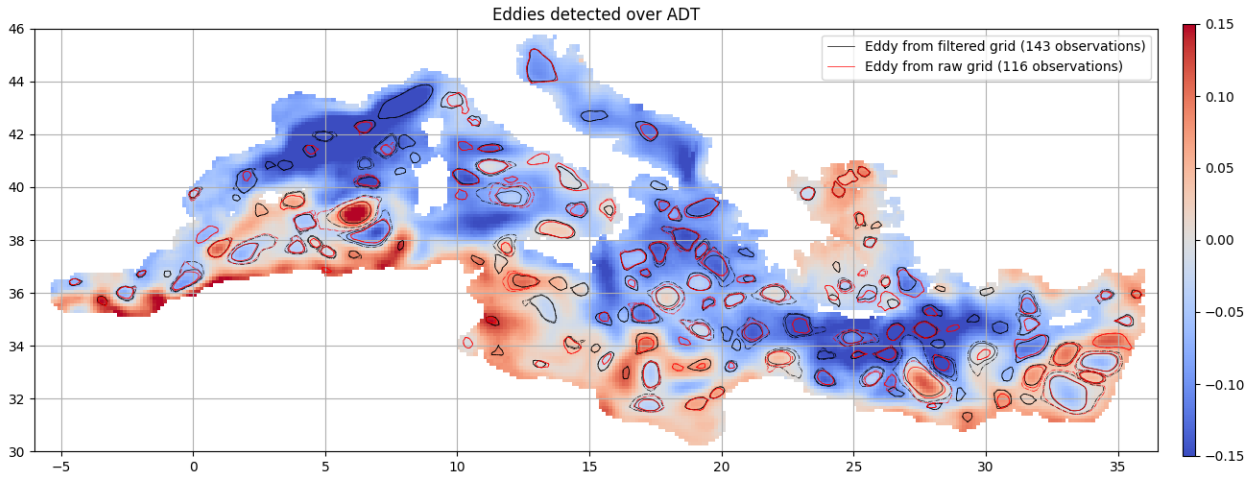
```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.2.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪2.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

Run algorithm of detection

```
a_f, c_f = g.eddy_identification("adt_high", "u", "v", date, 0.002)
merge_f = a_f.merge(c_f)
a_r, c_r = g.eddy_identification("adt", "u", "v", date, 0.002)
merge_r = a_r.merge(c_r)
```

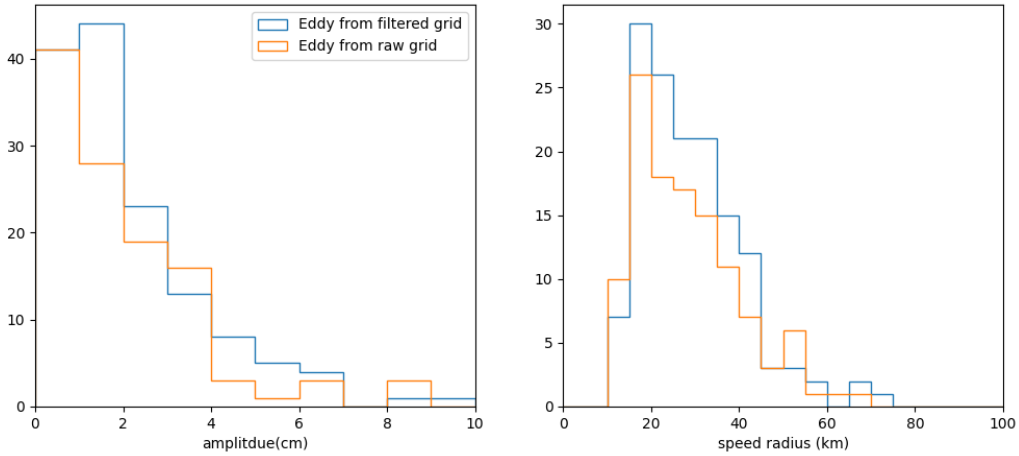
Display detection

```
ax = start_axes("Eddies detected over ADT")
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15)
merge_f.display(ax, lw=0.5, label="Eddy from filtered grid", ref=-10, color="k")
merge_r.display(ax, lw=0.5, label="Eddy from raw grid", ref=-10, color="r")
ax.legend()
update_axes(ax, m)
```



3.5.1 Parameters distribution

```
fig = plt.figure(figsize=(12, 5))
ax_a = plt.subplot(121, xlabel="amplitdue(cm)")
ax_r = plt.subplot(122, xlabel="speed radius (km)")
ax_a.hist(
    merge_f["amplitude"] * 100,
    bins=arange(0.0005, 100, 1),
    label="Eddy from filtered grid",
    histtype="step",
)
ax_a.hist(
    merge_r["amplitude"] * 100,
    bins=arange(0.0005, 100, 1),
    label="Eddy from raw grid",
    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_r["radius_s"] / 1000.0, bins=arange(0, 300, 5), histtype="step")
ax_r.set_xlim(0, 100)
ax_a.legend()
```

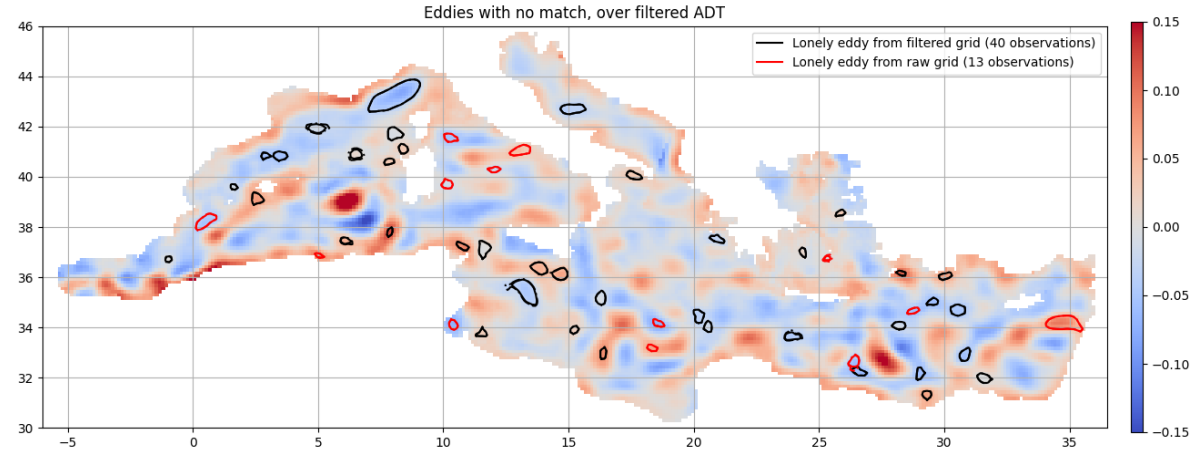


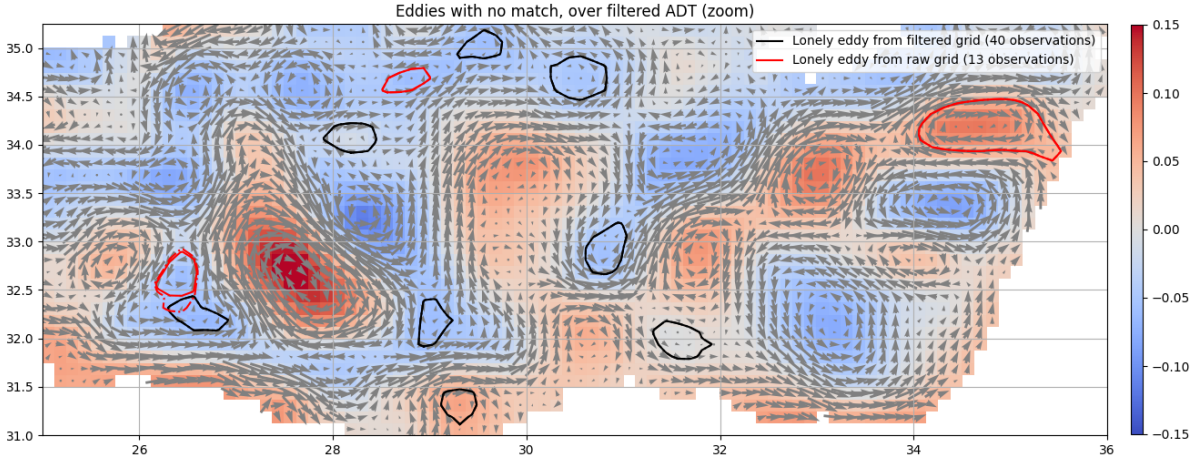
3.5.2 Match detection and compare

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

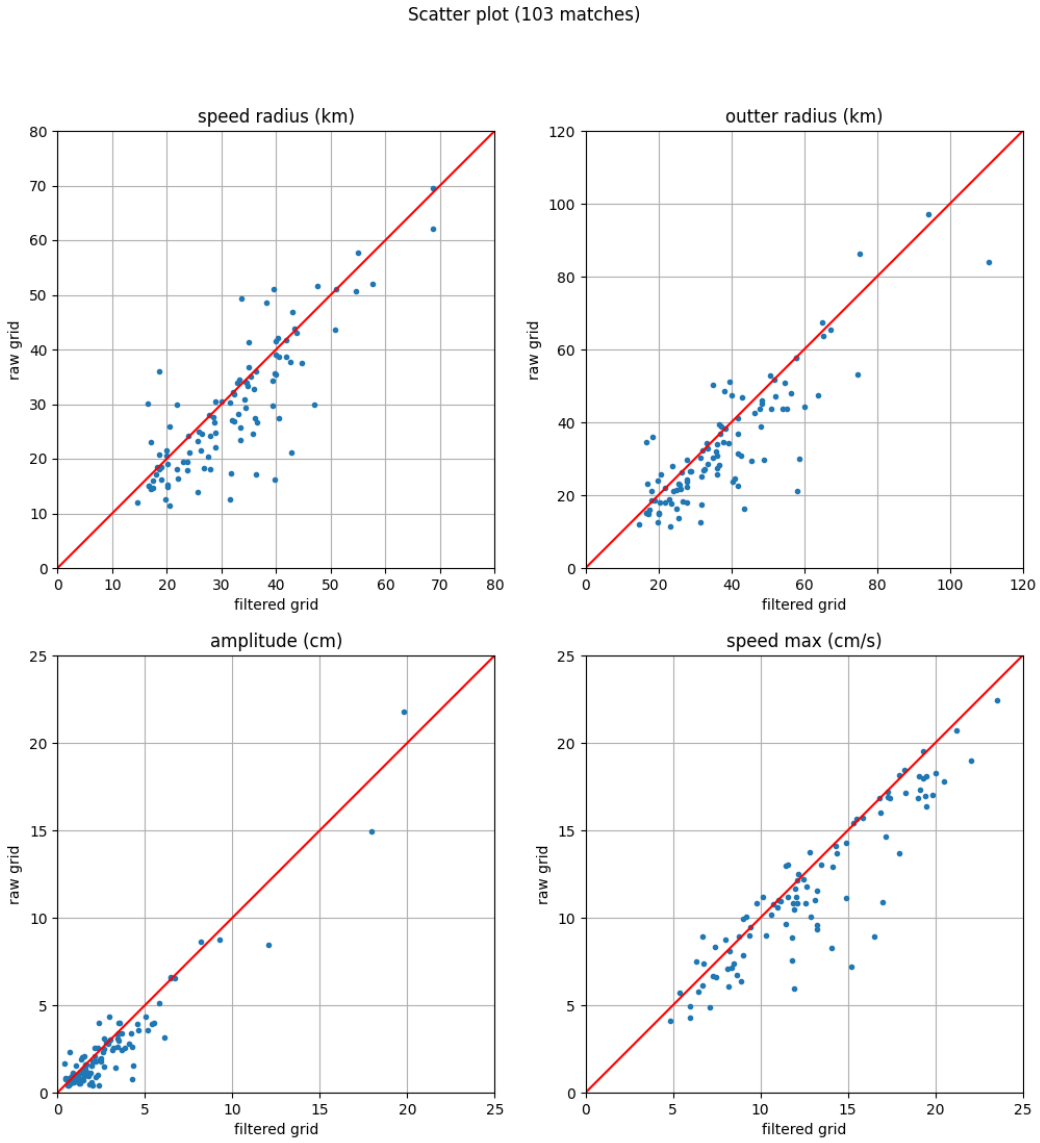
where is lonely eddies

```
kwargs_f = dict(lw=1.5, label="Lonely eddy from filtered grid", ref=-10, color="k")
kwargs_r = dict(lw=1.5, label="Lonely eddy from raw grid", 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_r.index(j_, reverse=True).display(ax, **kwargs_r)
ax.legend()
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_r.index(j_, reverse=True).display(ax, **kwargs_r)
ax.legend()
update_axes(ax, mappable)
```

- 

- 


```
fig = plt.figure(figsize=(12, 12))
fig.suptitle(f"Scatter plot ({i_.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, xlabel="filtered grid", ylabel="raw grid", title=label
    )
    ax.plot(merge_f[field][i_] * factor, merge_r[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)
```



Total running time of the script: (0 minutes 7.038 seconds)

3.6 Eddy detection on SLA and ADT

```
from datetime import datetime
from matplotlib import pyplot as plt
from py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker import data
```

```
def start_axes(title):
    fig = plt.figure(figsize=(13, 5))
    ax = fig.add_axes([0.03, 0.03, 0.90, 0.94])
```

(continues on next page)

(continued from previous page)

```
    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/v3.2.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪2.0-py3.7.egg/py_eddy_tracker/data/dt_med_allsat_phy_l4_20160515_20190101.nc
```

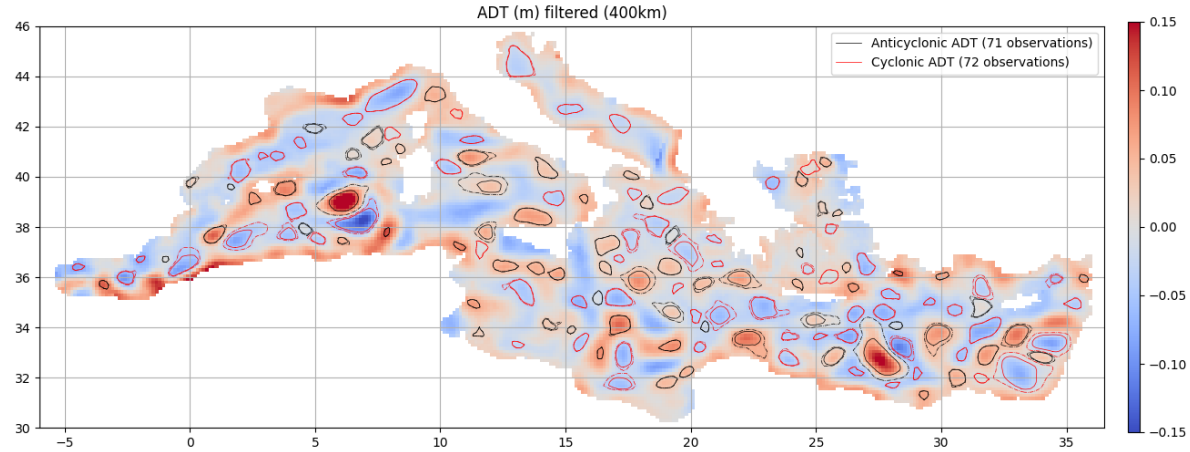
```
kwargs_a_adt = dict(lw=0.5, label="Anticyclonic ADT", ref=-10, color="k")
kwargs_c_adt = dict(lw=0.5, label="Cyclonic ADT", ref=-10, color="r")
kwargs_a_sla = dict(lw=0.5, label="Anticyclonic SLA", ref=-10, color="g")
kwargs_c_sla = dict(lw=0.5, label="Cyclonic SLA", ref=-10, color="b")
```

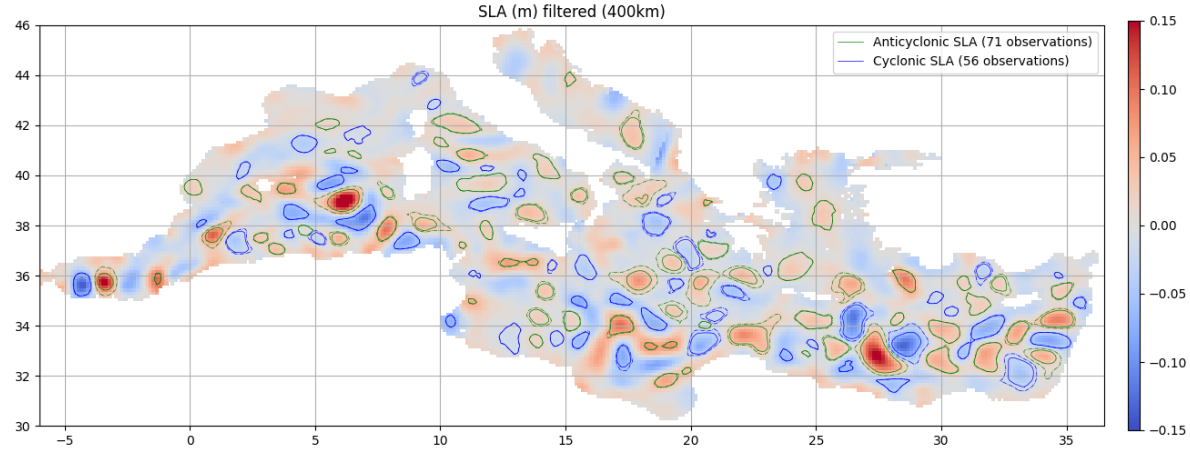
Run algorithm of detection

```
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

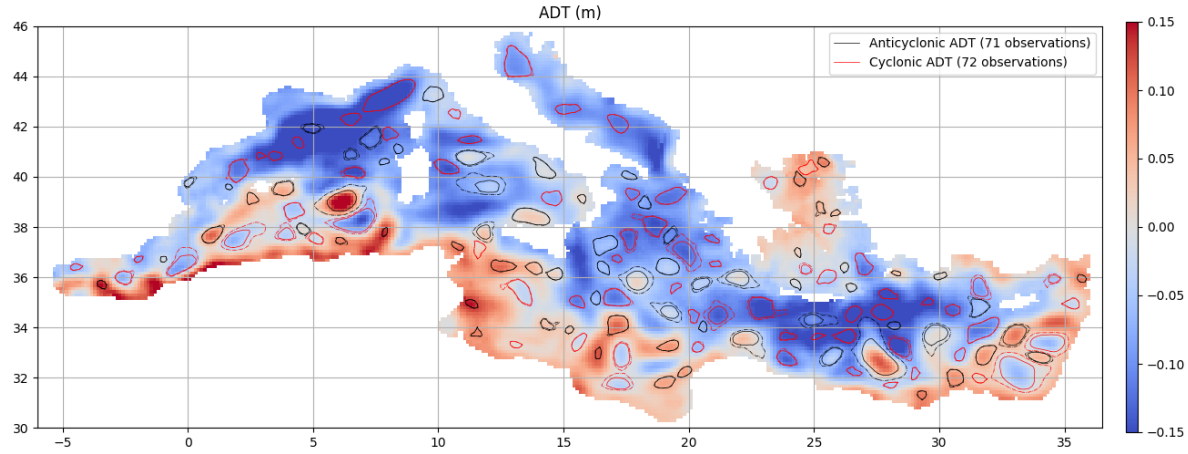
```
ax = start_axes(f"ADT (m) filtered ({wavelength}km)")
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15)
a_adt.display(ax, **kwargs_a_adt), c_adt.display(ax, **kwargs_c_adt)
ax.legend(), update_axes(ax, m)

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)
```

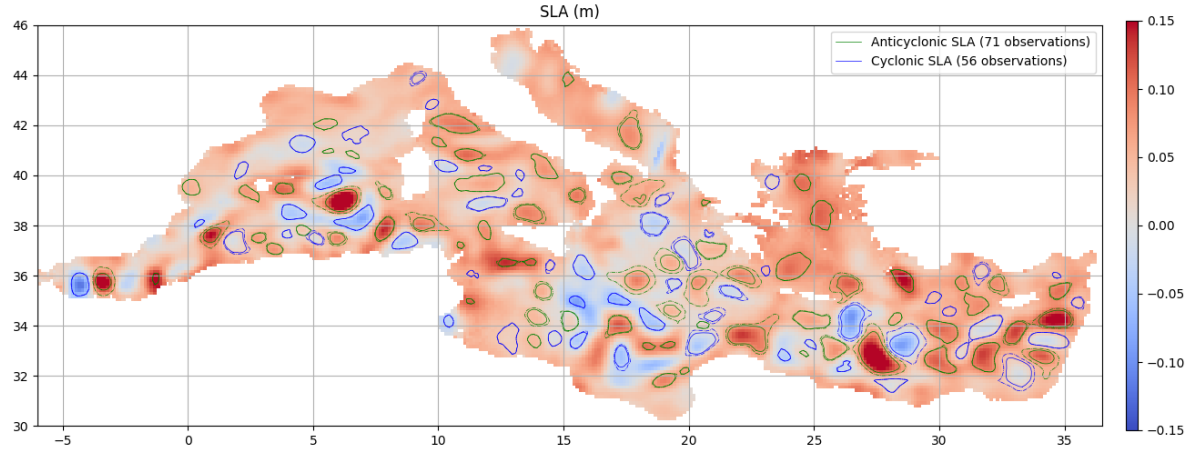
- 

- 

over raw

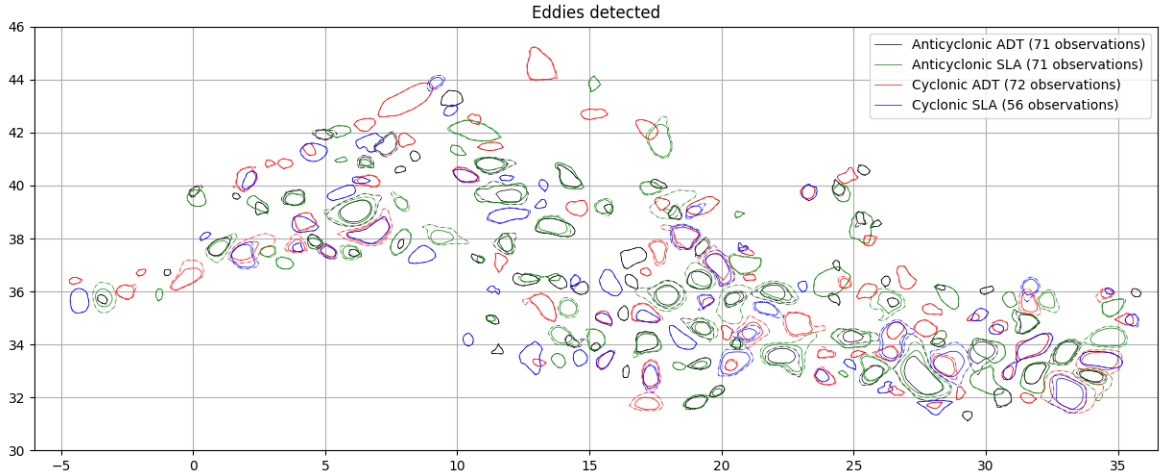
```
ax = start_axes("ADT (m)")
m = g.display(ax, "adt_raw", vmin=-0.15, vmax=0.15)
a_adt.display(ax, **kwargs_a_adt), c_adt.display(ax, **kwargs_c_adt)
ax.legend(), update_axes(ax, m)

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

- 

- 

Display detection

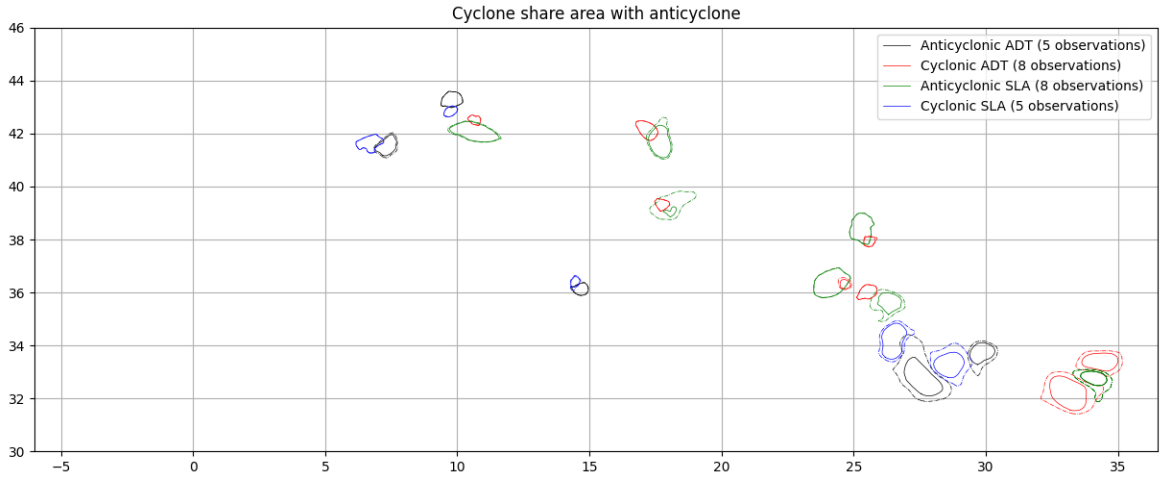
```
ax = start_axes("Eddies detected")
a_adt.display(ax, **kwargs_a_adt)
a_sla.display(ax, **kwargs_a_sla)
c_adt.display(ax, **kwargs_c_adt)
c_sla.display(ax, **kwargs_c_sla)
ax.legend()
update_axes(ax)
```



3.6.1 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)
```

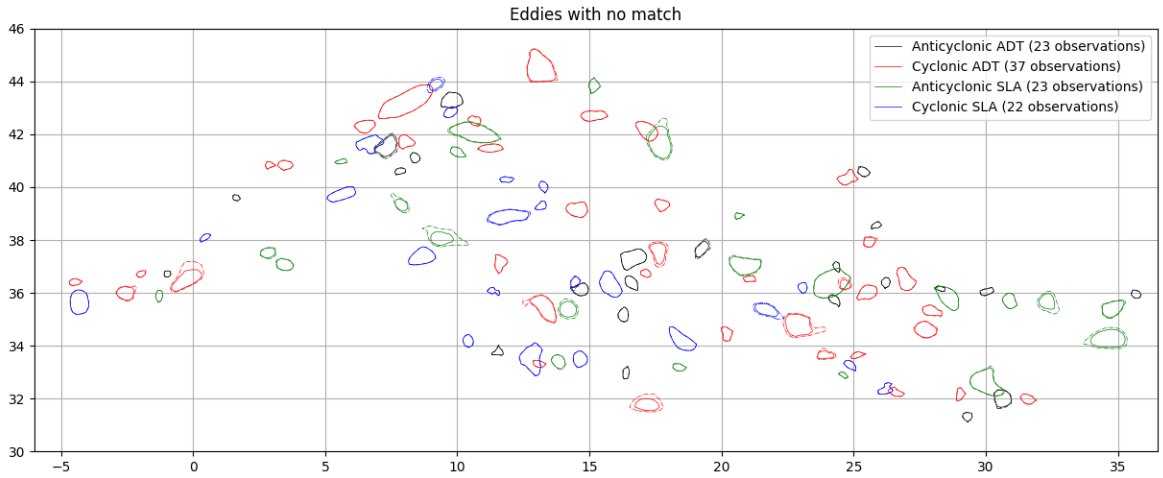


3.6.2 Scatter plot

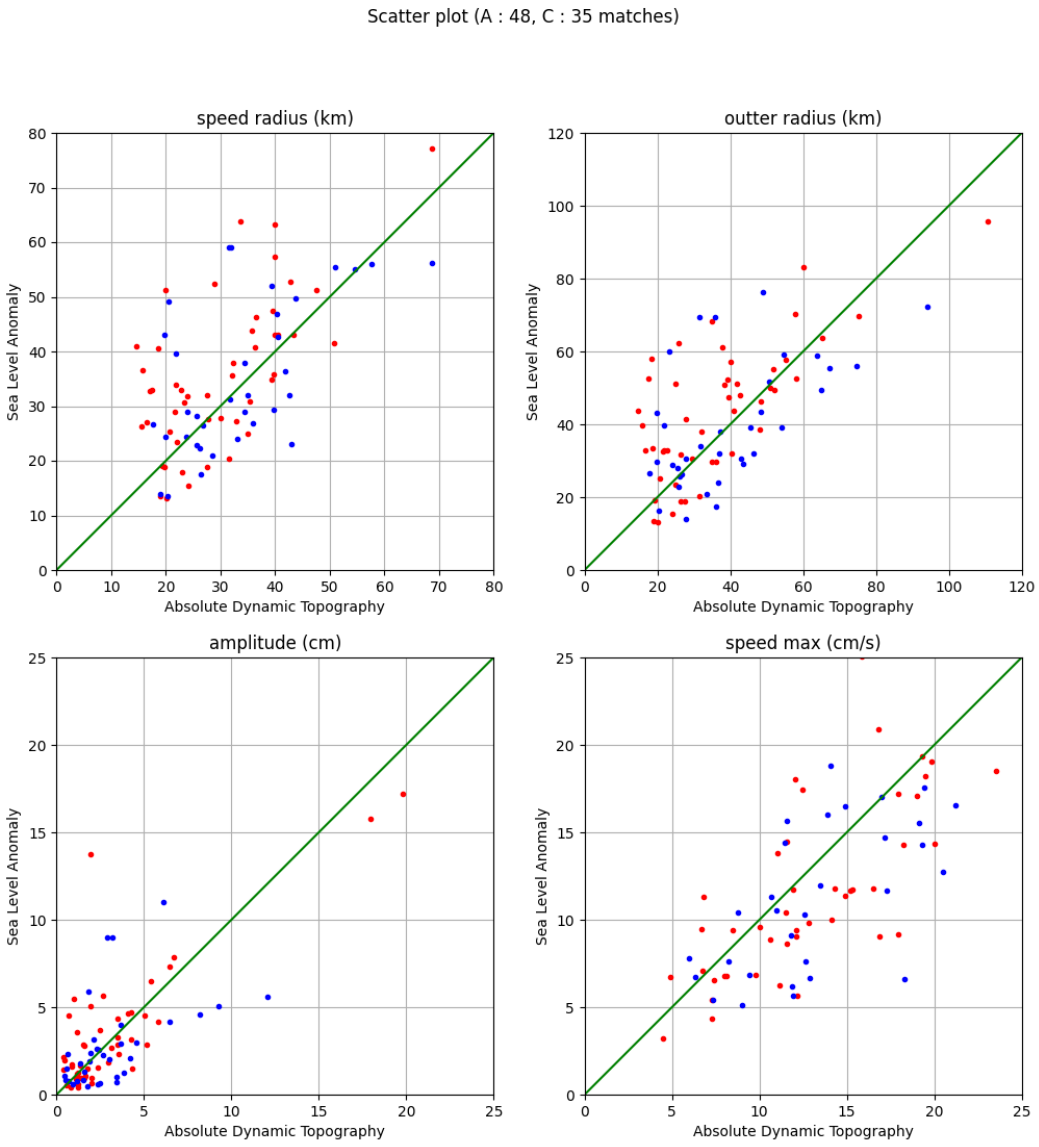
```
i_a_adt, i_a_sla, c = a_adt.match(a_sla, cmin=0.1)
i_c_adt, i_c_sla, c = c_adt.match(c_sla, cmin=0.1)
```

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.")
    ax.plot(c_adt[field][i_c_adt] * factor, c_sla[field][i_c_sla] * factor, "b.")
    ax.set_aspect("equal"), ax.grid()
    ax.plot((0, 1000), (0, 1000), "g")
    ax.set_xlim(0, stop), ax.set_ylim(0, stop)
```



Total running time of the script: (0 minutes 5.559 seconds)

CHAPTER 4

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

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/v3.2.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪2.0-py3.7.egg/py_eddy_tracker/data/nrt_global_allsat_phy_l4_20190223_20190226.nc
```

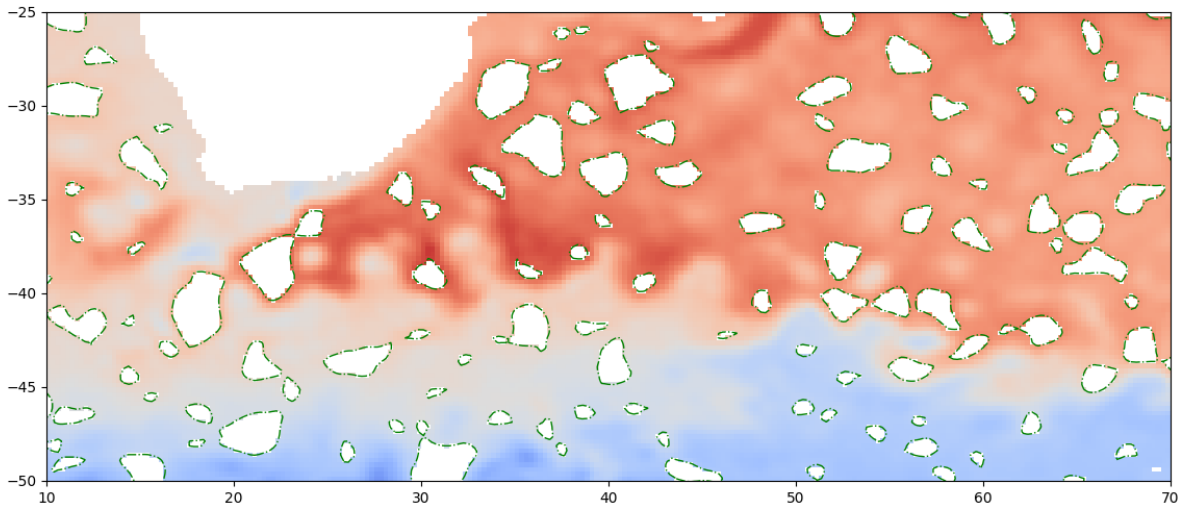
For each contours, we will get pixels indice in contour.

```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_axes((0.05, 0.05, 0.9, 0.9))
ax.set_aspect("equal")
```

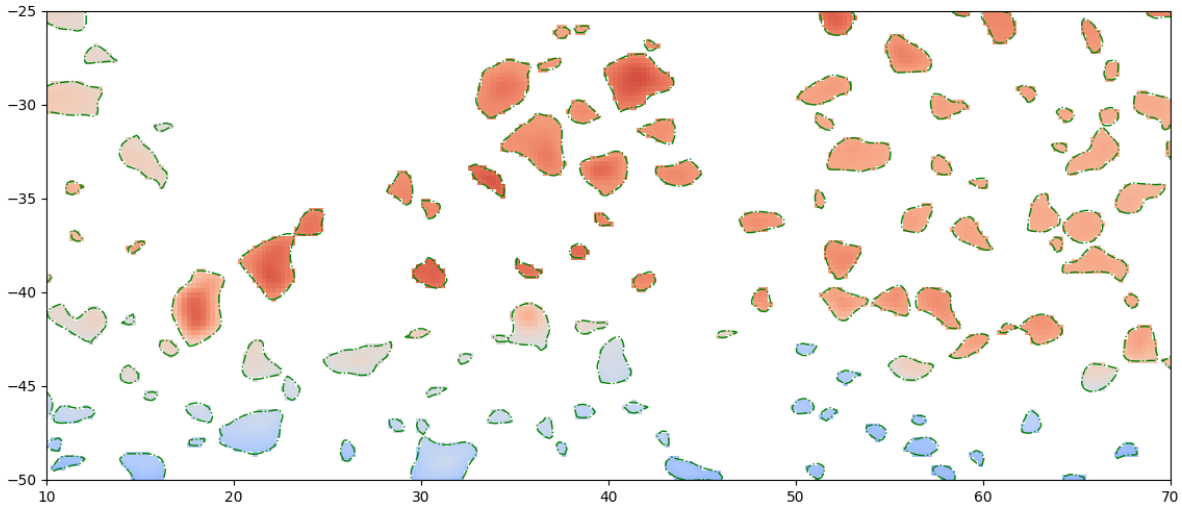
(continues on next page)

(continued from previous page)

```
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)
    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 1.222 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 py_eddy_tracker.dataset.grid import RegularGridDataset
from py_eddy_tracker import data
from matplotlib import pyplot as plt
from numpy import arange


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(m, 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
↪",
)
```

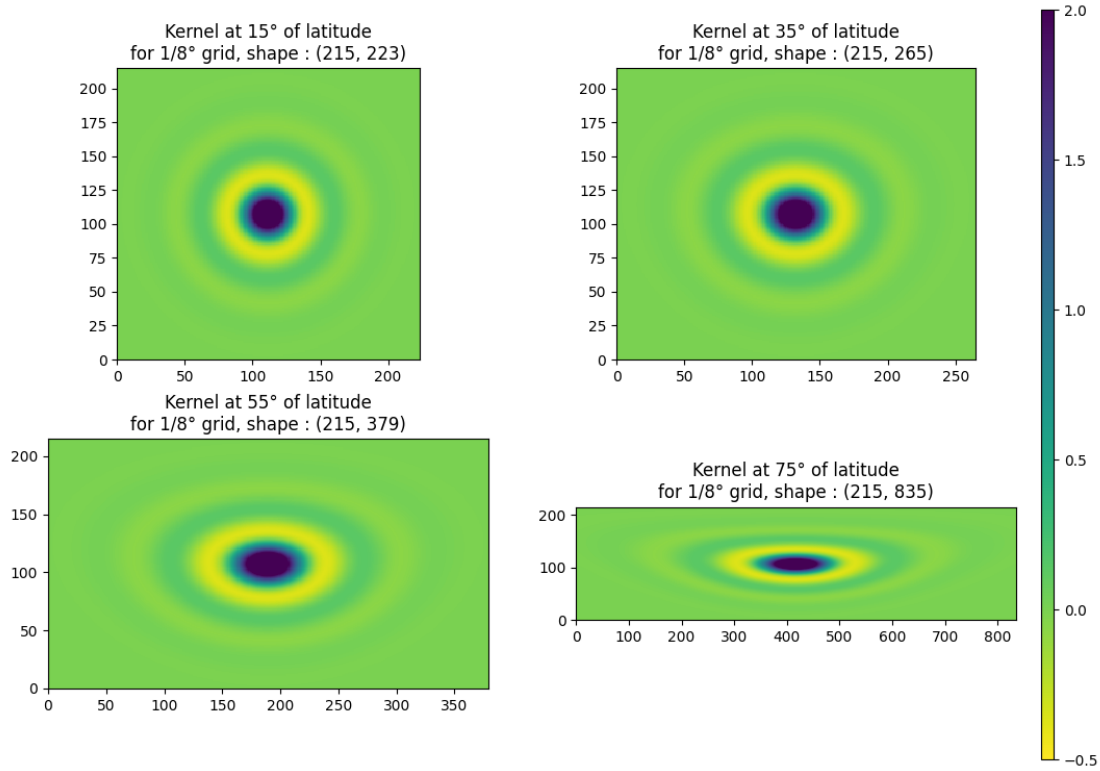
Out:

```
We assume pixel position of grid is center for /home/docs/checkouts/readthedocs.org/
↪user_builds/py-eddy-tracker/envs/v3.2.0/lib/python3.7/site-packages/pyEddyTracker-3.
↪2.0-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}",
        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)))
```



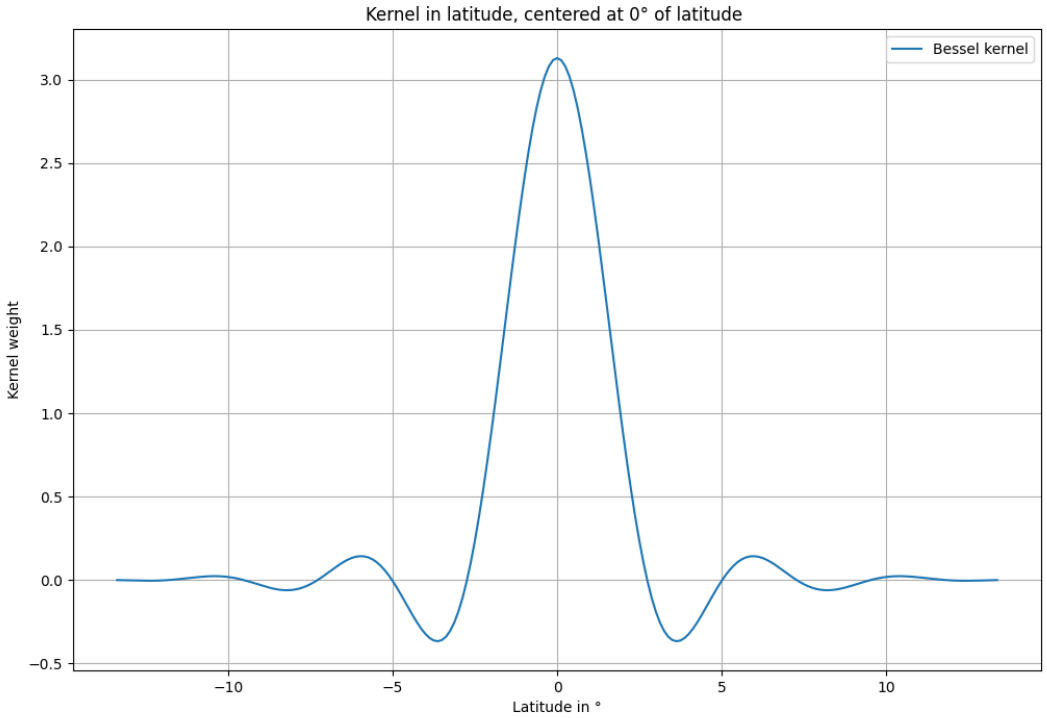
Kernel along latitude

```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(
```

(continues on next page)

(continued from previous page)

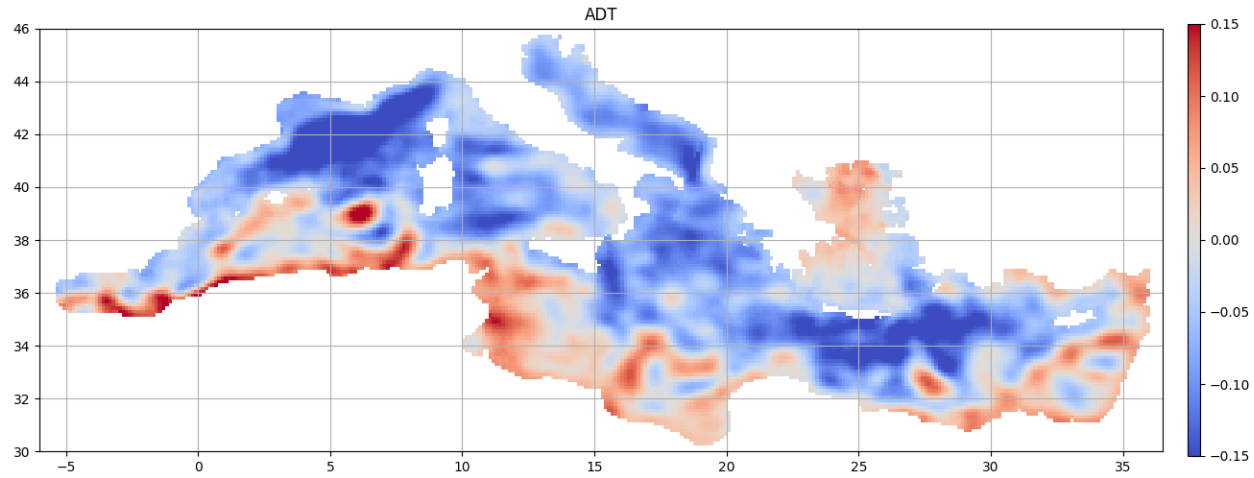
```
    111,
    ylabel="Kernel weight",
    xlabel="Latitude in °",
    title="Kernel in latitude, centered at 0° of latitude ",
)
k = g.kernel_bessel(0, 500, order=3)
k_lat = k[k.shape[0] // 2 + 1]
nb = k_lat.shape[0] // 2
ax.plot(
    arange(-nb * g.xstep, (nb + 0.5) * g.xstep, g.xstep), k_lat, label="Bessel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

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



4.2.2 Kernel applying

Original grid

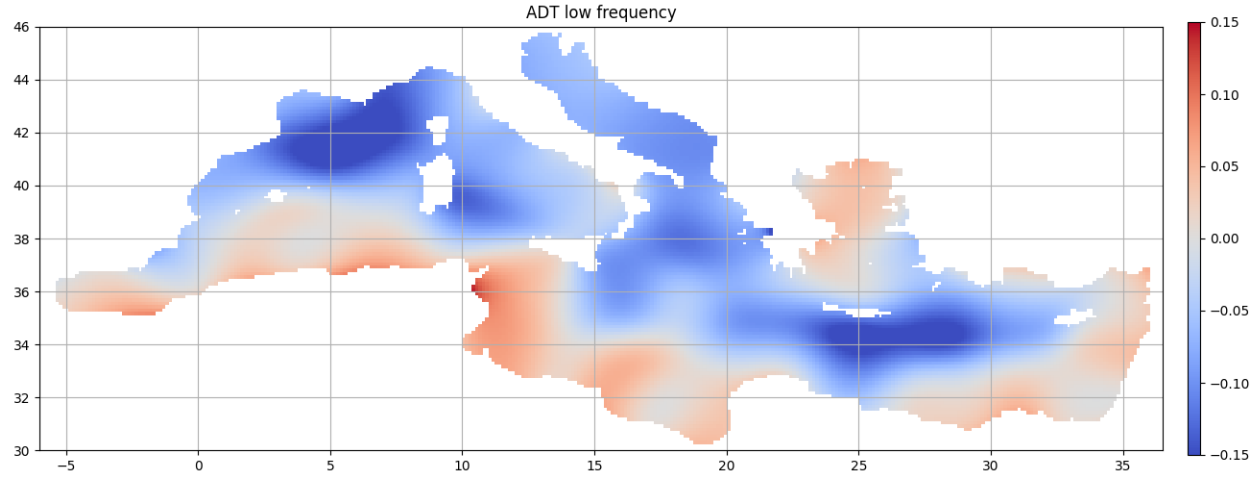
```
ax = start_axes("ADT")
m = g.display(ax, "adt", vmin=-0.15, vmax=0.15)
update_axes(ax, m)
```



We will select wavelength of 300 km

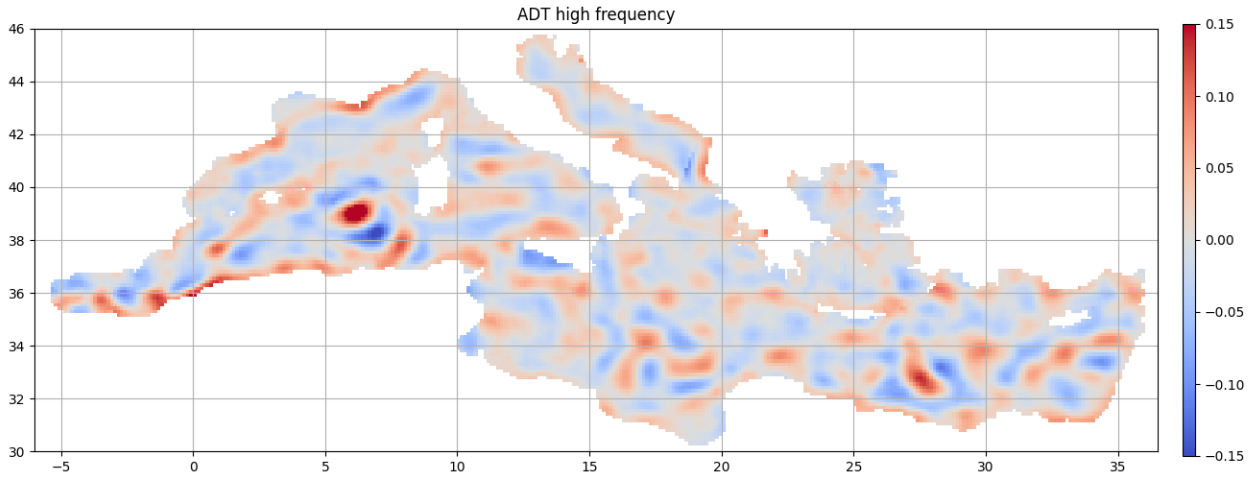
Low frequency

```
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)
```



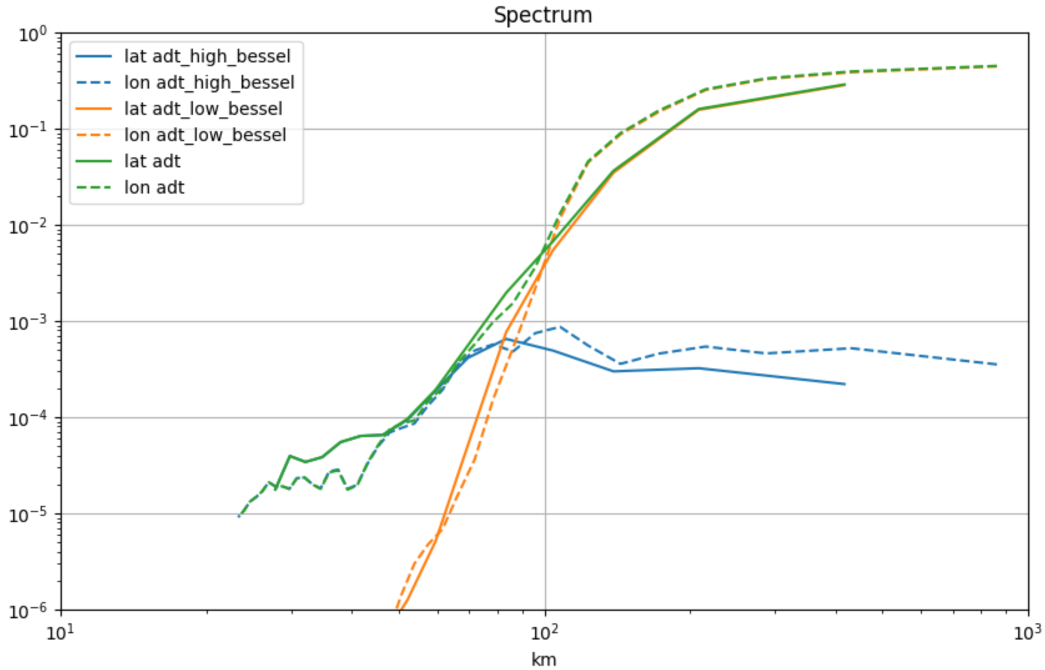
4.2.3 Clues

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/v3.2.0/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/v3.2.0/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/v3.2.0/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/v3.2.0/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/v3.2.0/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/v3.2.0/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))
```

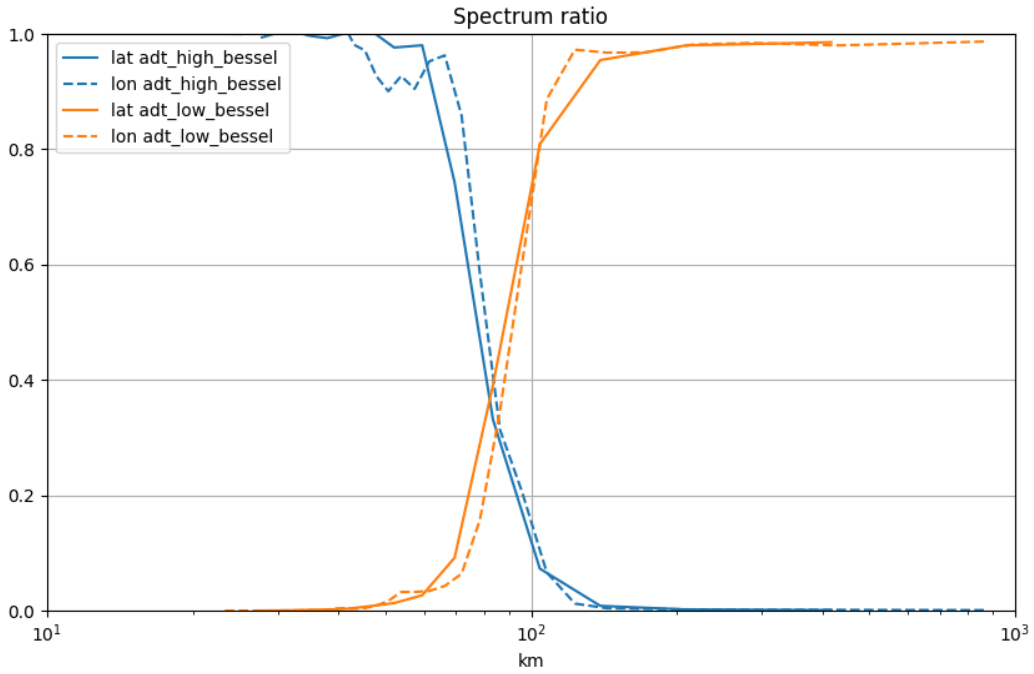
Spectrum ratio

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

(continues on next page)

(continued from previous page)

```
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/v3.2.0/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/v3.2.0/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/v3.2.0/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/v3.2.0/lib/
↪python3.7/site-packages/scipy/signal/spectral.py:1963: UserWarning: nperseg = 256␣
↪is greater than input length  = 74, using nperseg = 74
```

(continues on next page)

(continued from previous page)

```
  .format(nperseg, input_length))
```

4.2.4 Old filter

To do . . .

Total running time of the script: (0 minutes 6.347 seconds)

CHAPTER 5

Tracking Manipulation

5.1 Track animation

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

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

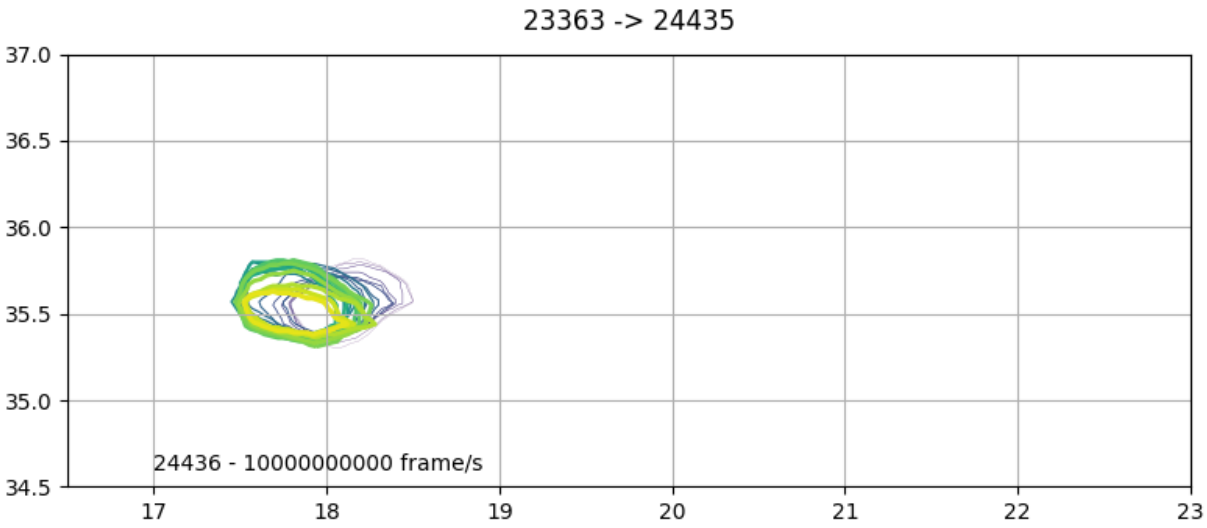
Load experimental atlas, and we select one eddy

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

Run animation 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 18.473 seconds)

5.2 Display fields

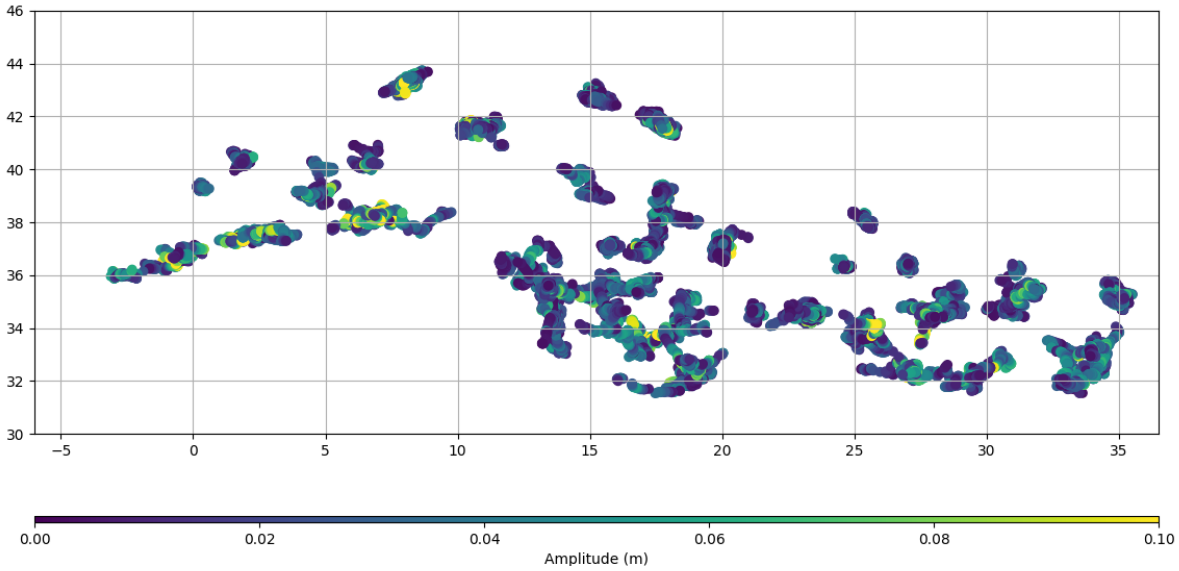
```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
import py_eddy_tracker_sample
```

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 2.135 seconds)

5.3 Display Tracks

```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
import py_eddy_tracker_sample
```

Load experimental atlas, and keep only eddies longer than 20 weeks

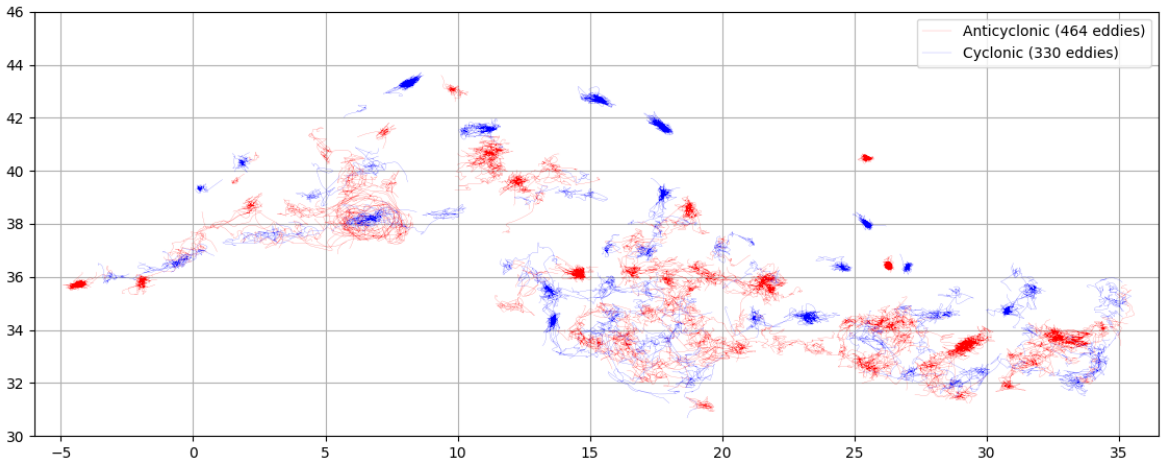
```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
c = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Cyclonic.zarr")
)
a = a.extract_with_length((7 * 20, -1))
c = c.extract_with_length((7 * 20, -1))
```

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", color="r", lw=0.1)
c.plot(ax, ref=-10, label="Cyclonic", color="b", lw=0.1)
ax.legend()
ax.grid()
```



Total running time of the script: (0 minutes 5.095 seconds)

5.4 Tracks which go through area

```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
import py_eddy_tracker_sample
```

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)
```

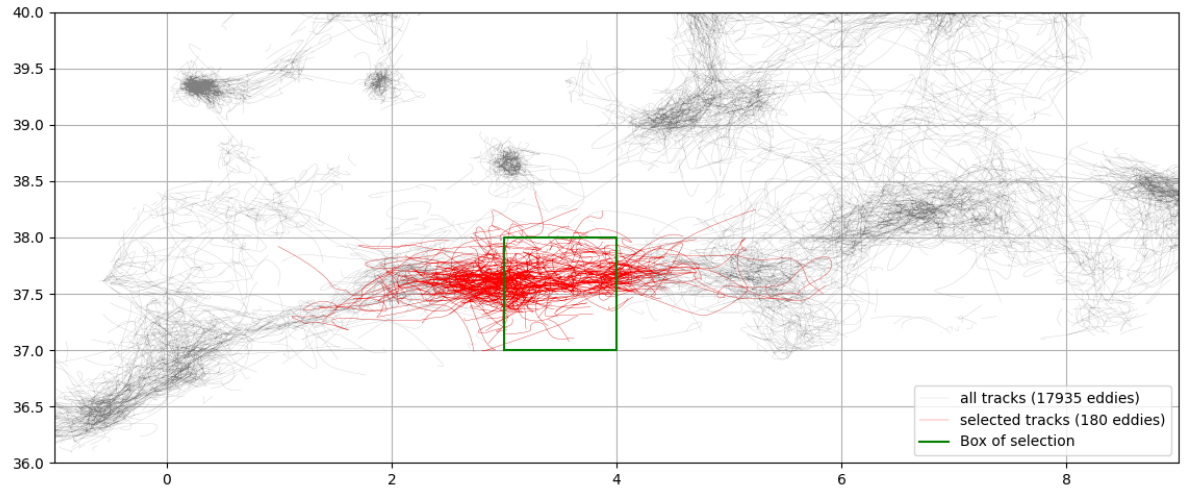
Plot

```
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")
c_subset.plot(ax, color="red", lw=0.2, ref=-10, label="selected tracks")
ax.plot(
    (x0, x0, x1, x1, x0,),
    (y0, y1, y1, y0, y0,),
    color="green",
    lw=1.5,
    label="Box of selection",
```

(continues on next page)

(continued from previous page)

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



Total running time of the script: (0 minutes 2.511 seconds)

5.5 One Track

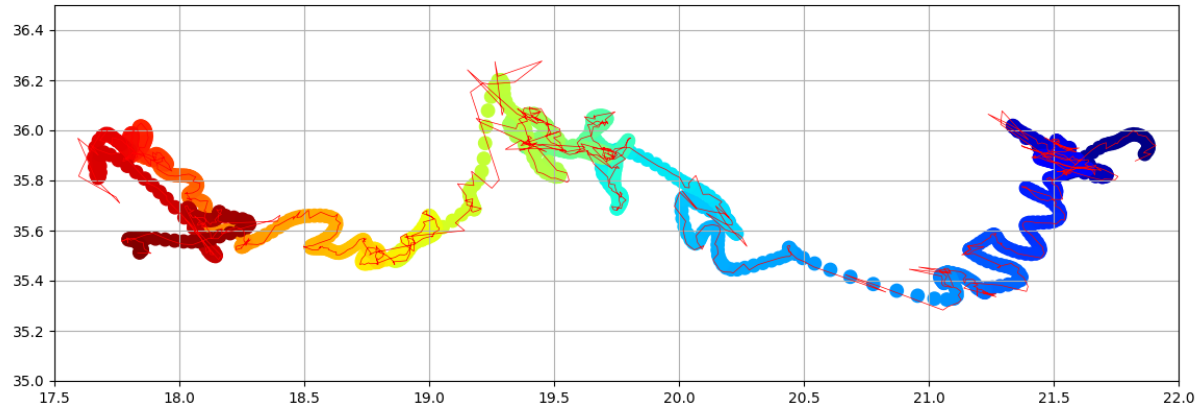
```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
import py_eddy_tracker_sample
```

Load experimental atlas, and we select one eddy

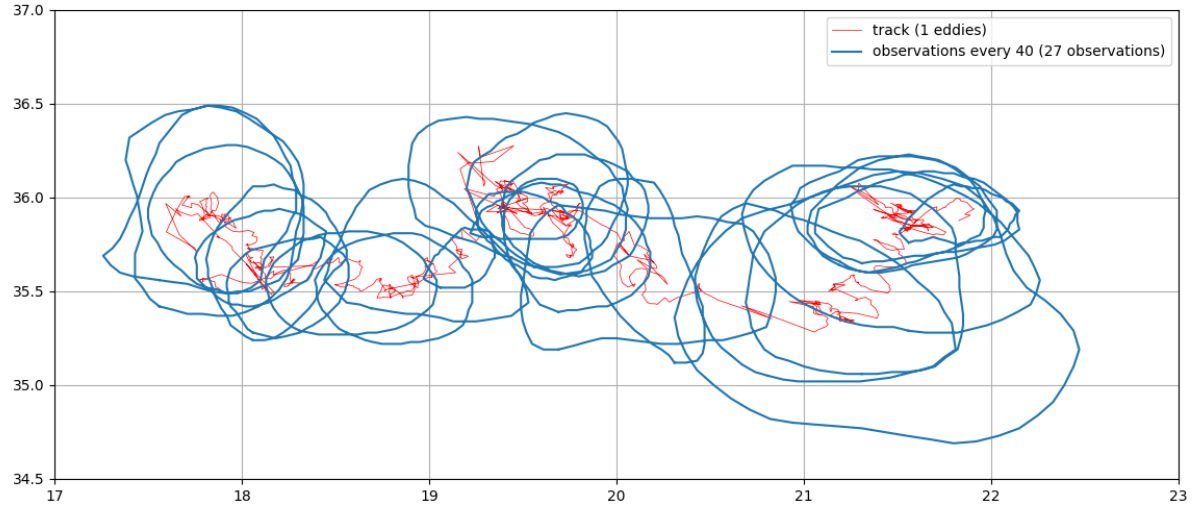
```
a = TrackEddiesObservations.load_file(
    py_eddy_tracker_sample.get_path("eddies_med_adt_allsat_dt2018/Anticyclonic.zarr")
)
eddy = a.extract_ids([9672])
eddy_f = a.extract_ids([9672])
eddy_f.position_filter(median_half_window=1, loess_half_window=5)
```

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"
)
ax.legend()
```



Total running time of the script: (0 minutes 1.275 seconds)

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

Get remote data, we will keep only 180 first days

```
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)
```

(continues on next page)

(continued from previous page)

```
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)
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)
```

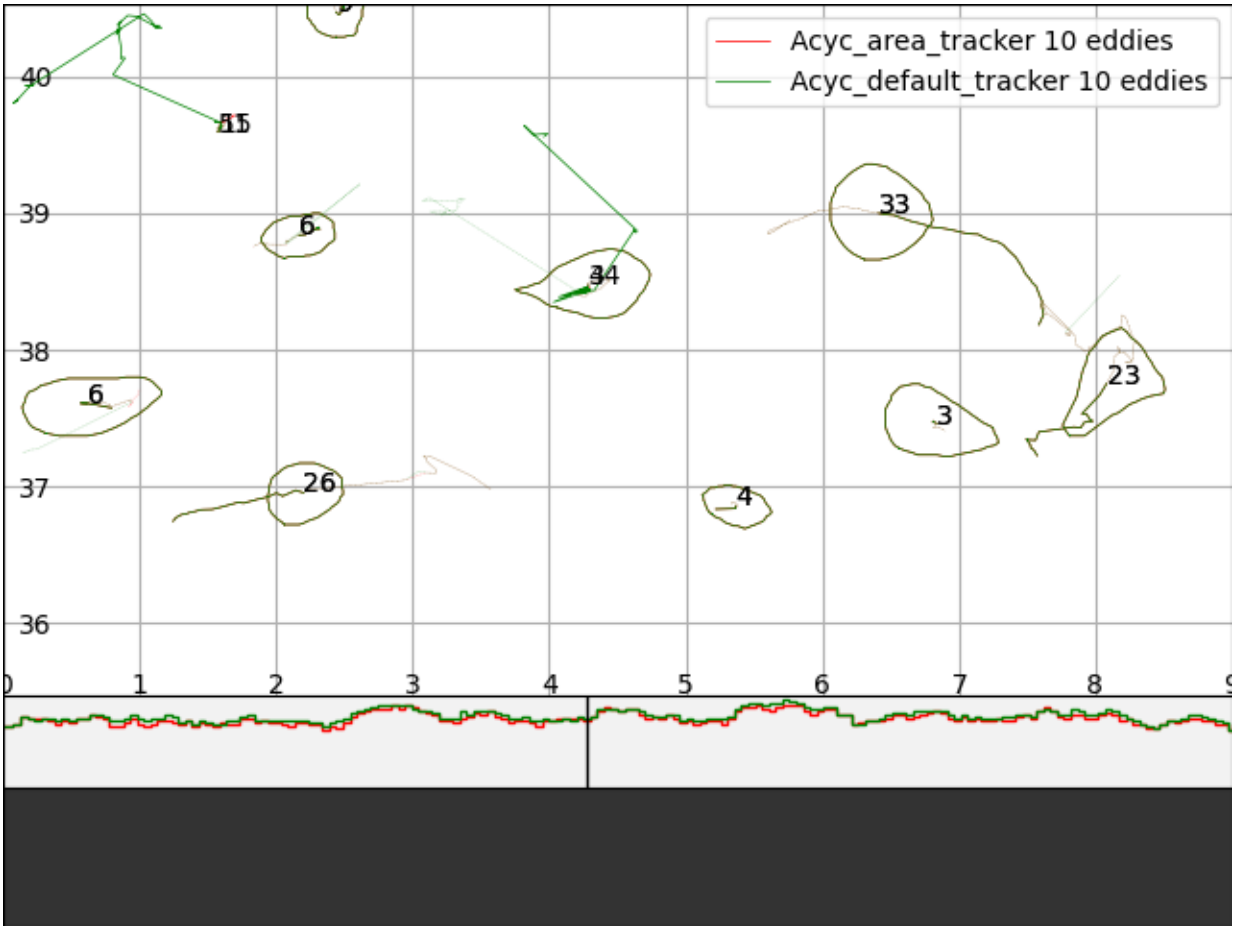
Start GUI to compare tracking

```
g = GUI(
    Acyc_area_tracker=eddies_area_tracker, Acyc_default_tracker=eddies_default_tracker
```

(continues on next page)

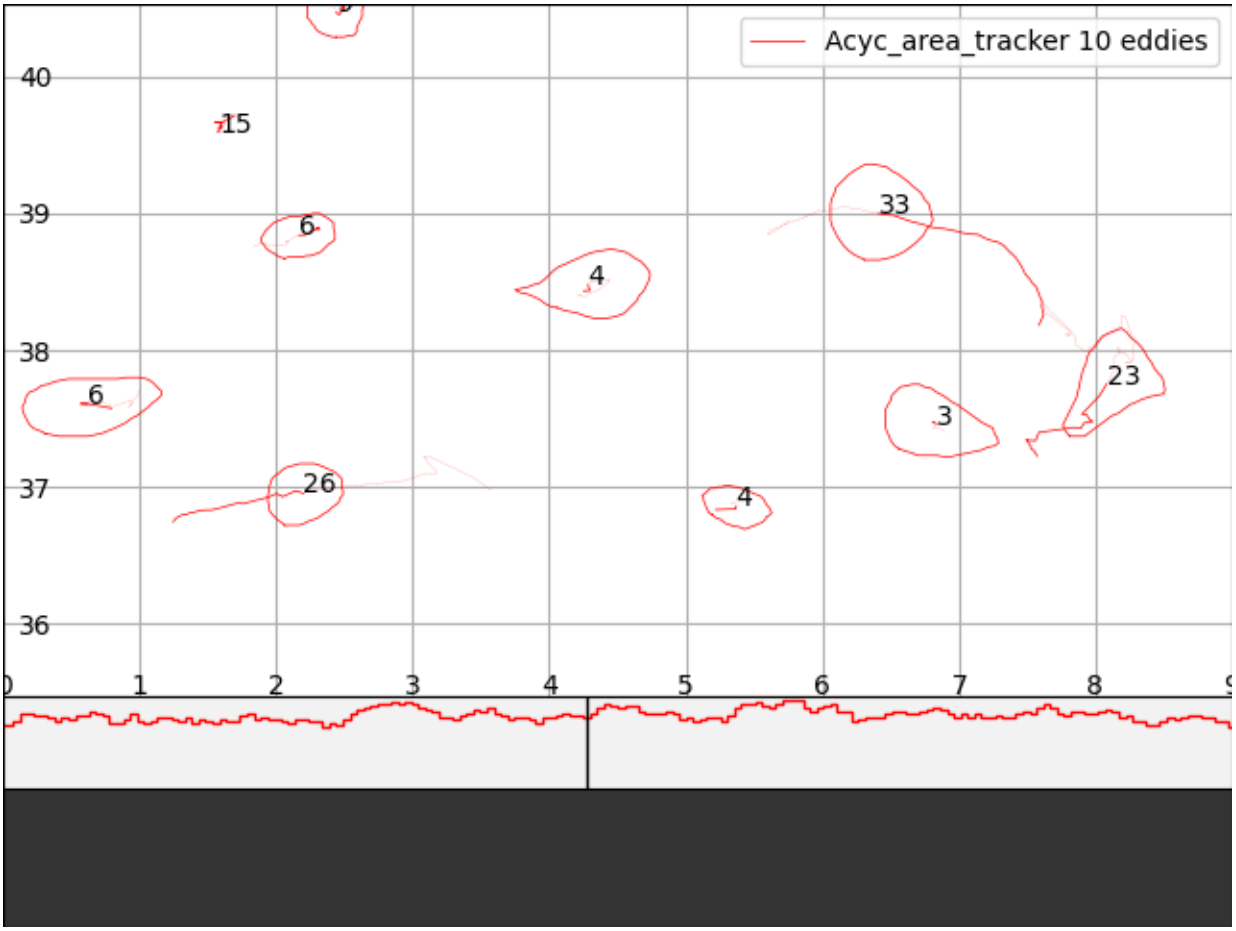
(continued from previous page)

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



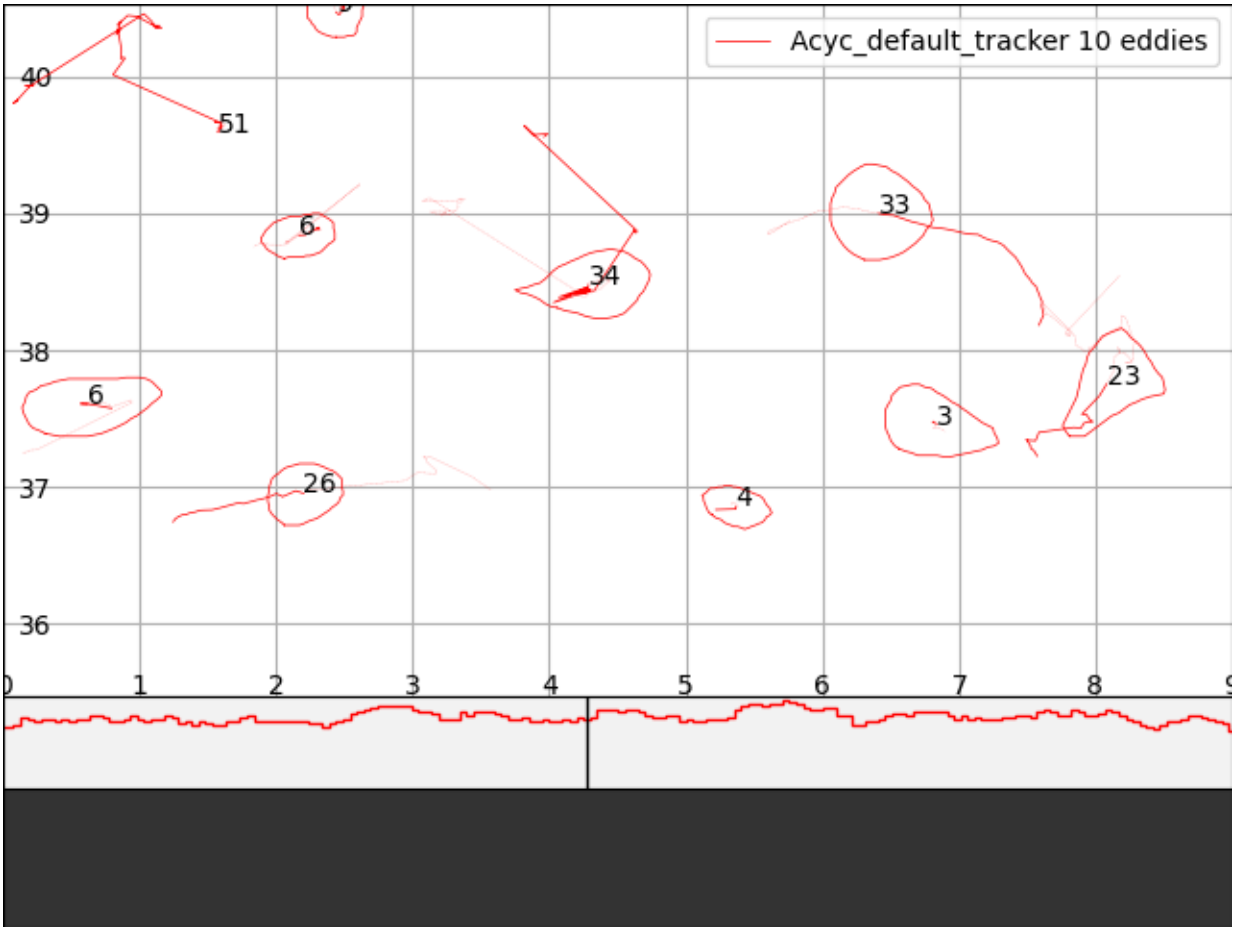
Start GUI with area tracker

```
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 16.719 seconds)

CHAPTER 6

Tracking diagnostics

6.1 Geographical statistics

```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
import py_eddy_tracker_sample


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
```

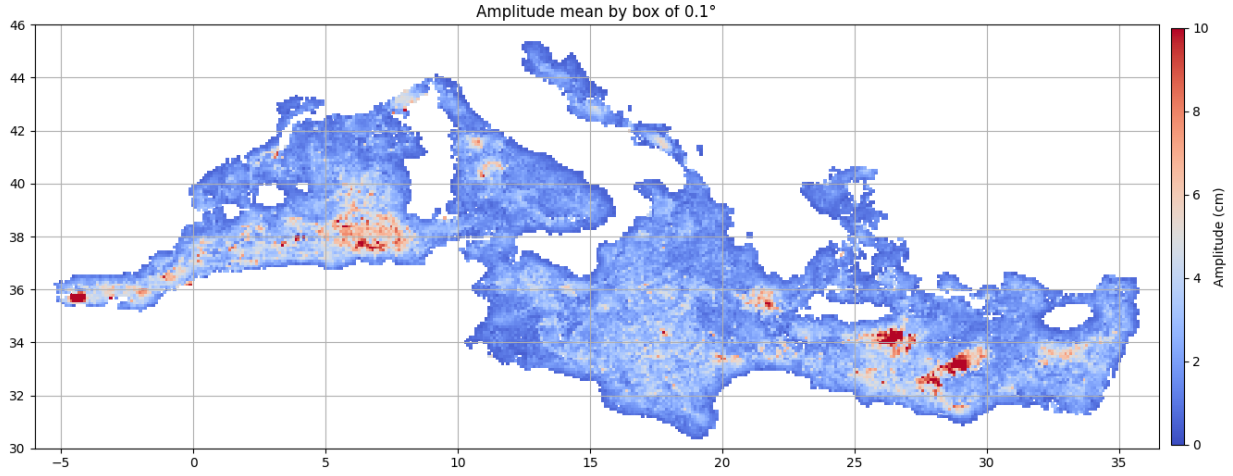
Mean of amplitude in each box

```
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()
```

(continues on next page)

(continued from previous page)

```
cb = plt.colorbar(m, cax=ax.figure.add_axes([0.94, 0.05, 0.01, 0.9]))
cb.set_label("Amplitude (cm)")
```

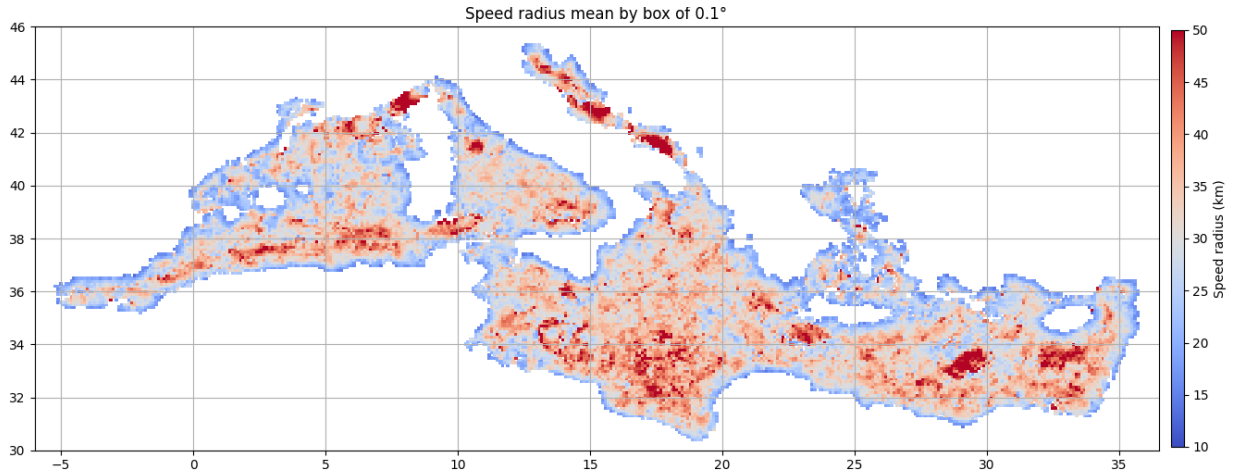


Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/envs/v3.2.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.2.0-py3.7.egg/py_eddy_tracker/observations/
↪observation.py:1324: RuntimeWarning: invalid value encountered in true_divide
  varname: ma.array(sum_obs / nb_obs, mask=nb_obs == 0),
We assume pixel position of grid is center for array
```

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/v3.2.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.2.0-py3.7.egg/py_eddy_tracker/observations/
↪observation.py:1324: RuntimeWarning: invalid value encountered in true_divide
```

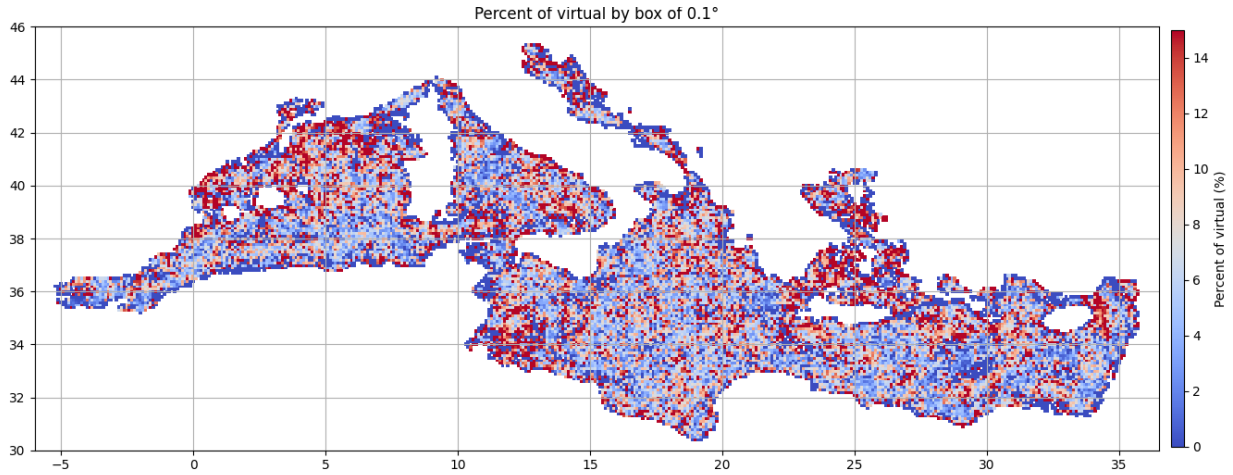
(continues on next page)

(continued from previous page)

```
  varname: ma.array(sum_obs / nb_obs, mask=nb_obs == 0),
We assume pixel position of grid is center for array
```

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/v3.2.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.2.0-py3.7.egg/py_eddy_tracker/observations/
↪observation.py:1324: RuntimeWarning: invalid value encountered in true_divide
  varname: ma.array(sum_obs / nb_obs, mask=nb_obs == 0),
We assume pixel position of grid is center for array
```

Total running time of the script: (0 minutes 4.636 seconds)

6.2 Lifetime Histogram

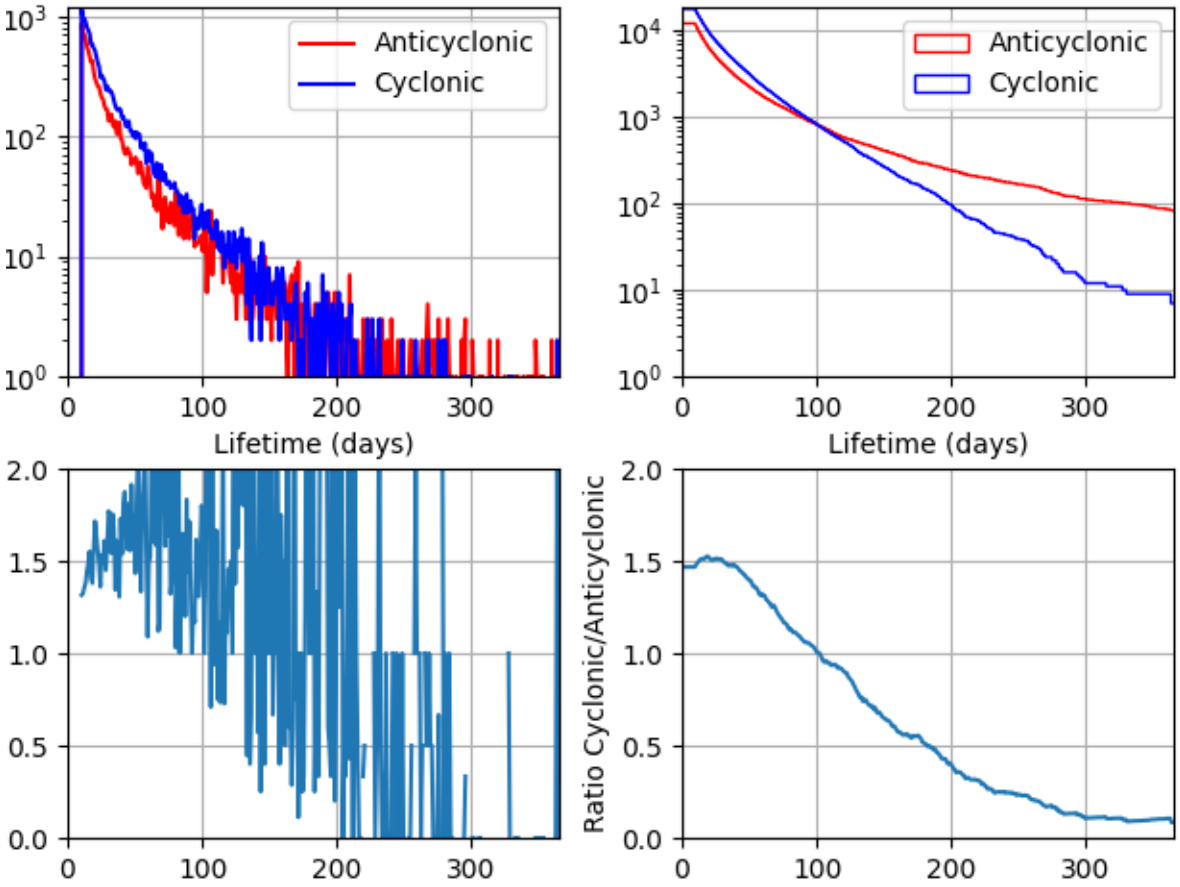
```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
import py_eddy_tracker_sample
from numpy import arange
```

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")
)
```

Plot

```
fig = plt.figure()
ax_lifetime = fig.add_axes([0.05, 0.55, 0.4, 0.4])
ax_cum_lifetime = fig.add_axes([0.55, 0.55, 0.4, 0.4])
ax_ratio_lifetime = fig.add_axes([0.05, 0.05, 0.4, 0.4])
ax_ratio_cum_lifetime = fig.add_axes([0.55, 0.05, 0.4, 0.4])

cum_a, bins, _ = ax_cum_lifetime.hist(
    a["n"], histtype="step", bins=arange(0, 800, 1), label="Anticyclonic", color="r"
)
cum_c, bins, _ = ax_cum_lifetime.hist(
    c["n"], histtype="step", bins=arange(0, 800, 1), label="Cyclonic", color="b"
)

x = (bins[1:] + bins[:-1]) / 2.0
ax_ratio_cum_lifetime.plot(x, cum_c / cum_a)

nb_a, nb_c = cum_a[:-1] - cum_a[1:], cum_c[:-1] - cum_c[1:]
ax_lifetime.plot(x[1:], nb_a, label="Anticyclonic", color="r")
ax_lifetime.plot(x[1:], nb_c, label="Cyclonic", color="b")

ax_ratio_lifetime.plot(x[1:], nb_c / nb_a)

for ax in (ax_lifetime, ax_cum_lifetime, ax_ratio_cum_lifetime, ax_ratio_lifetime):
    ax.set_xlim(0, 365)
    if ax in (ax_lifetime, ax_cum_lifetime):
        ax.set_ylim(1, None)
        ax.set_yscale("log")
        ax.legend()
    else:
        ax.set_ylim(0, 2)
        ax.set_ylabel("Ratio Cyclonic/Anticyclonic")
    ax.set_xlabel("Lifetime (days)")
    ax.grid()
```



Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.2.0/
↪examples/10_tracking_diagnostics/pet_lifetime.py:42: RuntimeWarning: divide by zero␣
↪encountered in true_divide
  ax_ratio_lifetime.plot(x[1:], nb_c / nb_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.2.0/
↪examples/10_tracking_diagnostics/pet_lifetime.py:42: RuntimeWarning: invalid value␣
↪encountered in true_divide
  ax_ratio_lifetime.plot(x[1:], nb_c / nb_a)
```

Total running time of the script: (0 minutes 2.759 seconds)

6.3 Count pixel used

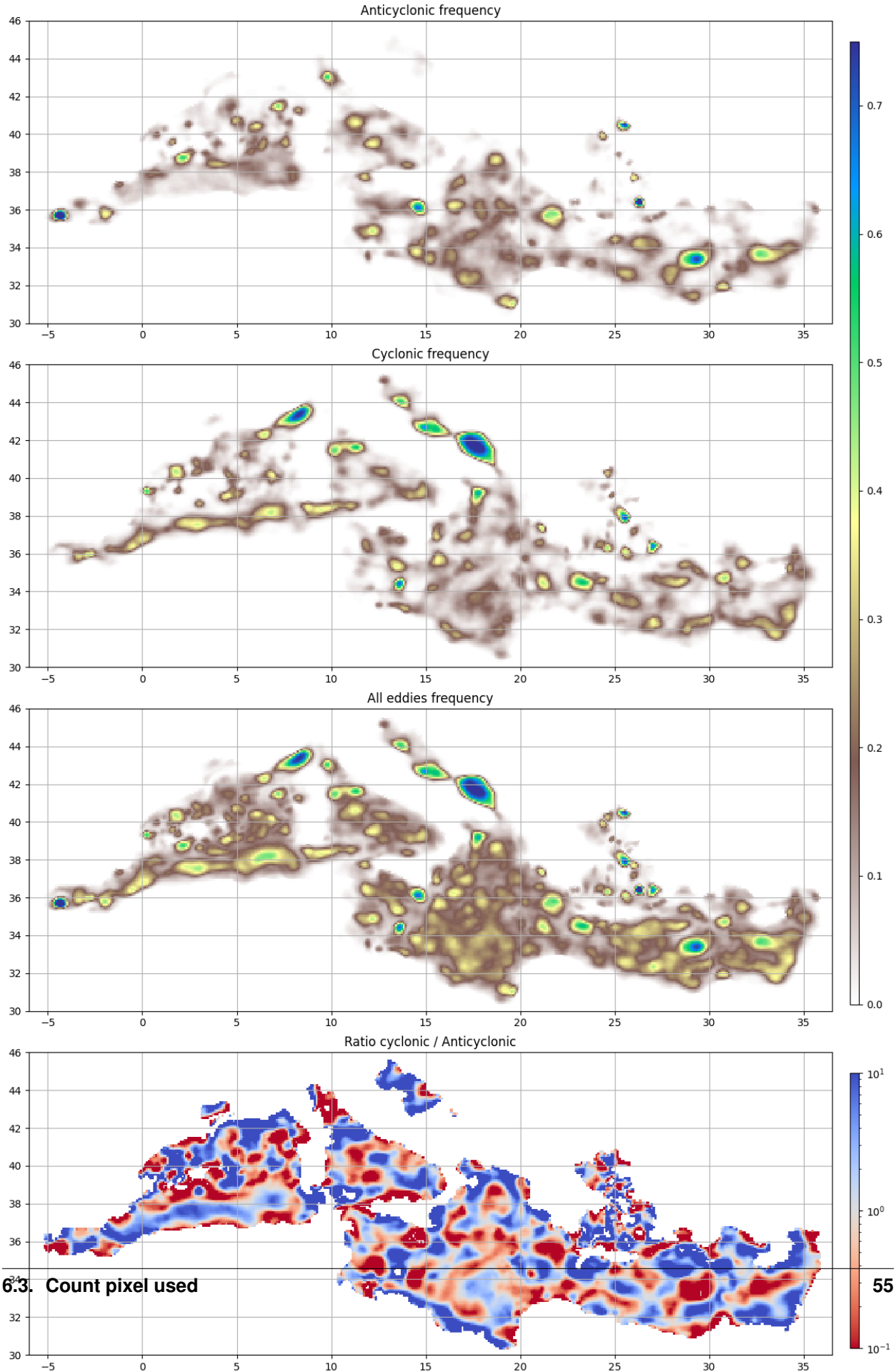
```
from matplotlib import pyplot as plt
from matplotlib.colors import LogNorm
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
import py_eddy_tracker_sample
```

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")
)
t0, t1 = a.period
step = 0.1
bins = ((-10, 37, step), (30, 46, step))
kwargs_pcolormesh = dict(
    cmap="terrain_r", vmin=0, vmax=0.75, factor=1 / (t1 - t0), 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/v3.2.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.2.0-py3.7.egg/py_eddy_tracker/dataset/grid.
↪py:1758: 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.
  self.x_bounds, self.y_bounds, self.grid(name).T * factor, **kwargs
```

Total running time of the script: (0 minutes 36.166 seconds)

6.4 Parameter Histogram

```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
import py_eddy_tracker_sample
from numpy import arange
```

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)
```

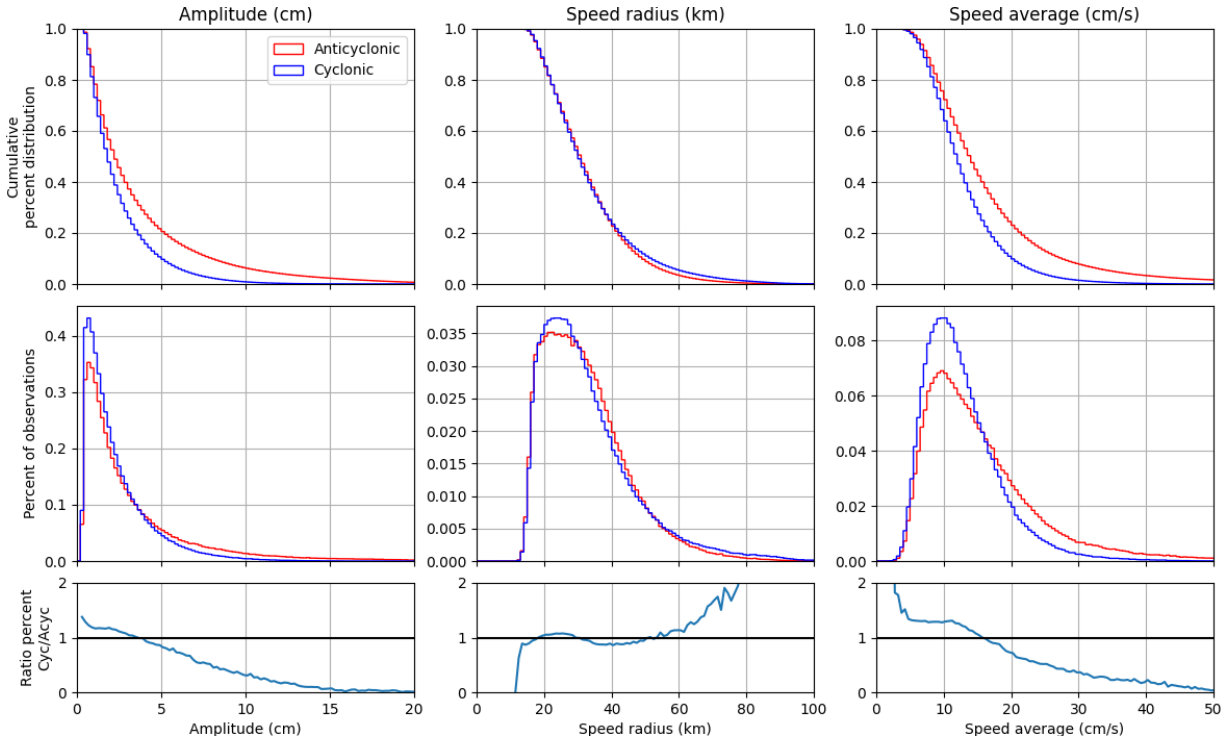
Plot

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

for x0, name, title, xmax, factor, bins in zip(
    (0.4, 0.72, 0.08),
    ("radius_s", "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)
```

(continues on next page)

(continued from previous page)

```
    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/v3.2.0/
↪examples/10_tracking_diagnostics/pet_histo.py:53: 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/v3.2.0/
↪examples/10_tracking_diagnostics/pet_histo.py:53: 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/v3.2.0/
↪examples/10_tracking_diagnostics/pet_histo.py:53: 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/v3.2.0/
↪examples/10_tracking_diagnostics/pet_histo.py:53: RuntimeWarning: invalid value␣
↪encountered in true_divide
```

(continues on next page)

(continued from previous page)

```
  ax_ratio.plot((bins[1:] + bins[:-1]) / 2, nb_c / nb_a)
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.2.0/
↪examples/10_tracking_diagnostics/pet_histo.py:53: 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 3.358 seconds)

6.5 Count center

```
from matplotlib import pyplot as plt
from matplotlib.colors import LogNorm
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
import py_eddy_tracker_sample
```

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
t0, t1 = a.period
step = 0.1
bins = ((-10, 37, step), (30, 46, step))
kwargs_pcolormesh = dict(
    cmap="terrain_r", vmin=0, vmax=2, factor=1 / (step ** 2 * (t1 - t0)), name="count"
)
```

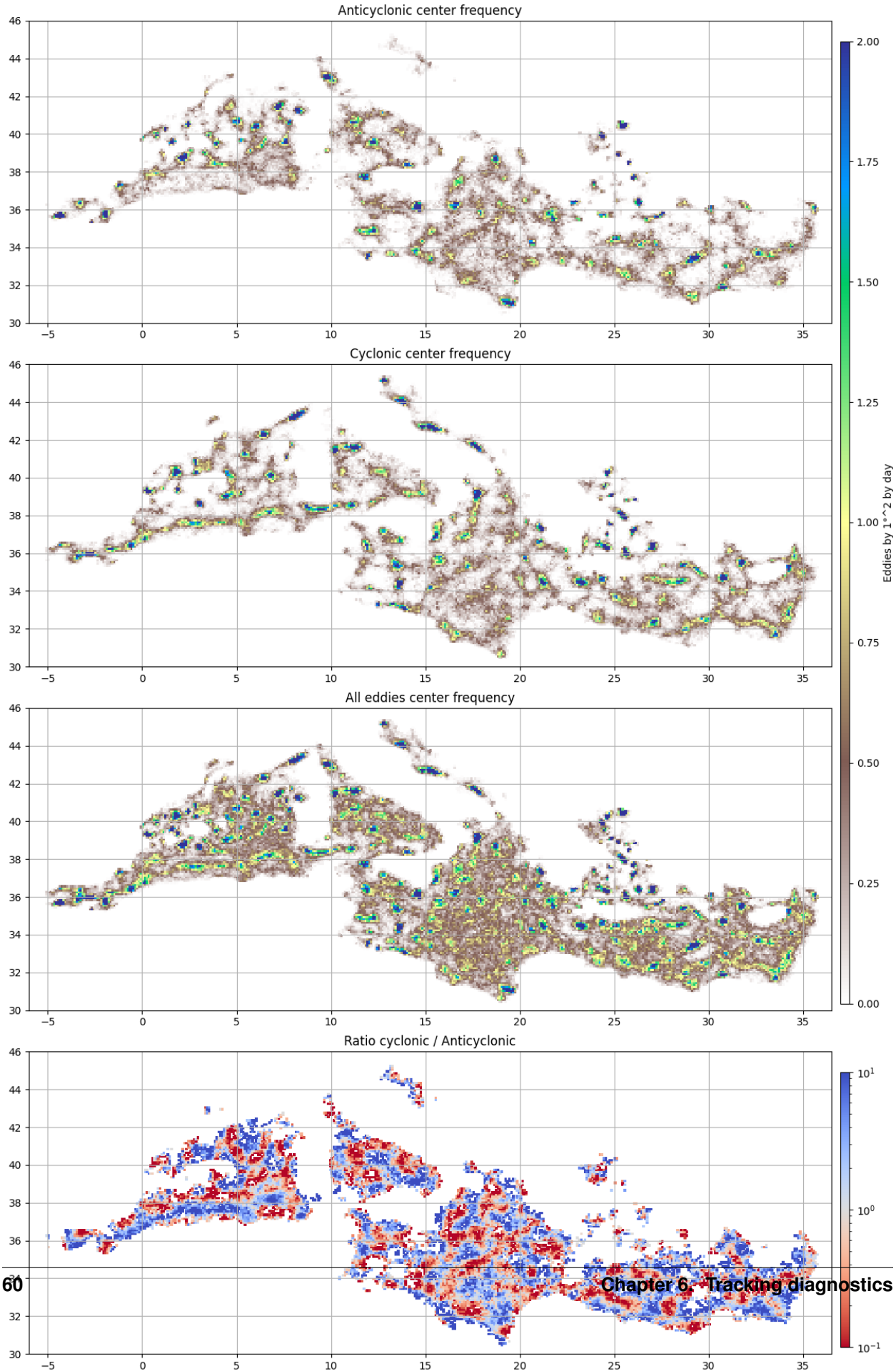
Plot

```
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
```

(continues on next page)

(continued from previous page)

```
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/v3.2.0/lib/
↪python3.7/site-packages/pyEddyTracker-3.2.0-py3.7.egg/py_eddy_tracker/dataset/grid.
↪py:1758: 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.
  self.x_bounds, self.y_bounds, self.grid(name).T * factor, **kwargs
```

Total running time of the script: (0 minutes 3.128 seconds)

6.6 Propagation Histogram

```
from matplotlib import pyplot as plt
from py_eddy_tracker.observations.tracking import TrackEddiesObservations
from py_eddy_tracker.generic import distance
import py_eddy_tracker_sample
from numpy import arange, empty
from numba import njit
```

We will create a function compile with numba, to compute a field which contains curvilign distance

```
@njit(cache=True)
def cum_distance_by_track(distance, track):
    tr_previous = 0
    d_cum = 0
    new_distance = empty(track.shape, dtype=distance.dtype)
    for i in range(distance.shape[0]):
        tr = track[i]
        if i != 0 and tr != tr_previous:
            d_cum = 0
        new_distance[i] = d_cum
        d_cum += distance[i]
        tr_previous = tr
    new_distance[i + 1] = d_cum
    return new_distance
```

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")
)
```

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

```
d_a = distance(a.longitude[:-1], a.latitude[:-1], a.longitude[1:], a.latitude[1:])
d_c = distance(c.longitude[:-1], c.latitude[:-1], c.longitude[1:], c.latitude[1:])
```

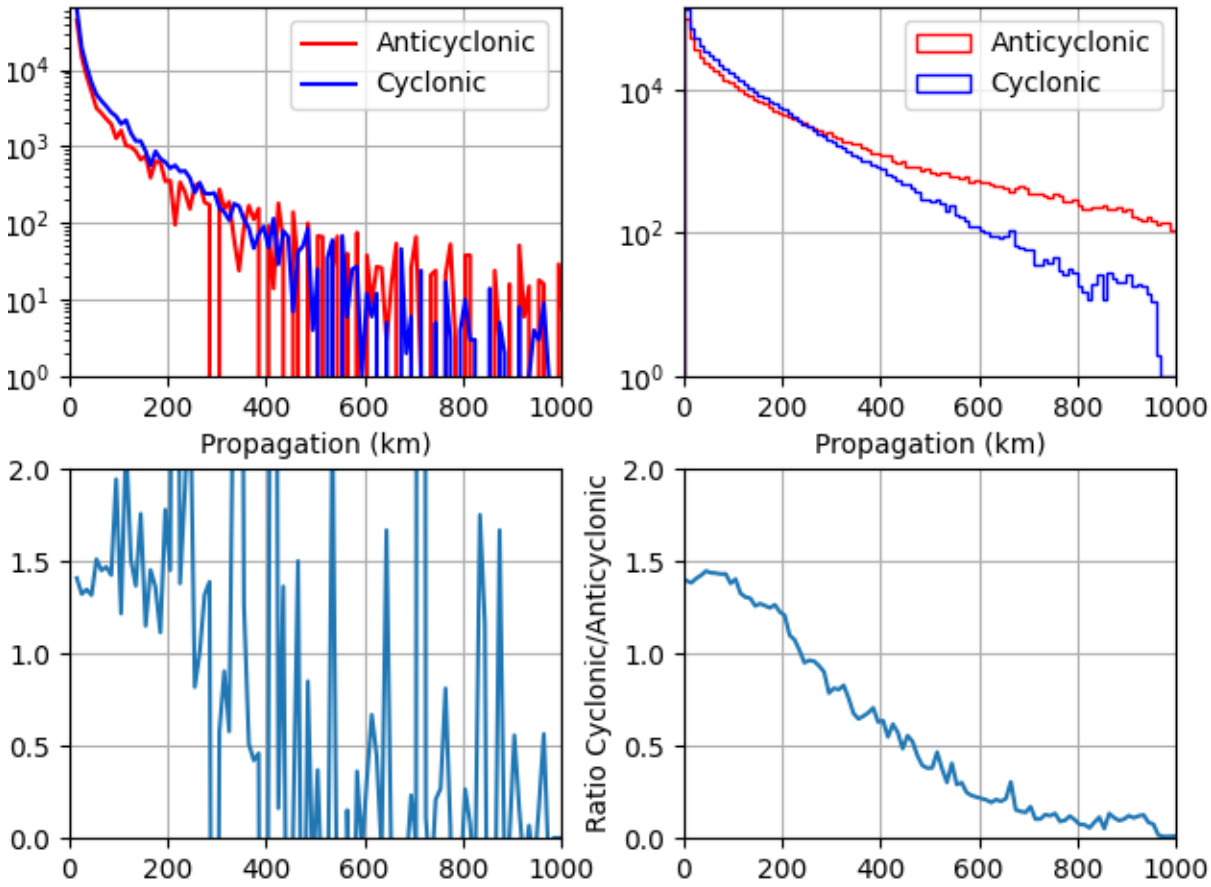
(continues on next page)

(continued from previous page)

```
d_a = cum_distance_by_track(d_a, a["track"]) / 1000.0
d_c = cum_distance_by_track(d_c, c["track"]) / 1000.0
```

Plot

```
fig = plt.figure()
ax_propagation = fig.add_axes([0.05, 0.55, 0.4, 0.4])
ax_cum_propagation = fig.add_axes([0.55, 0.55, 0.4, 0.4])
ax_ratio_propagation = fig.add_axes([0.05, 0.05, 0.4, 0.4])
ax_ratio_cum_propagation = fig.add_axes([0.55, 0.05, 0.4, 0.4])

bins = arange(0, 1500, 10)
cum_a, bins, _ = ax_cum_propagation.hist(
    d_a, histtype="step", bins=bins, label="Anticyclonic", color="r"
)
cum_c, bins, _ = ax_cum_propagation.hist(
    d_c, histtype="step", bins=bins, label="Cyclonic", color="b"
)

x = (bins[1:] + bins[:-1]) / 2.0
ax_ratio_cum_propagation.plot(x, cum_c / cum_a)

nb_a, nb_c = cum_a[:-1] - cum_a[1:], cum_c[:-1] - cum_c[1:]
ax_propagation.plot(x[1:], nb_a, label="Anticyclonic", color="r")
ax_propagation.plot(x[1:], nb_c, label="Cyclonic", color="b")

ax_ratio_propagation.plot(x[1:], nb_c / nb_a)

for ax in (
    ax_propagation,
    ax_cum_propagation,
    ax_ratio_cum_propagation,
    ax_ratio_propagation,
):
    ax.set_xlim(0, 1000)
    if ax in (ax_propagation, ax_cum_propagation):
        ax.set_ylim(1, None)
        ax.set_yscale("log")
        ax.legend()
    else:
        ax.set_ylim(0, 2)
        ax.set_ylabel("Ratio Cyclonic/Anticyclonic")
    ax.set_xlabel("Propagation (km)")
    ax.grid()
```



Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/py-eddy-tracker/checkouts/v3.2.0/
↪examples/10_tracking_diagnostics/pet_propagation.py:76: RuntimeWarning: invalid␣
↪value encountered in true_divide
  ax_ratio_propagation.plot(x[1:], nb_c / nb_a)
```

Total running time of the script: (0 minutes 3.791 seconds)

CHAPTER 7

Eddy identification

Run the identification process for a single day

7.1 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*

7.2 Python code

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

Activate verbose

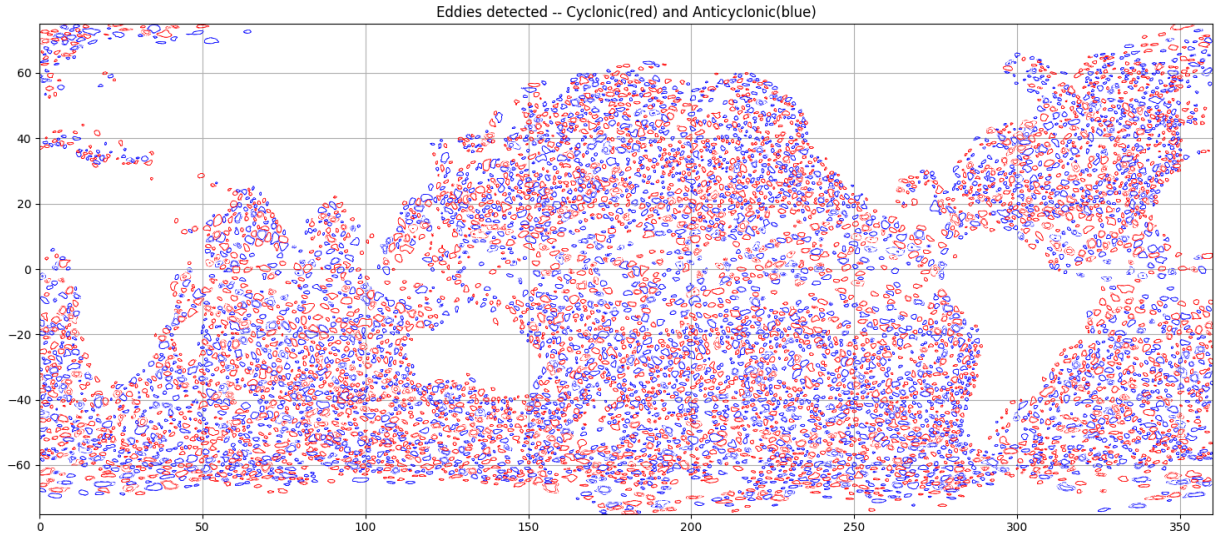
```
from py_eddy_tracker import start_logger
start_logger().setLevel('DEBUG') # Available options: ERROR, WARNING, INFO, DEBUG
```

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 8

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')
```

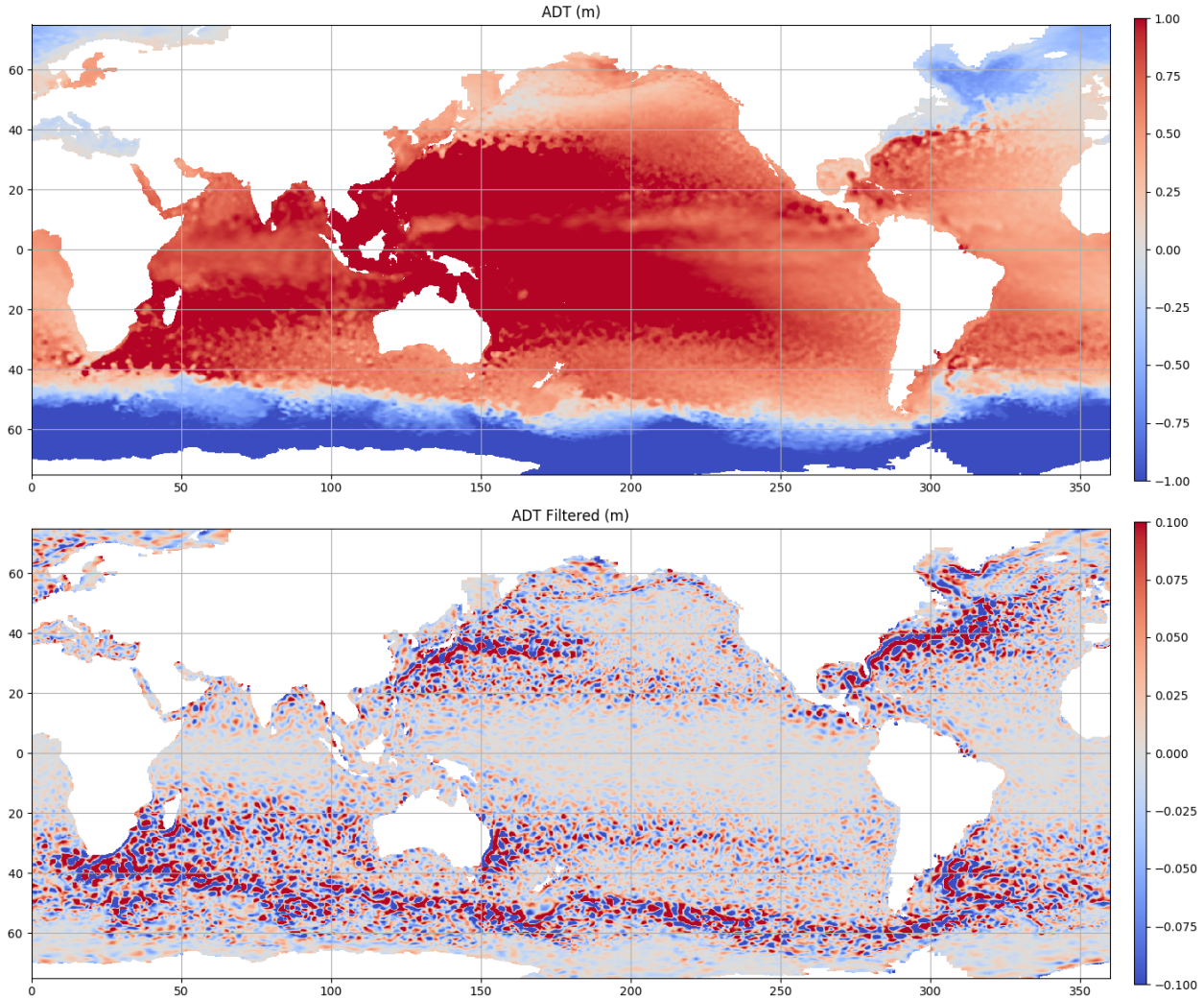
Add second plot

```
ax = fig.add_axes([.02, .02, .9, .45])
ax.set_title('ADT Filtered (m)')
ax.set_aspect('equal')
ax.set_ylim(-75, 75)
```

(continues on next page)

(continued from previous page)

```
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')
```



CHAPTER 9

Spectrum

9.1 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),
)
```

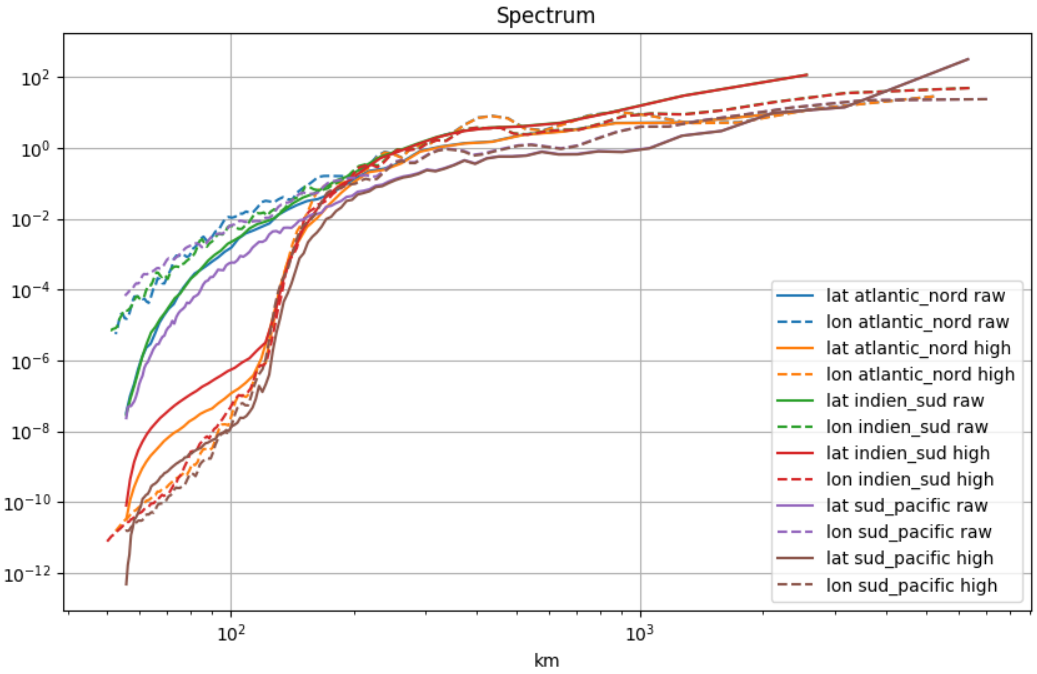
Compute and display spectrum

```
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')
```

(continues on next page)

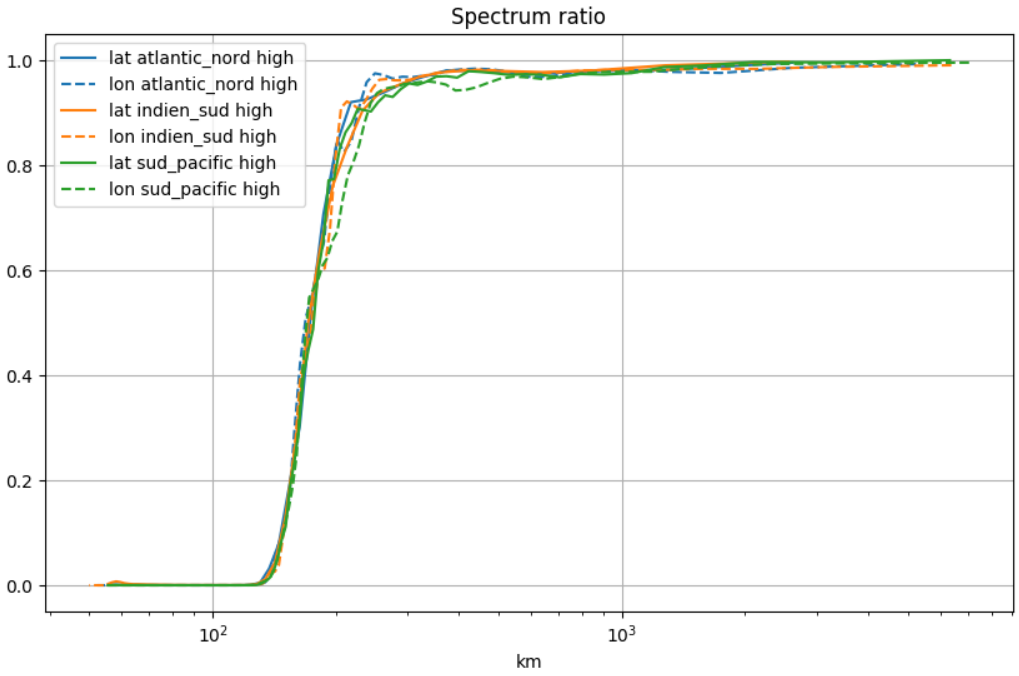
(continued from previous page)

```
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 10

Tracking

10.1 Default method

To run a tracking just create an yaml file with minimal specification (*FILES_PATTERN* and *SAVE_DIR*).

Example of 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

10.2 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 11

Customize tracking

11.1 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 12

Grid

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

EARTH_RADIUS = 6370997.0

GRAVITY = 9.807

N = 1

bounds
Give bound

centered

contours

coordinates

copy(*grid_in*, *grid_out*)
Duplicate a variable

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*)

Compute eddy identification on specified grid

Parameters

- **grid_height** (*str*) – Grid name of height
- **uname** (*str*) – Grid name of u speed component
- **vname** (*str*) – Grid name of v speed component
- **date** (*datetime.datetime*) – Date which will be store in object to date data
- **step** (*float*, *int*) – Height between two layers in m
- **shape_error** (*float*, *int*) – Maximal error allow for outter contour in %
- **sampling** (*int*) – Sampling of contour and speed profile
- **pixel_limit** ((*int*, *int*), *None*) – Min and max of pixel which must be inside inner and outer contour to be considered like an eddy
- **precision** (*float*, *None*) – Truncate value at the defined precision in m
- **force_height_unit** (*str*) – Unit to used for height unit
- **force_speed_unit** (*str*) – Unit to used for speed unit

Returns Return a list of 2 elements: Anticyclone and Cyclone

Return type *py_eddy_tracker.observations.observation.EddiesObservations*

filename

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

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 grid required

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

give grid tiles required, without buffer system

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

create a high filter with a low one

indexs

interpolators

is_centered

Give information if pixel is describe with center position or a corner

is_circular()
Check grid circularity

load()
Load variable (data)

load_general_features()
Load attrs

low_filter(*grid_name*, *w_cut*, ***kwargs*)
low filtering

setup_coordinates()

speed_coef

units(*varname*)

variables_description

vars

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

x_bounds

x_c

x_dim

xinterp

y_bounds

y_c

y_dim

yinterp

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

Class only for regular grid

add_uv(*grid_height*, *uname='u'*, *vname='v'*, *stencil_halfwidth=4*)
Compute a u and v grid

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*)

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 index which describe the path between to position

compute_stencil(*data*, *stencil_halfwidth=4*, *mode='reflect'*, *vertical=False*)

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

display(*ax*, *name*, *factor=1*, ***kwargs*)

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

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

get_pixels_in(*contour*)

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*)
Compute z over lons, lats

Parameters

- **grid_name** (*str*) – Grid which will be interp
- **lons** – new x
- **lats** – new y

Returns new z

is_circular()
Check if 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*)

setup_coordinates()

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

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

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

x_size

xstep
Only for regular grid with no step variation

ystep
Only for regular grid with no step variation

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 which manage 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

bbox_indice(*vertices*)

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*)

py_eddy_tracker.dataset.grid.**bbox_indice_regular**

py_eddy_tracker.dataset.grid.**bbox_slice**

py_eddy_tracker.dataset.grid.**compute_pixel_path**
Give a series of index which describe the path between to position

py_eddy_tracker.dataset.grid.**filter2D**(*src*, *ddepth*, *kernel*[, *dst*[, *anchor*[, *delta*[, *borderType*]]]]) → dst

. @brief Convolves an image with the kernel. . . The function applies an arbitrary linear filter to an image. In-place operation is supported. When . the aperture is partially outside the image, the function interpolates outlier pixel values . according to the specified border mode. . . The function does actually compute correlation, not the convolution: . . f[texttt{dst} (x,y) = sum _{ substack{0leq x' < texttt{kernel.cols}\{0leq y' < texttt{kernel.rows}}}} texttt{kernel} (x',y')* texttt{src} (x+x'- texttt{anchor.x} ,y+y'- texttt{anchor.y})f] . . That is, the kernel is not mirrored around the anchor point. If you need a real convolution, flip . the kernel using #flip and set the new anchor to *(kernel.cols - anchor.x - 1, kernel.rows - . anchor.y - 1).* . . The function uses the DFT-based algorithm in case of sufficiently large kernels (~‘11 x 11‘ or . larger) and the direct algorithm for small kernels. . . @param src input image. . @param dst output image of the same size and the same number of channels as src. . @param ddepth desired depth of the destination image, see @ref filter_depths “combinations” . @param kernel convolution kernel (or rather a correlation kernel), a single-channel floating point . matrix; if

you want to apply different kernels to different channels, split the image into . separate color planes using split and process them individually. . @param anchor anchor of the kernel that indicates the relative position of a filtered point within . the kernel; the anchor should lie within the kernel; default value (-1,-1) means that the anchor . is at the kernel center. . @param delta optional value added to the filtered pixels before storing them in dst. . @param borderType pixel extrapolation method, see #BorderTypes. #BORDER_WRAP is not supported. . @sa sepFilter2D, dft, matchTemplate

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

py_eddy_tracker.dataset.grid.**has_masked_value**

py_eddy_tracker.dataset.grid.**has_value**

py_eddy_tracker.dataset.grid.**lat**

py_eddy_tracker.dataset.grid.**lon**

py_eddy_tracker.dataset.grid.**mean_coordinates**

py_eddy_tracker.dataset.grid.**mean_on_regular_contour**

py_eddy_tracker.dataset.grid.**nb_pixel**

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

py_eddy_tracker.dataset.grid.**pixels_index**

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

py_eddy_tracker.dataset.grid.**uniform_resample_stack**

py_eddy_tracker.dataset.grid.**value_on_regular_contour**

CHAPTER 13

Observations

py-eddy-tracker is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

py-eddy-tracker is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with py-eddy-tracker. If not, see <http://www.gnu.org/licenses/>.

Copyright (c) 2014-2017 by Evan Mason and Antoine Delepoulle Email: emason@imedea.uib-csic.es
==========

observation.py

Version 3.0.0

---

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 hold eddy properties *amplitude* and counts of *local maxima/minima* within a closed region of a sea level anomaly field.

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

add_fields(*fields*)
Add a new field

add_rotation_type()

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

circle_contour()

coherence(*other*)
Check coherence between two dataset

classmethod concatenate(*observations*)

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

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*)

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

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

dtype
Return dtype to build numpy array

elements
Return all variable name

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

classmethod from_netcdf(*handler*)

classmethod from_zarr(*handler*)

global_attr

grid_count(*bins*, *intern=False*, *center=False*)

grid_stat(*bins*, *varname*)

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

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

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

latitude

classmethod load_file(*filename*, ***kwargs*)

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

classmethod load_from_zarr(*filename*, *raw_data=False*, *remove_vars=None*, *include_vars=None*)

longitude

mask_function(*other*, *distance*)

match(*other*, *intern=False*, *cmin=0*)
return index and score compute with area

merge(*other*)
Merge two dataset

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

static new_like(*eddies*, *new_size*)

obs
return an array observations

classmethod obs_dimension(*handler*)

observation_number

observations

only_variables

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*, *ref=None*, *factor=1*, ***kwargs*)

set_global_attr_netcdf(*h_nc*)

set_global_attr_zarr(*h_zarr*)

shape

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

sign_legend

sign_type

static solve_conflict(*cost*)

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

solve_function(*cost_matrix*)

static solve_simultaneous(*cost*)

time

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

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

track_array_variables

track_extra_variables

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

tracks

write_file(*path='./'*, *filename='%(path)s/%(sign_type)s.nc'*, *zarr_flag=False*)
Write a netcdf with eddy obs

static zarr_dimension(*filename*)

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

elements
Return all variable name

py_eddy_tracker.observations.observation.**grid_count_**

py_eddy_tracker.observations.observation.**shifted_ellipsoid_degrees_mask2**
work only if major is an array but faster * 6

==========

tracking.py

Version 3.0.0

---

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

Bases: *py_eddy_tracker.observations.observation.EddiesObservations*

Class to practice Tracking on observations

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

NOGROUP = 0

compute_index()

elements
Return all variable name

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_with_area(*area*, ***kwargs*)
Extract with a bounding box

Parameters **area** (*dict*) – 4 coordinates in a dictionary to specify bounding box (lower left corner and upper right corner)

extract_with_length(*bounds*)

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

Parameters **period** (*(datetime.datetime,datetime.datetime)*) – two date to define period, must be specify from 1/1/1950

Returns same object with selected data

filled_by_interpolation(*mask*)
Filled selected values by interpolation

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

get_mask_from_id(*tracks*)

index_from_track

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

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

nb_obs_by_track

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

period
Give time coverage Returns: 2 date

plot(*ax*, *ref=None*, ***kwargs*)

position_filter(*median_half_window*, *loess_half_window*)

set_global_attr_netcdf(*h_nc*)
Set global attr

set_tracks(*x*, *y*, *ids*, *window*)

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

py_eddy_tracker.observations.tracking.**compute_index**

py_eddy_tracker.observations.tracking.**compute_mask_from_id**

py_eddy_tracker.observations.tracking.**track_loess_filter**
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

py_eddy_tracker.observations.tracking.**track_median_filter**
Apply a loess 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

CHAPTER 14

Eddy Features

==========

class py_eddy_tracker.eddy_feature.**Amplitude**(*contour*, *contour_height*, *data*, *interval*)

Bases: object

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

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 anticyclonic 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

mle

nb_pixel

pixel_mask

sla

within_amplitude_limits()
Need update

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

DELTA_PREC = 1e-10

DELTA_SUP = 0.01

check_closing(*path*)

contour_index

contours

cvalues

display(*ax*, *step=1*, *only_used=False*, *only_unused=False*, *only_contain_eddies=False*, ***kwargs*)

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

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

py_eddy_tracker.eddy_feature.**detect_local_minima_**
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

py_eddy_tracker.eddy_feature.**index_from_nearest_path_with_pt_in_bbox_**
Get index from nearest path in edge bbox contain pt

CHAPTER 15

Featured tracking

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

Bases: *py_eddy_tracker.observations.observation.EddiesObservations*

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

py_eddy_tracker.featured_tracking.old_tracker_reference.**check_ratio**
Only very few case are remove with selection :param current_mask: :param self_amplitude: :param other_amplitude: :param self_radius: :param other_radius: :return:

CHAPTER 16

Polygon function

==========

py_eddy_tracker.poly.**bbox_intersection**
: compute bbox to check if there are a bbox intersection

py_eddy_tracker.poly.**create_vertice**

py_eddy_tracker.poly.**create_vertice_from_2darray**

py_eddy_tracker.poly.**get_wrap_vertice**

py_eddy_tracker.poly.**is_left**
: http://geomalgorithms.com/a03-_inclusion.html isLeft(): tests if a point is Left|On|Right of an infinite line. Input: three points P0, P1, and P2 Return: >0 for P2 left of the line through P0 and P1

  =0 for P2 on the line <0 for P2 right of the line

  See: Algorithm 1 “Area of Triangles and Polygons”

py_eddy_tracker.poly.**poly_area**

py_eddy_tracker.poly.**poly_contain_poly**

py_eddy_tracker.poly.**polygon_overlap**(*p0*, *p1*, *minimal_area=False*)

py_eddy_tracker.poly.**vertice_overlap**(*x0*, *y0*, *x1*, *y1*, *minimal_area=False*)

py_eddy_tracker.poly.**winding_number_grid_in_poly**
http://geomalgorithms.com/a03-_inclusion.html wn_PnPoly(): winding number test for a point in a polygon

Input: P = a point, V[] = vertex points of a polygon V[n+1] with V[n]=V[0]

Return: wn = the winding number (=0 only when P is outside)

py_eddy_tracker.poly.**winding_number_poly**

CHAPTER 17

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

H

I

Y

Z