tobac-tutorials Documentation

Release v0.1

Fabian Senf, Max Heikenfeld, Will Jones

May 06, 2024

GETTING STARTED

This content is obsolete and will be deactivated or archived as of April 1, 2024.

CHAPTER

ONE

HOW TO RUN TOBAC-TUTORIALS

1.1 Getting started: Installation

This documentation explains how to use tobac-tutorial notebooks for `tobac` 2.x using an Anaconda environment. For running tutorials on your local device, create a new folder where the data will be located. When working with a Linux environment, open a terminal in your folder and clone the `tobac` repository from Github with:

```
git clone https://github.com/climate-processes/tobac
```

Change your directory to the `tobac` folder.

```
cd tobac
```

The tutorials need to be used within the v2.0-dev development branch. For fetching all branches and checking which branch you're on, use:

```
git branch -a
```

As you can see, you're either on v2.0-dev branch or master branch (not compatible with these tutorials, for that use examples provided on tobac master branch). You need to switch branches, so use:

```
git checkout --track origin/v2.0-dev
```

Check again which branch you're on. The active branch will appear with a * and in green color.

```
git branch
```

Now this works fine, but you'll also need to clone the respective repository where the tutorials are located. Change your directory pointing to the main directory and the clone the repository like before with:

```
cd ..
git clone https://github.com/climate-processes/tobac-tutorials
cd tobac-tutorials
```

This creates a second folder next to your `tobac` folder. Again, check what branch you're in (you'll notice there exists only one branch, the master branch.)

```
git status
```

In the next step, you need to create a conda environment for providing all required software dependencies. First, you would need to go down and initiate a new conda environment with:

```
cd ..
conda create --prefix tobac-test-env
```

Enter your environment's name and the Python version of your choice. Check the travis.yml for further information on currently supported Python versions.

Activate your conda environment. Then, change to your main directory and install the `tobac` requirements from the provided file.

```
conda activate ./tobac-test-env
cd tobac
conda install -c conda-forge --file conda-requirements.txt
```

With this, all necessary prerequisites for running `tobac` are met so you can install the package. You can choose between two methods:

1. just install the `tobac` version from your current branch (v2.0-dev)

   ```
   pip install .
   ```

2. enable to access the code from other branches as well, you can install the package with (editable mode for developers):

   ```
   pip install -e .
   ```

   This will enable you to update your `tobac` import to the version of another branch, as you checkout on that branch. Switching between branches is not necessary to run these tutorials, but could be useful if you also plan to test the latest release of `tobac` or code from other development branches.

Up to this point, you should have successfully installed `tobac`. As the tutorials are provided by Jupyter Notebook, you'll need to install `jupyter` in your environment as well.

In the active environment, you need to install additional packages that are exclusively used my the tutorial notebooks. These packages are however not mandatory for the execution of `tobac`itself. So please type:

```
conda install jupyter boto3 basemap basemap-data-hires rioxarray seaborn
```

1.2 Running Jupyter Notebook

For running the tutorials, change the directory to where the examples are located. E.g., if you want to know how the toolset can be applied to OLR model data, write:

```
cd ..
cd tobac-tutorials
jupyter notebook
```

The Jupyter Notebook will be opened in a new browser. By clicking "Run" you can execute the code line after line and follow the examples for different data sets.

Please note: When running the tutorials for the first time, you'll need to modify the code in the part of "Download the data: …" for downloading the example data from Zenodo. Remove # for uncommenting the lines and enabling the download (only for lines with #, not ##). You only need to download the data once. So when the data for all examples is saved in your tobac-tutorials folder, uncomment the lines again.

CHAPTER

TWO

IDEALIZED CASE 1: TRACKING OF A TEST BLOB IN 2D

This tutorial shows the most important steps of tracking with `tobac` using an idealized case:

1. *Input Data*
2. *Feature Detection*
3. *Tracking / Trajectory Linking*
4. *Segmentation*
5. *Statistical Analysis*

2.1 Import Libraries

We start by importing tobac:

[1]:
```
import tobac
```

We will also need matplotlib in inline-mode for plotting and numpy:

[2]:
```
import matplotlib.pyplot as plt

%matplotlib inline

import numpy as np
```

For a better readability of the graphs:

[3]:
```
import seaborn as sns

sns.set_context("talk")
```

Tobac works with a Python package called `xarray`, which introduces `DataArrays`. In a nutshell these are `numpy`-arrays with labels. For a more extensive description have a look at the xarray Documentation.

2.2 1. Input Data

There are several utilities implemented in tobac to create simple examples of such arrays. In this tutorial we will use the function `make_simple_sample_data_2D()` to create a moving test blob in 2D:

[4]:
```
test_data = tobac.testing.make_simple_sample_data_2D()
test_data
```

[4]:
```
<xarray.DataArray 'w' (time: 100, y: 50, x: 100)>
[500000 values with dtype=float64]
Coordinates:
  * time       (time) datetime64[ns] 2000-01-01T12:00:00 ... 2000-01-01T13:39:00
  * y          (y) float64 0.0 1e+03 2e+03 3e+03 ... 4.7e+04 4.8e+04 4.9e+04
  * x          (x) float64 0.0 1e+03 2e+03 3e+03 ... 9.7e+04 9.8e+04 9.9e+04
    latitude   (y, x) float64 ...
    longitude  (y, x) float64 ...
Attributes:
    units:    m s-1
```

As you can see our generated data describes a field called 'w' with the unit m/s at 100, 50 and 100 datapoints of time, x and y. Additionally, the data contains the latitude and longitude coordinates of the field values. To access the values of 'w' in the first timeframe, we can use

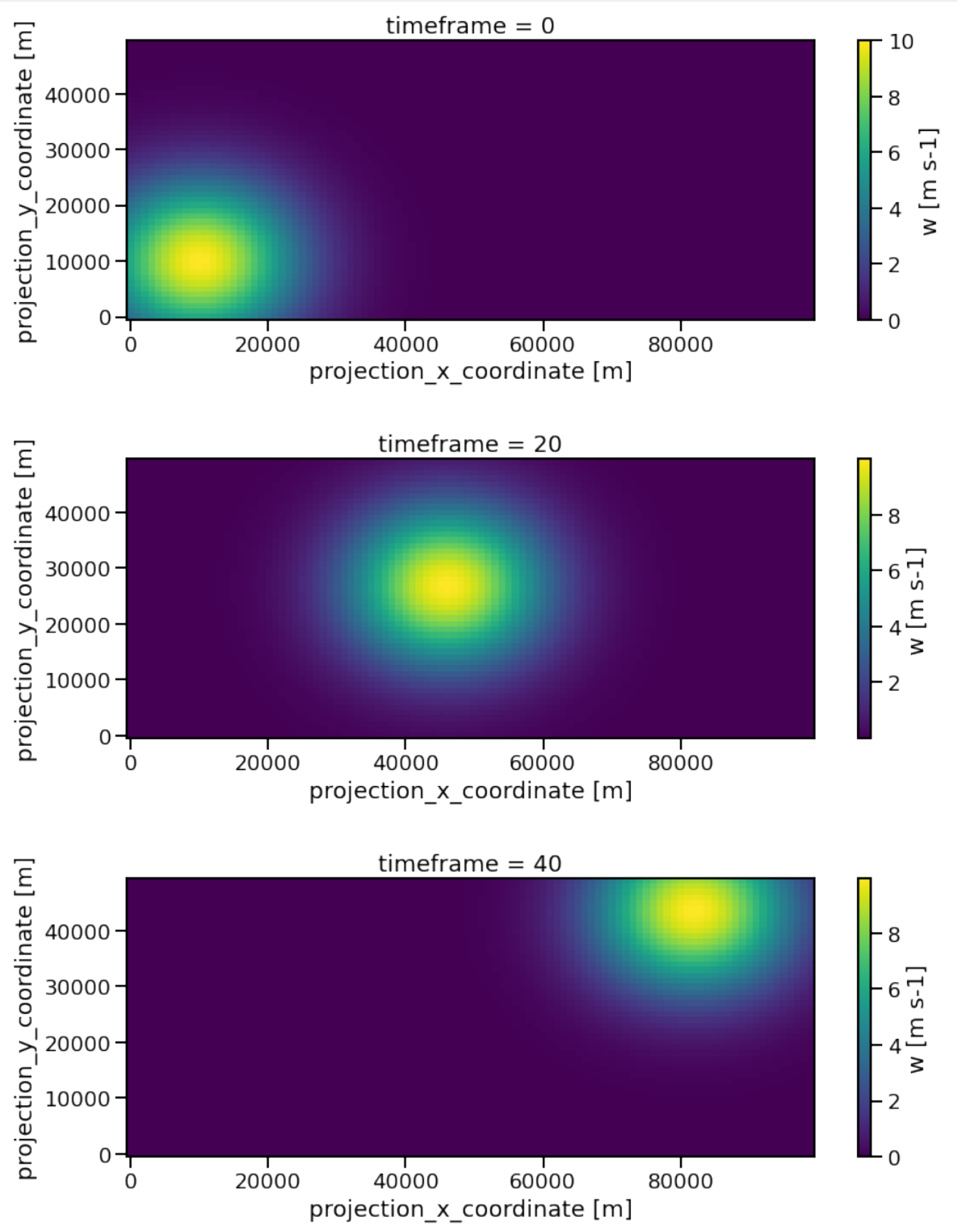
[5]:
```
test_data.data[0]
```

[5]:
```
array([[3.67879441e+00, 4.04541885e+00, 4.40431655e+00, ...,
        2.22319774e-16, 9.26766698e-17, 3.82489752e-17],
       [4.04541885e+00, 4.44858066e+00, 4.84324569e+00, ...,
        2.44475908e-16, 1.01912721e-16, 4.20608242e-17],
       [4.40431655e+00, 4.84324569e+00, 5.27292424e+00, ...,
        2.66165093e-16, 1.10954118e-16, 4.57923372e-17],
       ...,
       [6.45813368e-03, 7.10174389e-03, 7.73178977e-03, ...,
        3.90282972e-19, 1.62694148e-19, 6.71461808e-20],
       [4.43860604e-03, 4.88095244e-03, 5.31397622e-03, ...,
        2.68237303e-19, 1.11817944e-19, 4.61488502e-20],
       [3.02025230e-03, 3.32124719e-03, 3.61589850e-03, ...,
        1.82522243e-19, 7.60865909e-20, 3.14020145e-20]])
```

which is then just an array of numbers of the described shape.

To visualize the data, we can plot individual time frames using matplotlib per imshow:

[6]:
```
fig, axs = plt.subplots(ncols=1, nrows=3, figsize=(12, 16), sharey=True)
plt.subplots_adjust(hspace=0.5)

for i, itime in enumerate([0, 20, 40]):

    # plot the 2D blob field in colors
    test_data.isel(time=itime).plot(ax=axs[i])

    axs[i].set_title(f"timeframe = {itime}")
```



This tells as that our data is a single moving blob, which is what we are going the analyze with tobac now.

2.3 2. Feature Detection

The first step of the general working routine of tobac is the identification of features. This essentially means finding the maxima or minima of the data.

To use the according functions of tobac we need to specify:

- the thresholds below/above the features are detected
- the spacing of our data

The spacing of the temporal and spatial dimension can be extracted from the data using a build-in utility:

[7]:
```
dxy, dt = tobac.utils.get_spacings(test_data)
```

To get an idea of which order of magnitude our thresholds should be, we check the maximum of our data:

[8]:
```
test_data.max()
```

[8]:
```
<xarray.DataArray 'w' ()>
array(10.)
```

Since we know that our data will only have one maximum it is reasoable to choose 9 as our threshold, but keep in mind that we could also add multiple values here if our data would be more complex.

[9]:
```
threshold = 9
```

Now we are ready to apply the feature detection algorithm. Notice that this is a minimal input. The function has several other option we will cover in later tutorials.

[10]:
```
%%capture
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    test_data, dxy, threshold
)
```

Let's inspect the resulting object:

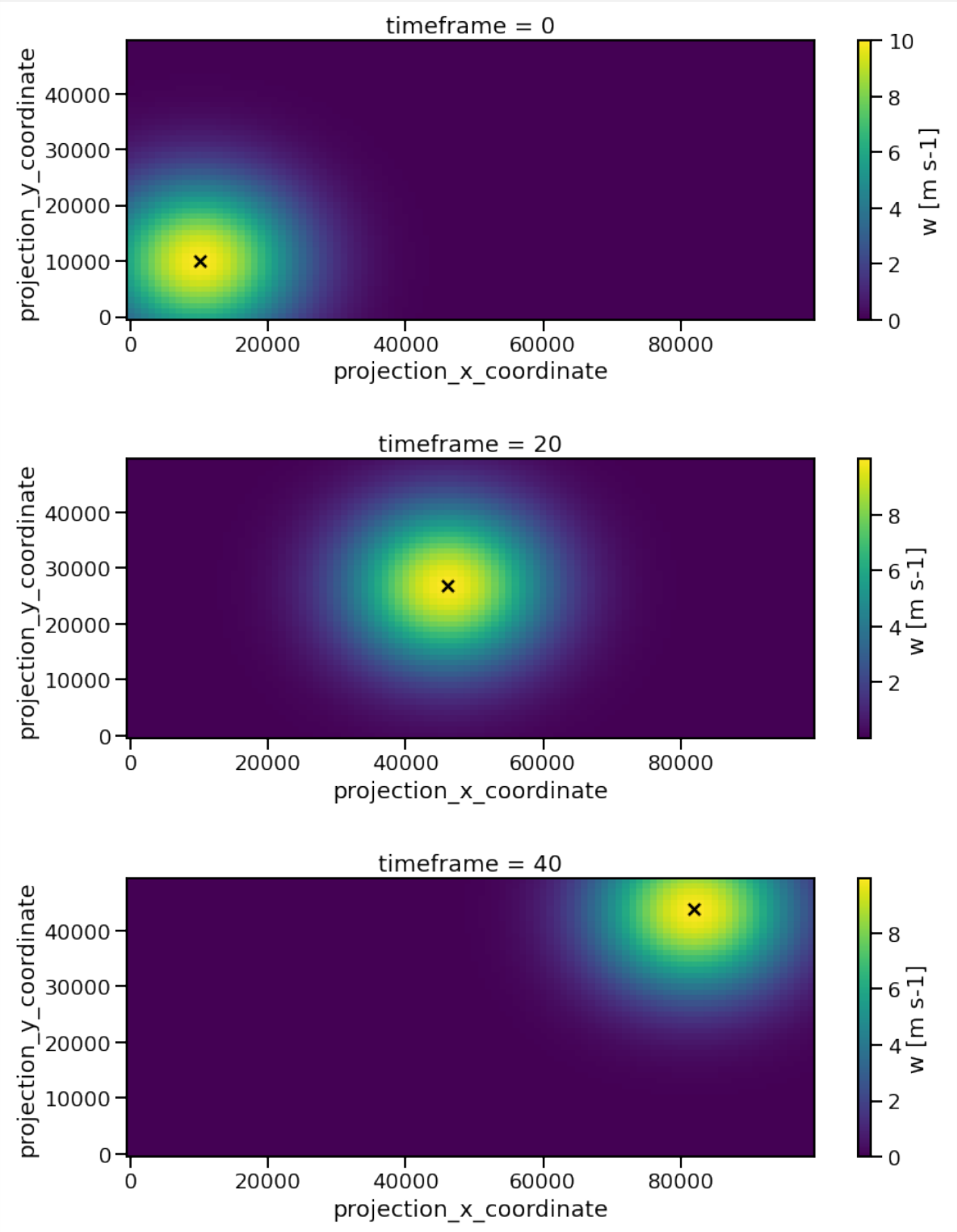
[11]:
```
features
```

[11]:
```
<xarray.Dataset>
Dimensions:                   (index: 51)
Coordinates:
  * index                     (index) int64 0 1 2 3 4 5 6 ... 45 46 47 48 49 50
Data variables: (12/13)
    frame                     (index) int64 0 1 2 3 4 5 6 ... 45 46 47 48 49 50
    idx                       (index) int64 1 1 1 1 1 1 1 1 ... 1 1 1 1 1 1 1 1
    hdim_1                    (index) float64 10.0 10.94 11.71 ... 48.15 48.6
    hdim_2                    (index) float64 10.0 11.85 13.66 ... 97.23 98.2
    num                       (index) int64 69 66 65 65 65 65 ... 39 34 23 13 5
    threshold_value           (index) int64 9 9 9 9 9 9 9 9 9 ... 9 9 9 9 9 9 9 9
    ...                        ...
    time                      (index) object 2000-01-01 12:00:00 ... 2000-01-0...
    timestr                   (index) object '2000-01-01 12:00:00' ... '2000-0...
    projection_y_coordinate   (index) float64 1e+04 1.094e+04 ... 4.86e+04
    projection_x_coordinate   (index) float64 1e+04 1.185e+04 ... 9.82e+04
    latitude                  (index) object 24.1 24.12 24.14 ... 24.97 24.98
    longitude                 (index) object 150.1 150.1 150.1 ... 150.5 150.5
```

The ouputs tells us that features were found in 51 **frame**s (index 0 to 50) of our data. The variable **idx** is 1 for every frames, which means that only 1 feature was found in every time step, as we expected. **hdim_1** and **hdim_2** are the position of this feature with respect to the y and x-indices.

We can plot the detected feature positions on top of the colored blob.

[12]:
```
fig, axs = plt.subplots(ncols=1, nrows=3, figsize=(12, 16), sharey=True)
plt.subplots_adjust(hspace=0.5)

for i, itime in enumerate([0, 20, 40]):

    # plot the 2D blob field in colors
    test_data.isel(time=itime).plot(ax=axs[i])

    # plot the detected feature as black cross
    f = features.sel(index=[itime])
    f.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=axs[i],
        color="black",
        marker="x",
    )

    axs[i].set_title(f"timeframe = {itime}")
```



The function has succesfully detected the maximum of our data in the individual timeframes.

2.4 3. Trajectory Linking

After we are done finding the features and associated segments for each frame it is necessary for further analysis to keep track of those elements troughout time. Linking is the tool for that. It connects the features of the timesteps which belong together. We are going to use the *linking_trackpy()* function here. The required inputs are the features, the two spacings and a maximum velocity of the features.

[13]:
```
trajectories = tobac.themes.tobac_v1.linking_trackpy(
    features, test_data, dt=dt, dxy=dxy, v_max=100
)
```

```
Frame 50: 1 trajectories present.
```

fails without v_max

Unsurprisingly, one trajectory was found. The returned object is another Dataset:

[14]:
```
trajectories
```

[14]:
```
<xarray.Dataset>
Dimensions:                  (index: 51)
Coordinates:
  * index                    (index) int64 0 1 2 3 4 5 6 ... 45 46 47 48 49 50
Data variables: (12/15)
    frame                    (index) int64 0 1 2 3 4 5 6 ... 45 46 47 48 49 50
    idx                      (index) int64 1 1 1 1 1 1 1 1 ... 1 1 1 1 1 1 1 1
    hdim_1                   (index) float64 10.0 10.94 11.71 ... 48.15 48.6
    hdim_2                   (index) float64 10.0 11.85 13.66 ... 97.23 98.2
    num                      (index) int64 69 66 65 65 65 65 ... 39 34 23 13 5
    threshold_value          (index) int64 9 9 9 9 9 9 9 9 9 ... 9 9 9 9 9 9 9 9
    ...                       ...
    projection_y_coordinate  (index) float64 1e+04 1.094e+04 ... 4.86e+04
    projection_x_coordinate  (index) float64 1e+04 1.185e+04 ... 9.82e+04
    latitude                 (index) object 24.1 24.12 24.14 ... 24.97 24.98
    longitude                (index) object 150.1 150.1 150.1 ... 150.5 150.5
    cell                     (index) float64 1.0 1.0 1.0 1.0 ... 1.0 1.0 1.0 1.0
    time_cell                (index) timedelta64[ns] 00:00:00 ... 00:50:00
```

The new variable cell now indexes the features in the different time steps. Therefore we can use it to create a mask for our moving feature:

[15]:
```
track_mask = trajectories["cell"] == 1.0
```

This mask can then be used - to select the track:

[16]:
```
track = trajectories.where(track_mask).dropna("index")
```

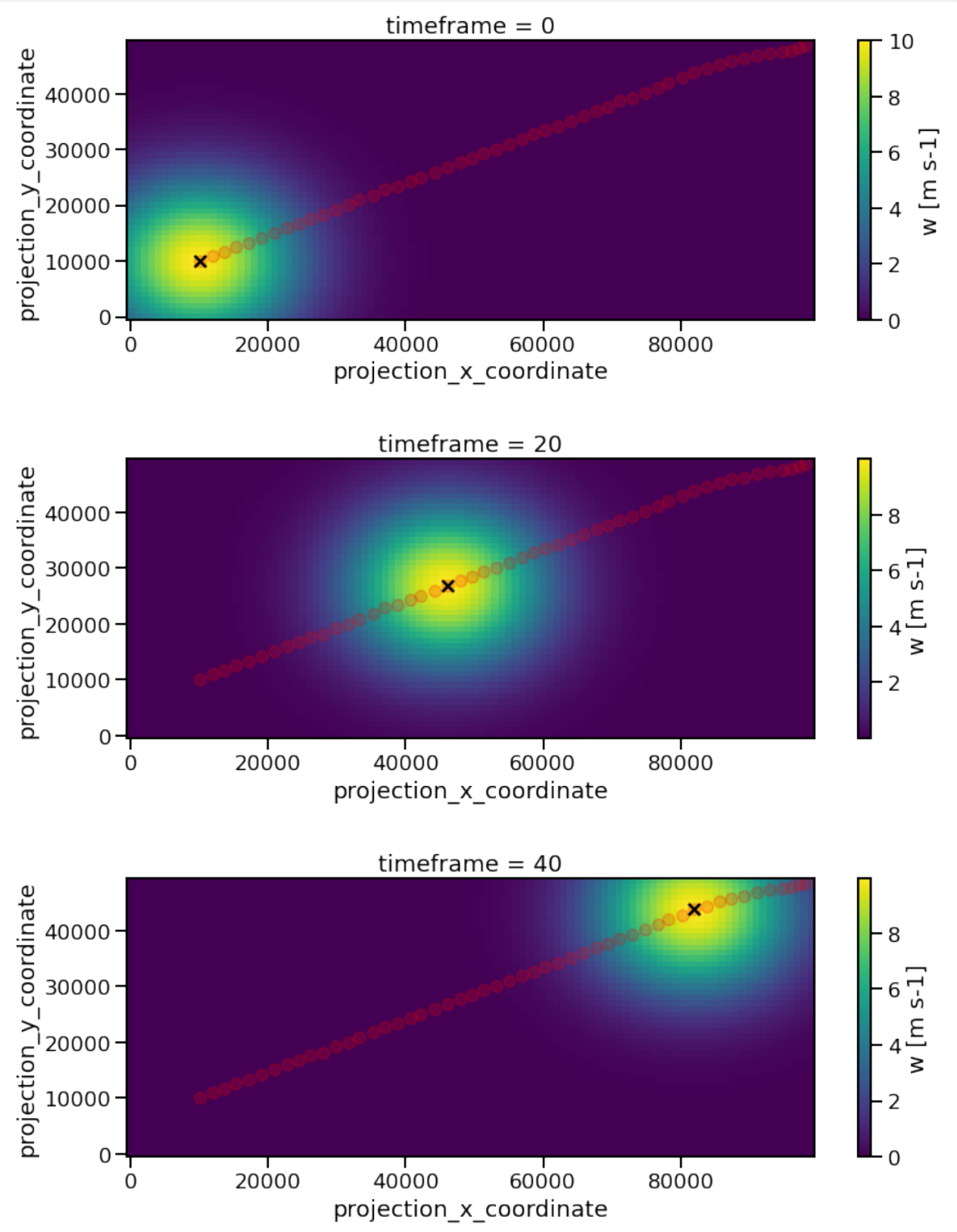
- and to show the track in our plots:

[17]:
```
fig, axs = plt.subplots(ncols=1, nrows=3, figsize=(12, 16), sharey=True)
plt.subplots_adjust(hspace=0.5)
# fig.tight_layout()


for i, itime in enumerate([0, 20, 40]):
```

(continues on next page)

(continued from previous page)

```
    # plot the 2D blob field in colors
    test_data.isel(time=itime).plot(ax=axs[i])

    # plot the track
    track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=axs[i],
        color="red",
        marker="o",
        alpha=0.2,
    )

    # plot the detected feature as black cross
    f = features.sel(index=[itime])
    f.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=axs[i],
        color="black",
        marker="x",
    )

    axs[i].set_title(f"timeframe = {itime}")
```


timeframe = 0
projection_y_coordinate
projection_x_coordinate
w [m s-1]
timeframe = 20
projection_y_coordinate
projection_x_coordinate
w [m s-1]
timeframe = 40
projection_y_coordinate
projection_x_coordinate
w [m s-1]

2.5 4. Segmentation

Another step after the feature detection and tracking is the segmentation of the data. This means we find the area surrounding our features belonging to the same cluster. Logically, we now need the already detected features as an additional input.

Just a small thought here:

We often introduce the different steps as 1.feature detection, 2.segmentation and 3. tracking, so having the segmenation step actually before the trajectory linking. The order actually does not matter since the current segmentation approach is based on the feature detection output only and can be done before or after the linking.

On further extensions:

- Also, one could actually use the output of the segmentation (`features_test` Dataset in the example below) as input for the tracking with the advantage that information on the area (ncells) is added in the dataframe.
- One could also use the output of the tracking in the segmentation (`trajectories` Dataset from above) with the advantage that mask will contain only the features that are also linked to trajectories.

These possibilities exist and could be utlized by you …

[18]:
```
%%capture
mask, features_test = tobac.themes.tobac_v1.segmentation(
    features, test_data, dxy, threshold=9
)
```

As the name implies, the first object returned is a Boolean mask that is true for all segments belonging to features. The second output is again the features of the field.

[19]:
```
mask
```

[19]:
```
<xarray.DataArray 'segmentation_mask' (time: 100, y: 50, x: 100)>
[500000 values with dtype=int32]
Coordinates:
  * time       (time) datetime64[ns] 2000-01-01T12:00:00 ... 2000-01-01T13:39:00
  * y          (y) float64 0.0 1e+03 2e+03 3e+03 ... 4.7e+04 4.8e+04 4.9e+04
  * x          (x) float64 0.0 1e+03 2e+03 3e+03 ... 9.7e+04 9.8e+04 9.9e+04
    latitude   (y, x) float64 ...
    longitude  (y, x) float64 ...
Attributes:
    long_name:  segmentation_mask
```

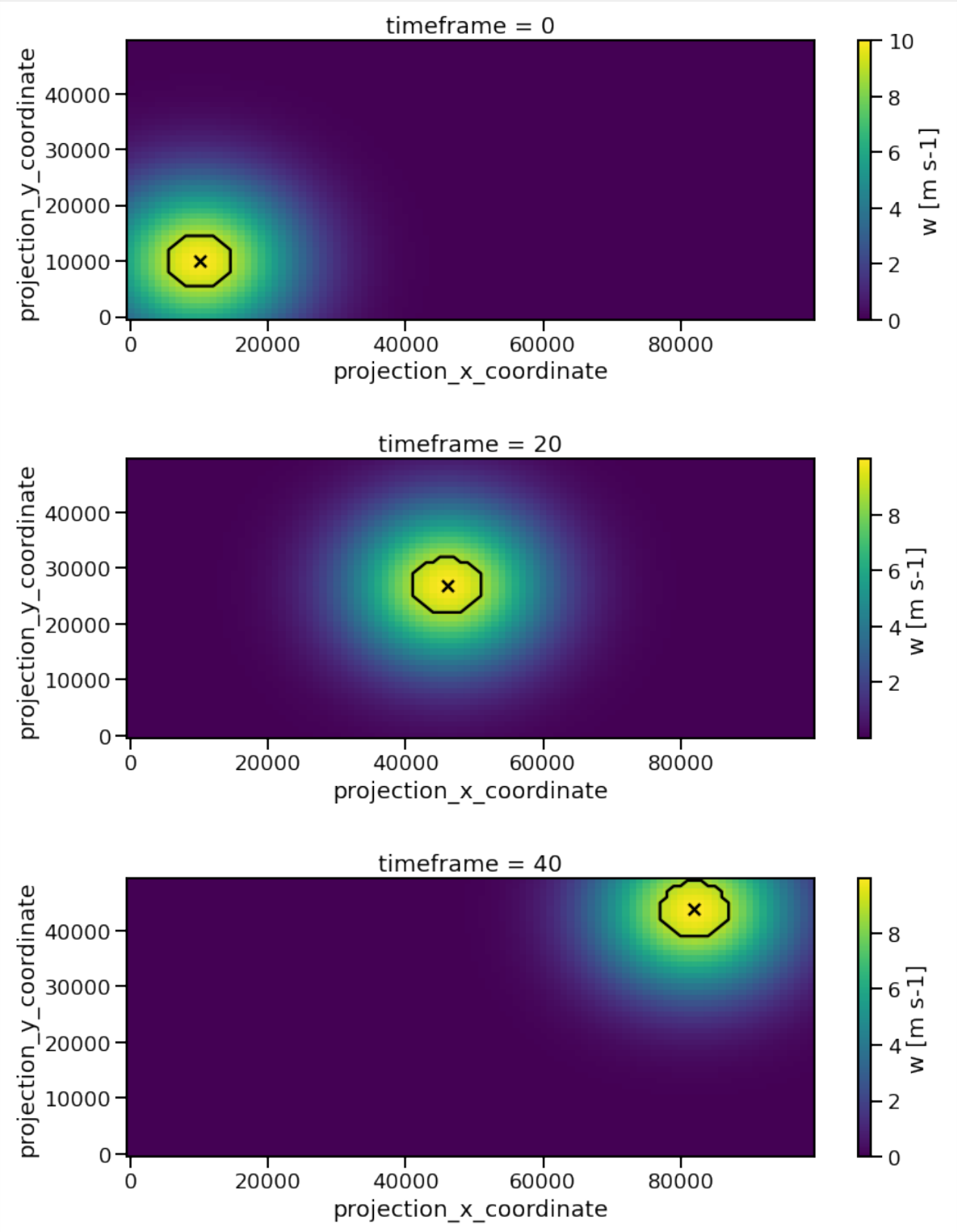
Since True/False corresponds to 1/0 in python, we can visualize the segments with a simple contour plot:

[20]:
```
fig, axs = plt.subplots(ncols=1, nrows=3, figsize=(12, 16), sharey=True)
plt.subplots_adjust(hspace=0.5)


for i, itime in enumerate([0, 20, 40]):

    # plot the 2D blob field in colors
    test_data.isel(time=itime).plot(ax=axs[i])

    # plot the mask outline
    mask.isel(time=itime).plot.contour(levels=[0.5], ax=axs[i], colors="k")
```

(continues on next page)

(continued from previous page)

```
    # plot the detected feature as black cross
    f = features.sel(index=[itime])
    f.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=axs[i],
        color="black",
        marker="x",
    )

    axs[i].set_title(f"timeframe = {itime}")
```



Keep in mind that the area of the resulting segments crucially depends on the defined thresholds.

2.6 5. Statistical Analysis

2.6.1 Blob Velocity / Motion Speed

Now different functions of tobacs analysis module can be used to calculate or plot properties of the tracks and the features. For example the velocity of the feature along the track can be calulated via:

[21]:
```
vel = tobac.analysis.calculate_velocity(track)
```

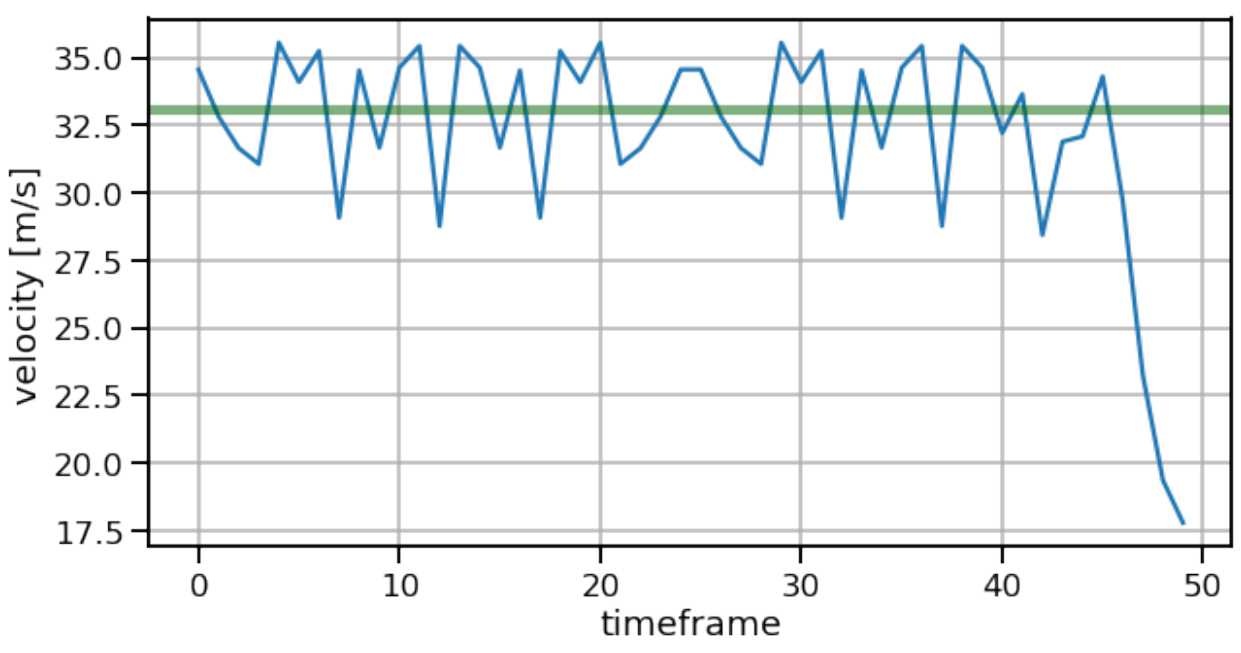
The expected velocity (hard-coded in the test function) is:

[22]:
```
expected_velocity = np.sqrt(30**2 + 14**2)
```

Plotting the velocity vs the timeframe will give us

[23]:
```
plt.figure(figsize=(10, 5))
plt.tight_layout()
plt.plot(vel["frame"], vel["v"])

plt.xlabel("timeframe")
plt.ylabel("velocity [m/s]")
plt.grid()

plt.axhline(expected_velocity, color="darkgreen", lw=5, alpha=0.5)
```

[23]:
```
<matplotlib.lines.Line2D at 0x7fe2abdd2890>
```

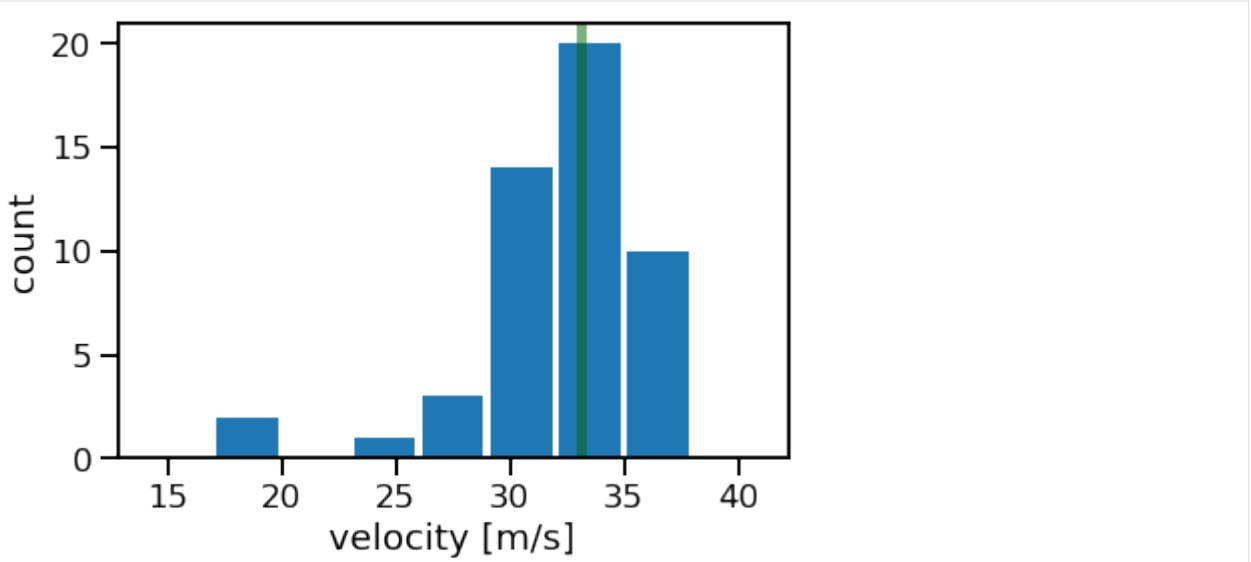


We can also create an histogram of the detected velocities:

[24]:
```
hist, edges = tobac.analysis.velocity_histogram(
    track,
    bin_edges=np.arange(14, 43, 3),
)
```

[25]:
```
width = 0.9 * (edges[1] - edges[0])
center = (edges[:-1] + edges[1:]) / 2

plt.tight_layout()
plt.bar(center, hist, width=width)
plt.ylabel("count")
plt.xlabel("velocity [m/s]")

plt.axvline(expected_velocity, color="darkgreen", lw=5, alpha=0.5)
```

[25]:
```
<matplotlib.lines.Line2D at 0x7fe2abcf2e00>
```

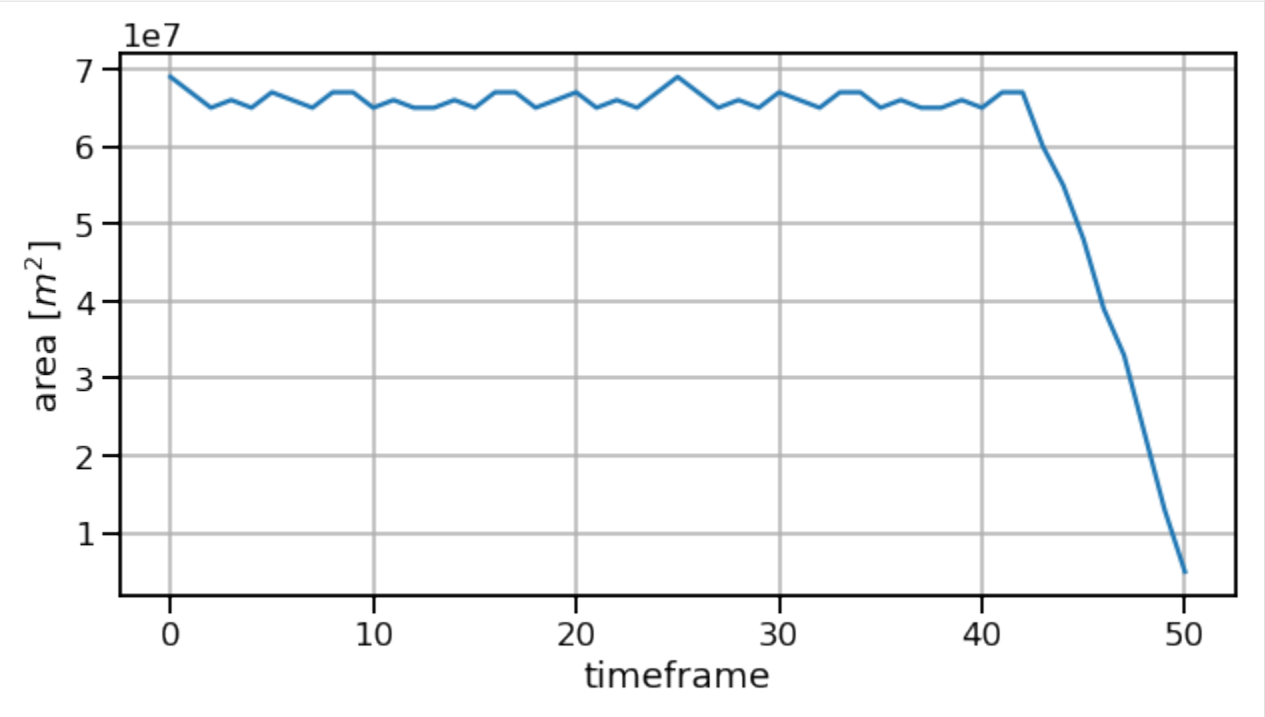


2.6.2 Area

The area of the features can also be calculated and plotted throughout time:

[26]:
```
area = tobac.analysis.calculate_area(features, mask)
```

[27]:
```
plt.figure(figsize=(10, 5))
plt.tight_layout()
plt.plot(area["frame"], area["area"])
plt.xlabel("timeframe")
plt.ylabel(r"area [$m^2$]")
plt.grid()
```

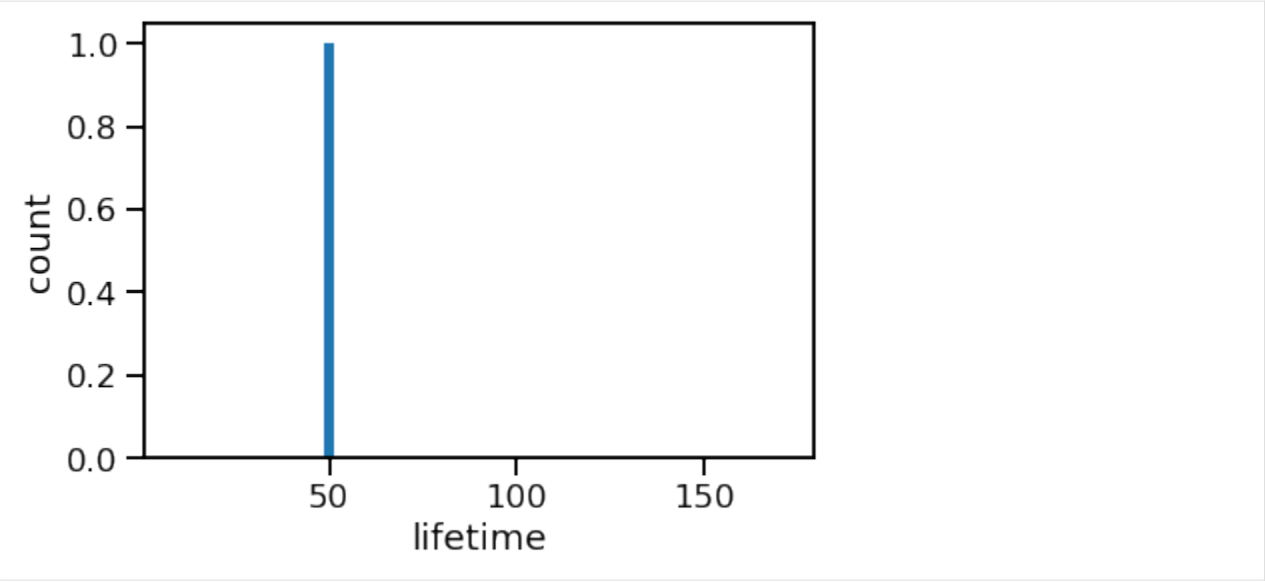


2.6.3 Lifetime

Another interesting property for real data are the lifetimes of our features. Tobac can also produce a histogram of this:

[28]:
```
hist, bins, centers = tobac.analysis.lifetime_histogram(
    track, bin_edges=np.arange(0, 200, 20)
)
```

[29]:
```
plt.tight_layout()
plt.bar(centers, hist, width=width)
plt.ylabel("count")
plt.xlabel("lifetime")
plt.show()
```



We can deduce, that our singular feature in the data has a lifetime of 50.

CHAPTER

THREE

IDEALIZED CASE 2: TWO CROSSING BLOBS

This tutorial explores the different methods of the linking process using an example of two crossing blobs. The following chapters will be covered:

1. *Data generation*
2. *Feature detection*
3. *Influence of tracking method*
4. *Analysis*

3.1 1. Data generation

We start by importing the usual libraries and adjusting some settings:

[1]:
```
import tobac
import numpy as np
import matplotlib.pyplot as plt
import datetime
import xarray as xr
import seaborn as sns

sns.set_context("talk")
%matplotlib inline
```

We will need to generate our own dataset for this tutorial. For this reason we define some bounds for our system:

[2]:
```
(
    x_min,
    y_min,
    x_max,
    y_max,
) = (
    0,
    0,
    1e5,
    1e5,
)
t_min, t_max = 0, 10000
```

We use these to create a mesh:

[3]:
```
def create_mesh(x_min, y_min, x_max, y_max, t_min, t_max, N_x=200, N_y=200, dt=520):
    x = np.linspace(x_min, x_max, N_x)
    y = np.linspace(y_min, y_max, N_y)
    t = np.arange(t_min, t_max, dt)
    mesh = np.meshgrid(t, y, x, indexing="ij")

    return mesh


mesh = create_mesh(x_min, y_min, x_max, y_max, t_min, t_max)
```

Additionally, we need to set velocities for our blobs:

[4]:
```
v_x = 10
v_y = 10
```

The dataset is created by using two functions. The first creates a wandering Gaussian blob as `numpy`-Array on our grid and the second transforms it into an `xarray`-DataArray with an arbitrary `datetime`.

[5]:
```
def create_wandering_blob(mesh, x_0, y_0, v_x, v_y, t_create, t_vanish, sigma=1e7):
    tt, yy, xx = mesh
    exponent = (xx - x_0 - v_x * (tt - t_create)) ** 2 + (
        yy - y_0 - v_y * (tt - t_create)
    ) ** 2
    blob = np.exp(-exponent / sigma)
    blob = np.where(np.logical_and(tt >= t_create, tt <= t_vanish), blob, 0)

    return blob


def create_xarray(array, mesh, starting_time="2022-04-01T00:00"):
    tt, yy, xx = mesh
    t = np.unique(tt)
    y = np.unique(yy)
    x = np.unique(xx)

    N_t = len(t)
    dt = np.diff(t)[0]

    t_0 = np.datetime64(starting_time)
    t_delta = np.timedelta64(dt, "s")

    time = np.array([t_0 + i * t_delta for i in range(len(array))])

    dims = ("time", "projection_x_coordinate", "projection_y_coordinate")
    coords = {"time": time, "projection_x_coordinate": x, "projection_y_coordinate": y}
    attributes = {"units": ("m s-1")}

    data = xr.DataArray(data=array, dims=dims, coords=coords, attrs=attributes)
    # data = data.projection_x_coordinate.assign_attrs({"units": ("m")})
    # data = data.projection_y_coordinate.assign_attrs({"units": ("m")})
    data = data.assign_coords(latitude=("projection_x_coordinate", x / x.max() * 90))
    data = data.assign_coords(longitude=("projection_y_coordinate", y / y.max() * 90))
```

(continues on next page)

(continued from previous page)

```
    return data
```

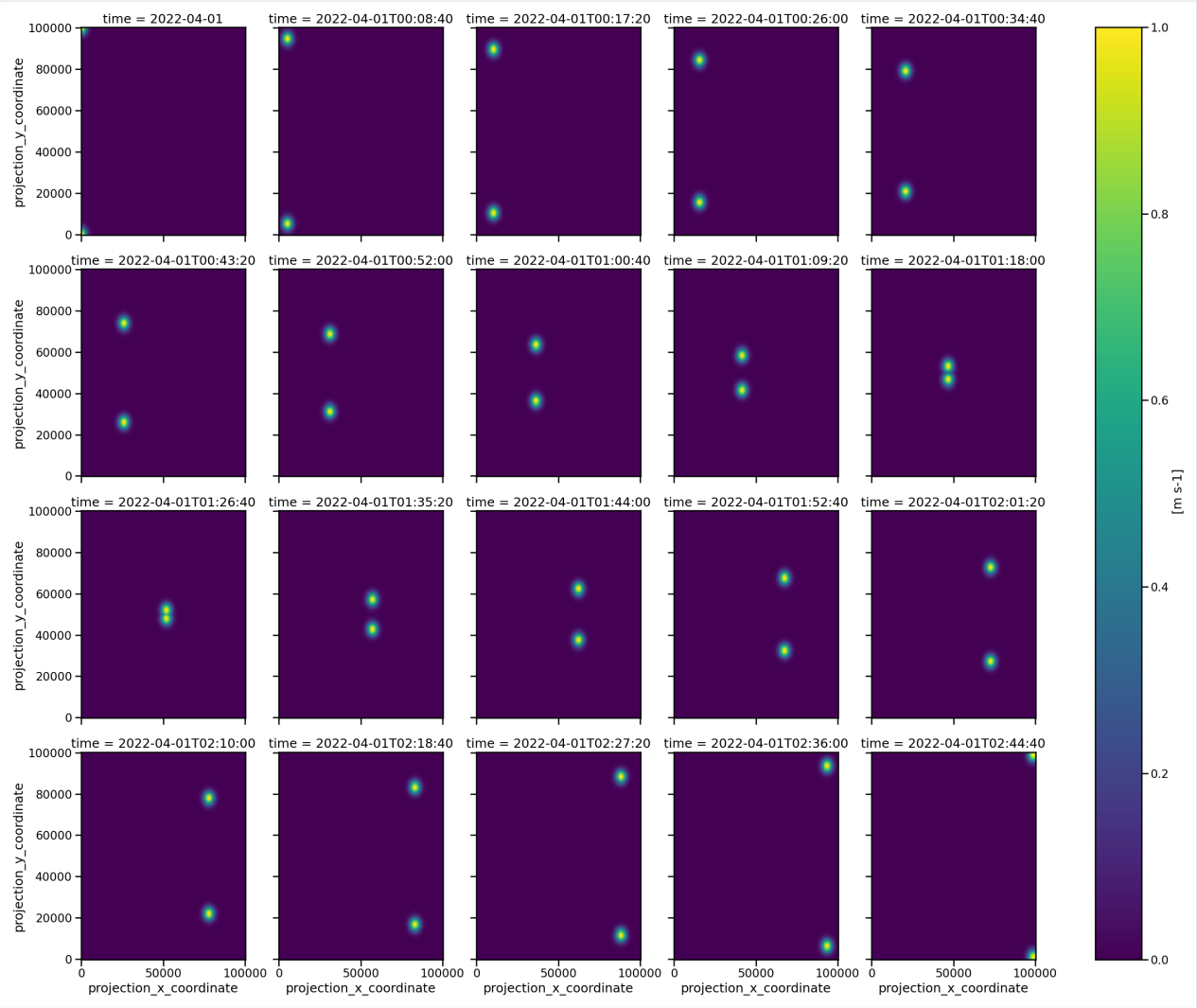
We use the first function to create two blobs whose paths will cross. To keep them detectable as seperate features in the dataset we don't want to just add them together. Instead we are going to use the highest value of each pixel by applying boolean masking and the resulting field is transformed into the `xarray` format.

[6]:
```
blob_1 = create_wandering_blob(mesh, x_min, y_min, v_x, v_y, t_min, t_max)
blob_2 = create_wandering_blob(mesh, x_max, y_min, -v_x, v_y, t_min, t_max)
blob_mask = blob_1 > blob_2
blob = np.where(blob_mask, blob_1, blob_2)

data = create_xarray(blob, mesh)
```

Let's check if we achived what we wanted by plotting the result:

[7]:
```
data.plot(
    cmap="viridis",
    col="time",
    col_wrap=5,
    x="projection_x_coordinate",
    y="projection_y_coordinate",
    size=5,
)
```

[7]:
```
<xarray.plot.facetgrid.FacetGrid at 0x7f53ff739210>
```



Looks good! We see two features crossing each other, and they are clearly separable in every frame.

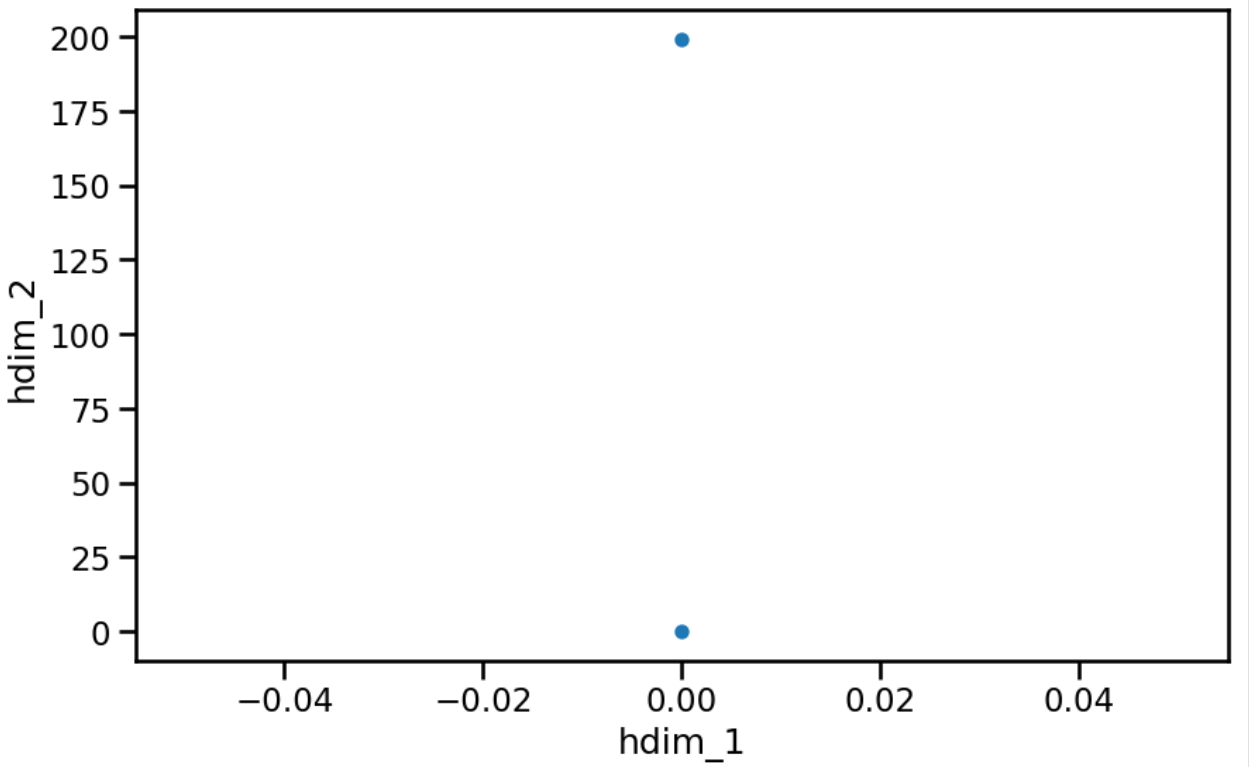
3.2 2. Feature detection

Before we can perform the tracking we need to detect the features with the usual function. The grid spacing is deduced from the generated field. We still need to find a reasonable threshold value. Let's try a really high one:

[8]:
```
%%capture

spacing = np.diff(np.unique(mesh[1]))[0]

dxy, dt = tobac.utils.get_spacings(data, grid_spacing=spacing)
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, threshold=0.99
)
```

[9]:
```
plt.figure(figsize=(10, 6))
features.plot.scatter(x="hdim_1", y="hdim_2")
```

[9]:
```
<matplotlib.collections.PathCollection at 0x7f53f8265d10>
```

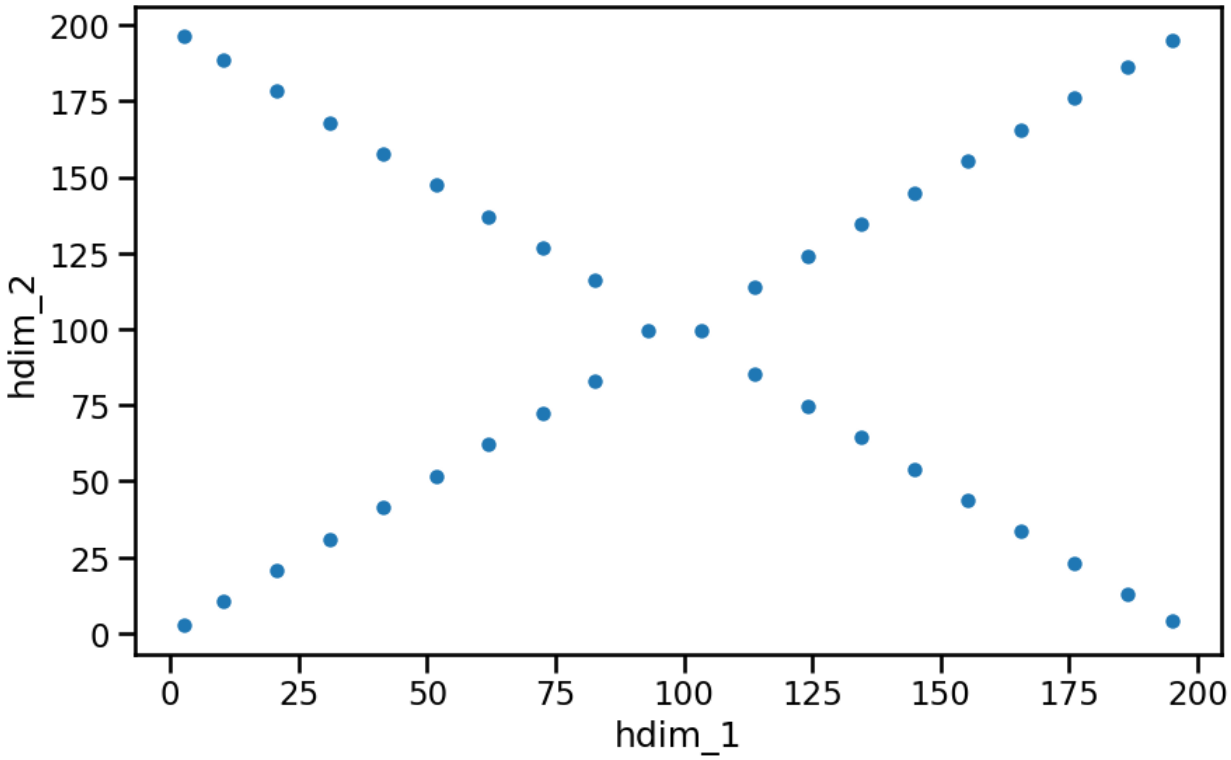


As you can see almost no features are detected. This means our threshold is too high and neglects many datapoints. Therefore it is a good idea to try a low threshold value:

[10]:
```
%%capture
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, threshold=0.3
)
```

[11]:
```
plt.figure(figsize=(10, 6))
features.plot.scatter(x="hdim_1", y="hdim_2")
```

[11]:
```
<matplotlib.collections.PathCollection at 0x7f5428f6b350>
```

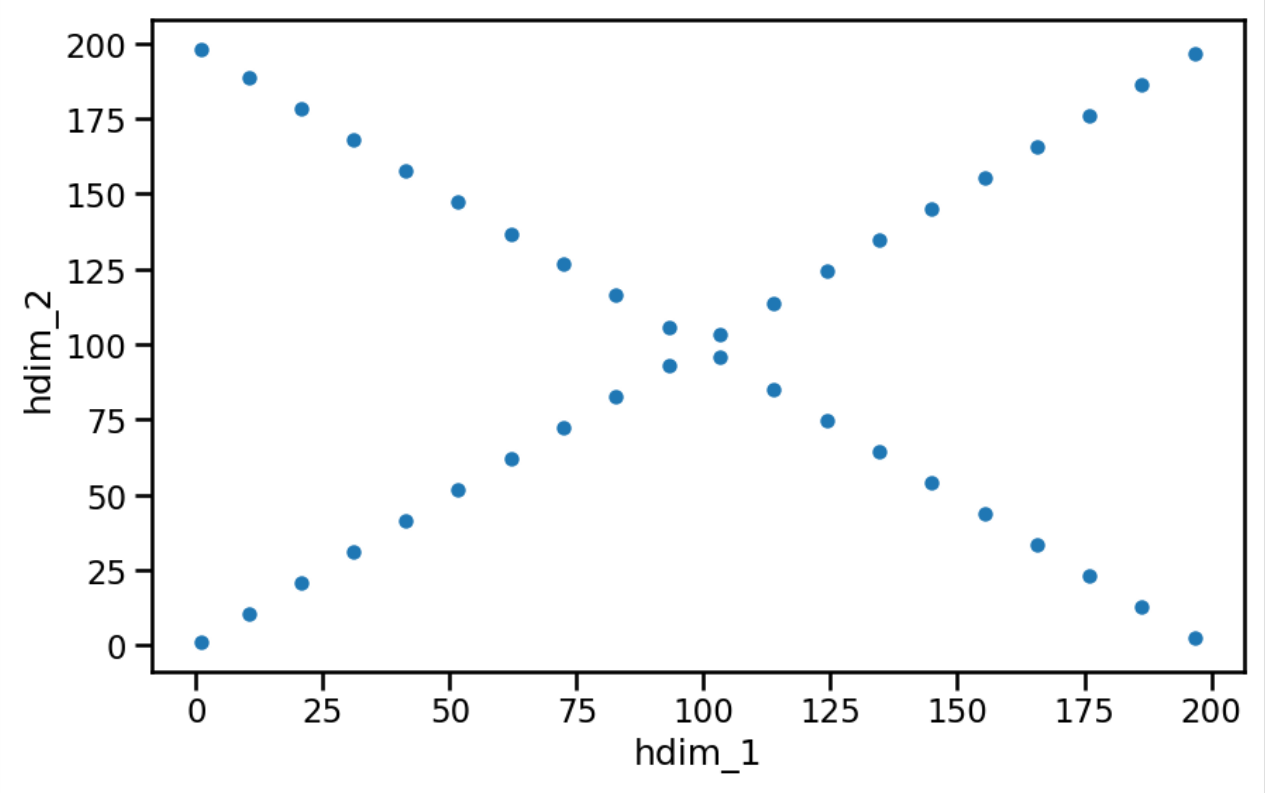


Here the paths of the blobs are clearly visible, but there is an area in the middle where both merge into one feature. This should be avoided. Therefore we try another value for the threshold somewhere in the middle of the available range:

[12]:
```
%%capture
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, threshold=0.8
)
```

[13]:
```
plt.figure(figsize=(10, 6))
features.plot.scatter(x="hdim_1", y="hdim_2")
```

[13]:
```
<matplotlib.collections.PathCollection at 0x7f53f3aeb290>
```



This is the picture we wanted to see. This means we can continue working with this set of features.

3.3 3. Influence of the tracking method

Now the tracking can be performed. We will create two outputs, one with `method = 'random'`, and the other one with `method = 'predict'`. Since we know what the velocities of our features are beforehand, we can select a reasonable value for `v_max`. Normally this would need to be finetuned.

[14]:
```
%matplotlib inline
v_max = 20

track_1 = tobac.themes.tobac_v1.linking_trackpy(
    features, data, dt, dxy, v_max=v_max, method_linking="random"
)

track_2 = tobac.themes.tobac_v1.linking_trackpy(
    features, data, dt, dxy, v_max=v_max, method_linking="predict"
)
```

```
Frame 19: 2 trajectories present.
```

```
/home/nils/mambaforge/envs/tob_v2/lib/python3.11/site-packages/tobac/themes/tobac_v1/
↪tracking.py:269: FutureWarning: Not prepending group keys to the result index of␣
↪transform-like apply. In the future, the group keys will be included in the index,␣
↪regardless of whether the applied function returns a like-indexed object.
To preserve the previous behavior, use
```

(continues on next page)

(continued from previous page)

```
        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  ).time.apply(lambda x: x - x.iloc[0])
```

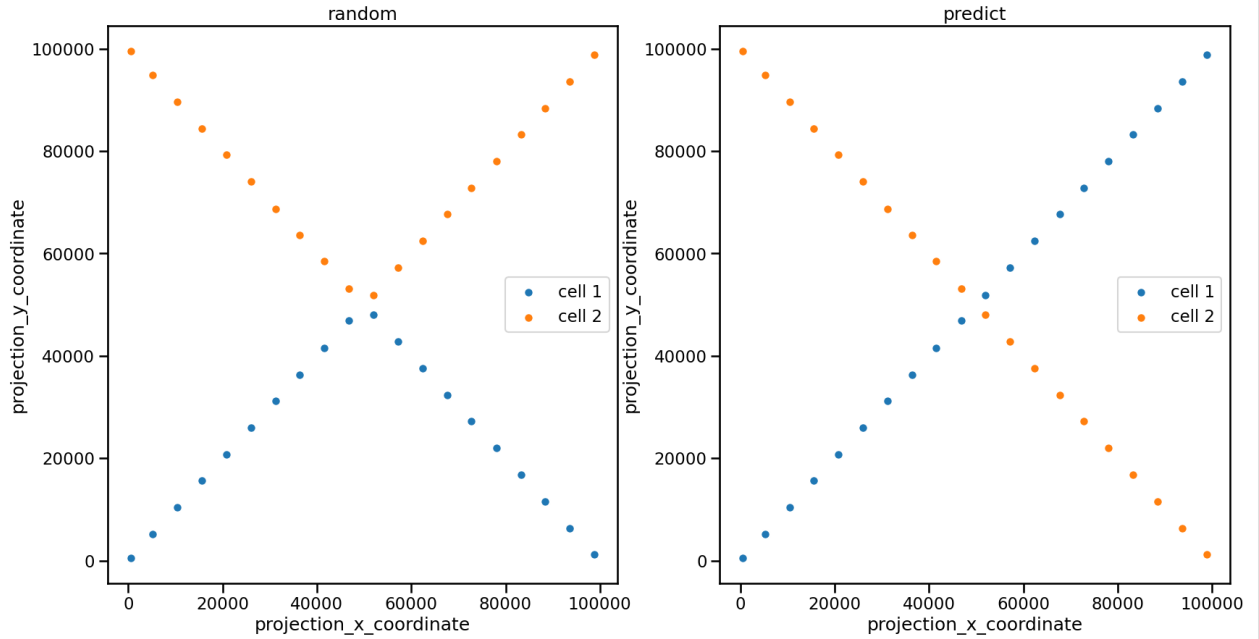
[15]:
```
track_1
```

[15]:
```
<xarray.Dataset>
Dimensions:                  (index: 40)
Coordinates:
  * index                    (index) int64 0 1 2 3 4 5 6 ... 34 35 36 37 38 39
Data variables: (12/15)
    frame                    (index) int64 0 0 1 1 2 2 3 ... 17 17 18 18 19 19
    idx                      (index) int64 1 2 1 2 1 2 1 2 2 ... 1 2 1 2 1 2 1 2
    hdim_1                   (index) float64 1.0 1.0 10.32 ... 186.3 196.7 196.7
    hdim_2                   (index) float64 1.0 198.0 10.32 ... 2.321 196.7
    num                      (index) int64 9 9 28 28 28 28 ... 24 24 28 28 28 28
    threshold_value          (index) float64 0.8 0.8 0.8 0.8 ... 0.8 0.8 0.8 0.8
    ...                       ...
    projection_x_coordinate  (index) float64 502.5 502.5 ... 9.883e+04 9.883e+04
    projection_y_coordinate  (index) float64 502.5 9.95e+04 ... 9.883e+04
    latitude                 (index) float64 0.4523 0.4523 4.668 ... 88.95 88.95
    longitude                (index) float64 0.4523 89.55 4.668 ... 1.05 88.95
    cell                     (index) float64 1.0 2.0 1.0 2.0 ... 1.0 2.0 1.0 2.0
    time_cell                (index) timedelta64[ns] 00:00:00 ... 02:44:40
```

Let's have a look at the resulting tracks:

[16]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(20, 10))

for i, cell_track in track_1.groupby("cell"):
    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax1,
        marker="o",
        label="cell {0}".format(int(i)),
    )
ax1.legend()
ax1.set_title("random")

for i, cell_track in track_2.groupby("cell"):
    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax2,
        marker="o",
        label="cell {0}".format(int(i)),
    )
ax2.legend()
ax2.set_title("predict")
```

[16]:
```
Text(0.5, 1.0, 'predict')
```



As you can see, there is a clear difference. While in the first link output the feature positions in the top half of the graph are linked into one cell, in the second output the path of the cell follows the actual way of the Gaussian blobs we created. This is possible because `method = "predict"` uses an extraploation to infer the next position from the previous timeframes.

3.4 4. Analysis

We know that the second option is “correct”, because we created the data. But can we also decude this by analyzing our tracks?

Let’s calculate the values for the velocities:

[17]:
```
track_1 = tobac.analysis.calculate_velocity(track_1)
track_2 = tobac.analysis.calculate_velocity(track_2)

v1 = track_1.where(track_1["cell"] == 1).dropna().v.values
v2 = track_2.where(track_1["cell"] == 1).dropna().v.values
```

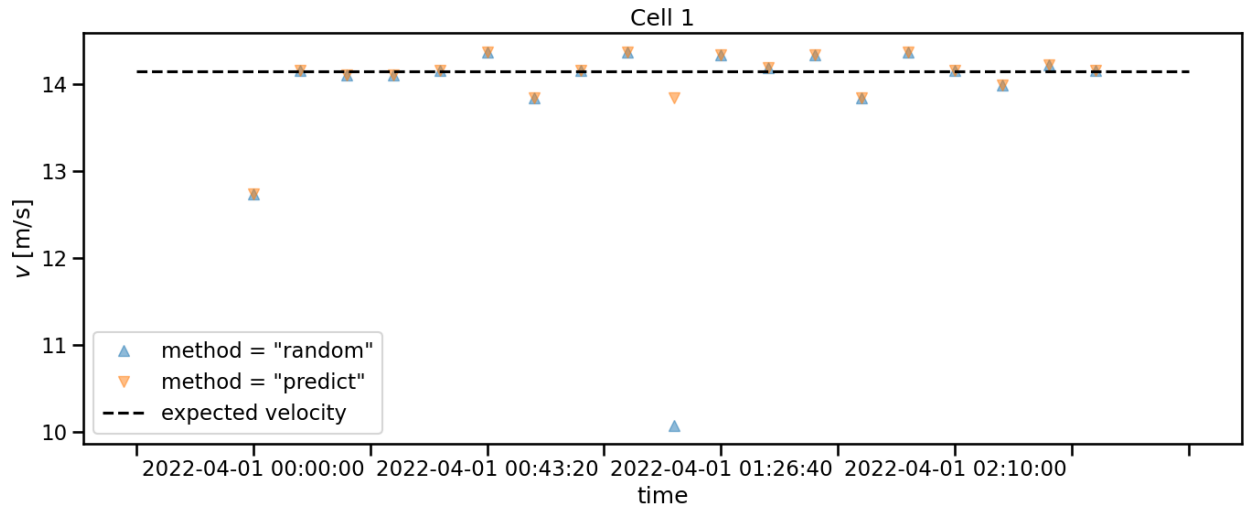
Vizualizing these can help us with our investigation:

[18]:
```
fig, (ax) = plt.subplots(figsize=(14, 6))

ax.set_title("Cell 1")

mask_1 = track_1["cell"] == 1
mask_2 = track_2["cell"] == 1

track_1.where(mask_1).dropna().plot(
    x="time",
```

(continues on next page)

(continued from previous page)

```
    y="v",
    ax=ax,
    label='method = "random"',
    marker="^",
    linestyle="",
    alpha=0.5,
)
track_2.where(mask_2).dropna().plot(
    x="time",
    y="v",
    ax=ax,
    label='method = "predict"',
    marker="v",
    linestyle="",
    alpha=0.5,
)

ticks = ax.get_xticks()

plt.hlines(
    [np.sqrt(v_x**2 + v_y**2)],
    ticks.min(),
    ticks.max(),
    color="black",
    label="expected velocity",
    linestyle="--",
)


ax.set_ylabel("$v$ [m/s]")
plt.legend()
plt.tight_layout()
```



The expected velocity is just added for reference. But also without looking at this, we can see that the values for `method = "random"` have an outlier, that deviates far from the other values. This is a clear sign, that this method is not suited well for this case.

CHAPTER

FOUR

METHODS AND PARAMETERS FOR FEATURE DETECTION: PART 1

In this notebook, we will take a detailed look at tobac's feature detection and examine some of its parameters. We concentrate on:

- Minima/Maxima and Multiple Thresholds for Feature Identification
- *Feature Position*
- *Sigma Parameter for Smoothing of Noisy Data*
- *Band Pass Filter for Input Fields*

[1]:
```
import tobac
import matplotlib.pyplot as plt
import numpy as np
import xarray as xr

%matplotlib inline

import seaborn as sns

sns.set_context("talk")
```

4.1 Minima/Maxima and Multiple Thresholds for Feature Identification

Feature identification search for local maxima in the data.

When working different inputs it is sometimes necessary to switch the feature detection from finding maxima to minima. Furthermore, for more complex datasets containing multiple features differing in magnitude, a categorization according to this magnitude is desirable. Both will be demonstrated with the `make_sample_data_2D_3blobs()` function, which creates such a dataset. For the search for minima we will simply turn the dataset negative:

[2]:
```
data = tobac.testing.make_sample_data_2D_3blobs()
neg_data = -data
```

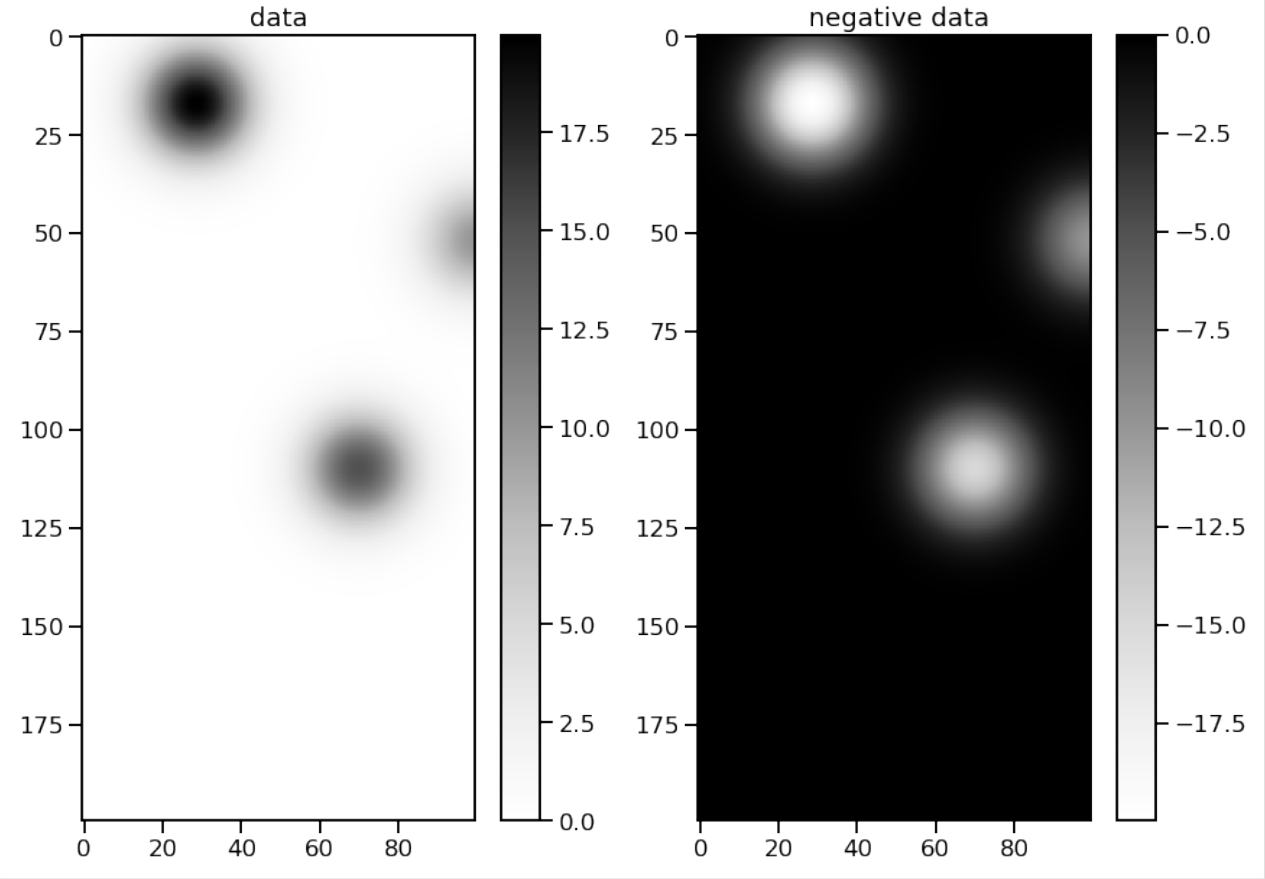
Let us have a look at frame number 50:

[3]:
```
n = 50

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(14, 10))

im1 = ax1.imshow(data.data[50], cmap="Greys")
```

(continues on next page)

(continued from previous page)

```
ax1.set_title("data")
cbar = plt.colorbar(im1, ax=ax1)

im2 = ax2.imshow(neg_data.data[50], cmap="Greys")
ax2.set_title(" negative data")
cbar = plt.colorbar(im2, ax=ax2)
```



As you can see the data has 3 maxima/minima with different extremal values. To capture these, we use list comprehensions to obtain multiple thresholds in the range of the data:

[4]:
```
thresholds = [i for i in range(9, 18)]
neg_thresholds = [-i for i in range(9, 18)]
```

These can now be passed as arguments to `feature_detection_multithreshold()`. With the `target`-keyword we can set a flag whether to search for minima or maxima. The standard is `"maxima"`.

[5]:
```
%%capture

dxy, dt = tobac.utils.get_spacings(data)

features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, thresholds, target="maximum"
)
features_inv = tobac.themes.tobac_v1.feature_detection_multithreshold(
```

(continues on next page)

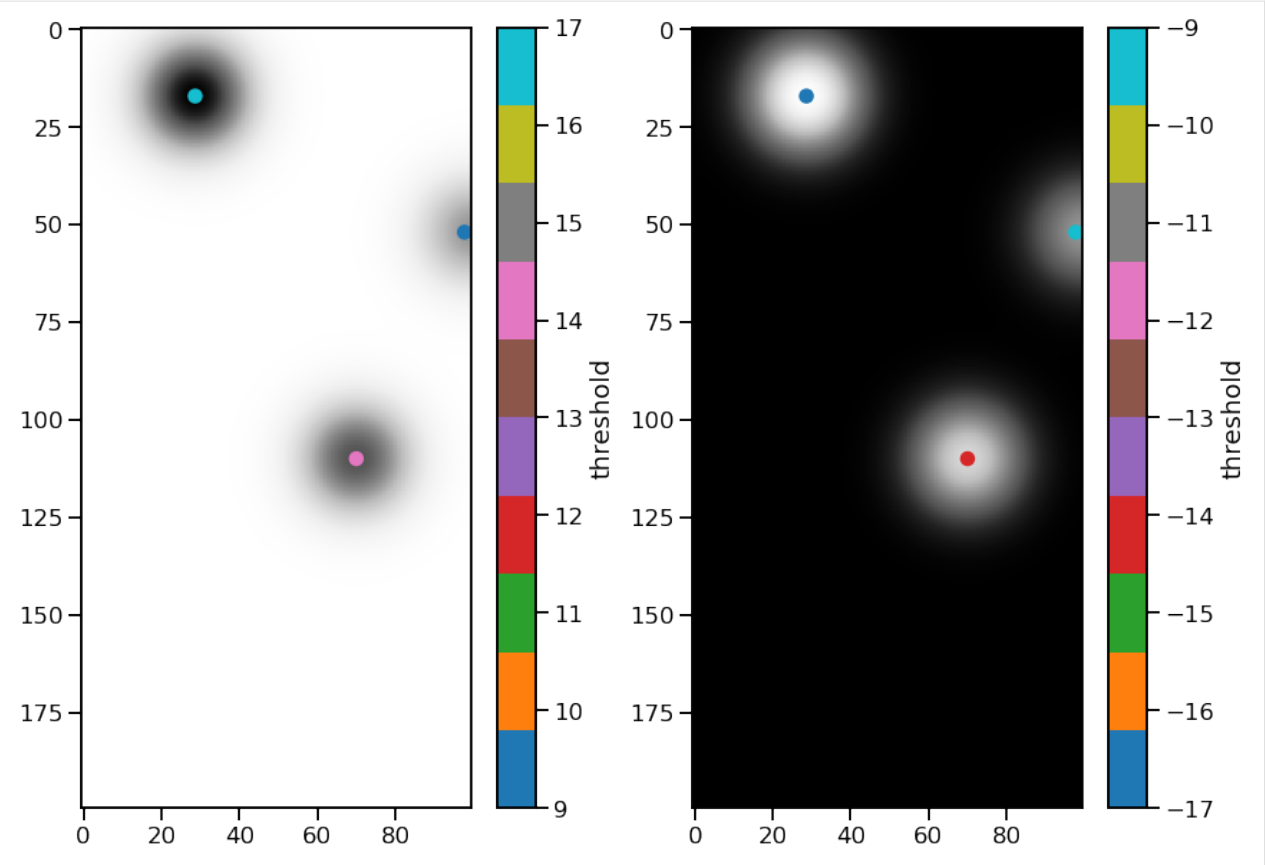
(continued from previous page)

```
    neg_data, dxy, neg_thresholds, target="minimum"
)
```

Let's scatter the detected features onto frame 50 of the dataset and create colorbars for the threshold values:

[6]:
```
mask_1 = features["frame"] == n
mask_2 = features_inv["frame"] == n

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(14, 10))

ax1.imshow(data.data[50], cmap="Greys")
im1 = ax1.scatter(
    features.where(mask_1)["hdim_2"],
    features.where(mask_1)["hdim_1"],
    c=features.where(mask_1)["threshold_value"],
    cmap="tab10",
)
cbar = plt.colorbar(im1, ax=ax1)
cbar.ax.set_ylabel("threshold")

ax2.imshow(neg_data.data[50], cmap="Greys")
im2 = ax2.scatter(
    features_inv.where(mask_2)["hdim_2"],
    features_inv.where(mask_2)["hdim_1"],
    c=features_inv.where(mask_2)["threshold_value"],
    cmap="tab10",
)
cbar = plt.colorbar(im2, ax=ax2)
cbar.ax.set_ylabel("threshold")
```

[6]:
```
Text(0, 0.5, 'threshold')
```



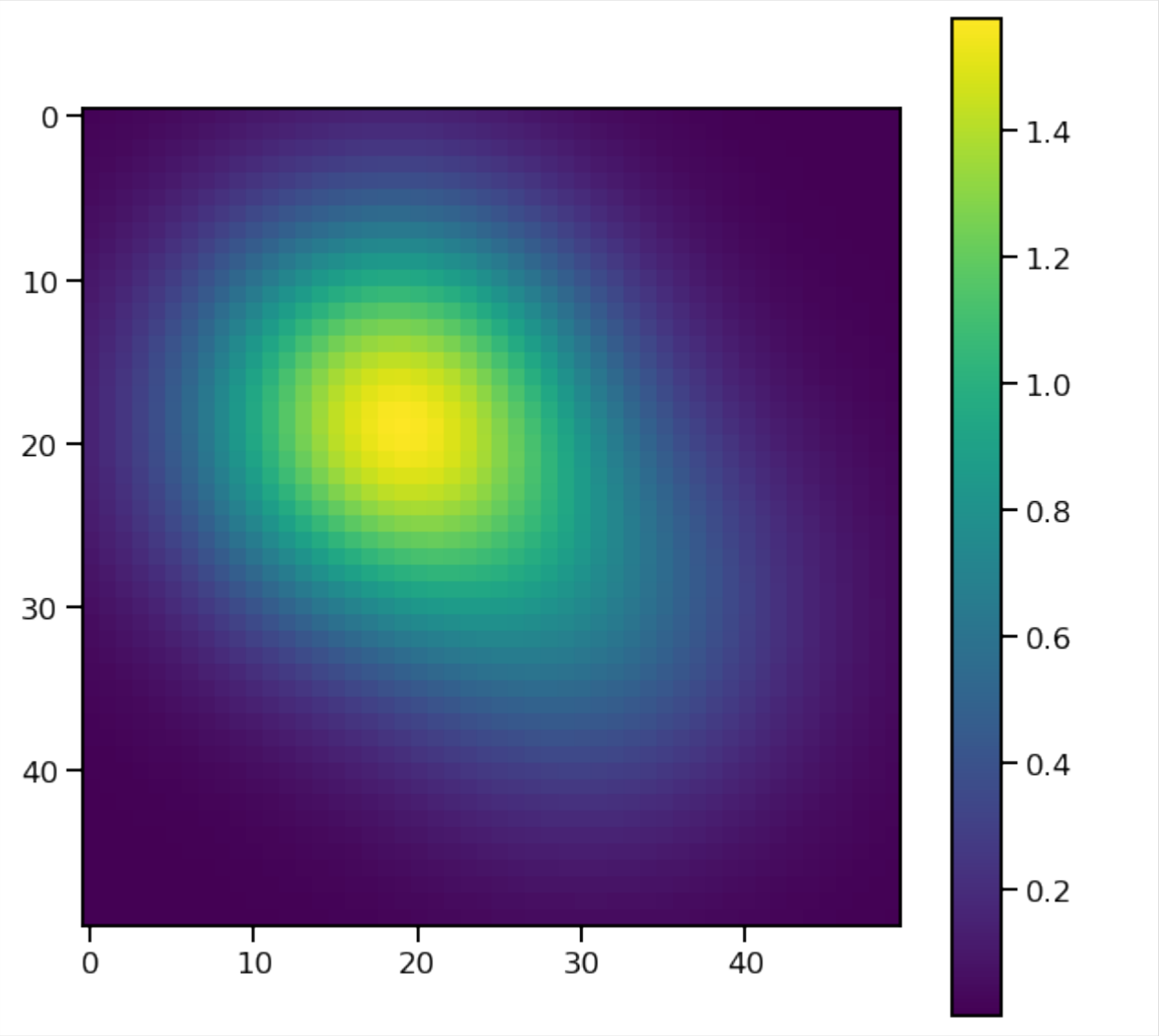
The three features were found in both data sets, and the color bars indicate which threshold they belong to. When using multiple thresholds, note that the order of the list is important. Each feature is assigned the threshold value that was reached last. Therefore, it makes sense to start with the lowest value in case of maxima.

4.2 Feature Position

To explore the influence of the `position_threshold` flag we need a radially asymmetric feature. Let's create a simple one by adding two 2d-gaussians and add an extra dimension for the time, which is required for working with tobac:

[7]:
```
x = np.linspace(-2, 2)
y = np.linspace(-2, 2)
xx, yy = np.meshgrid(x, y)

exp1 = 1.5 * np.exp(-((xx + 0.5) ** 2 + (yy + 0.5) ** 2))
exp2 = 0.5 * np.exp(-((0.5 - xx) ** 2 + (0.5 - yy) ** 2))

asymmetric_data = np.expand_dims(exp1 + exp2, axis=0)

plt.figure(figsize=(10, 10))
plt.imshow(asymmetric_data[0])
plt.colorbar()
```

[7]:
```
<matplotlib.colorbar.Colorbar at 0x7f49d17f6da0>
```



To feed this data into the feature detection we need to convert it into an `xarray.DataArray`. Before we do that we select an arbitrary time and date for the single frame of our synthetic field:

[8]:
```
date = np.datetime64(
    "2022-04-01T00:00",
)
assym = xr.DataArray(
    data=asymmetric_data, coords={"time": np.expand_dims(date, axis=0), "y": y, "x": x}
)
assym
```

[8]:
```
<xarray.DataArray (time: 1, y: 50, x: 50)>
array([[[0.01666536, 0.02114886, 0.02648334, ..., 0.00083392,
         0.00058509, 0.00040694],
        [0.02114886, 0.02683866, 0.03360844, ..., 0.00109464,
         0.00077154, 0.0005393 ],
        [0.02648334, 0.03360844, 0.04208603, ..., 0.00142435,
```

(continues on next page)

(continued from previous page)

```
         0.00100896, 0.00070907],
        ...,
        [0.00083392, 0.00109464, 0.00142435, ..., 0.01405279,
         0.01121917, 0.00883872],
        [0.00058509, 0.00077154, 0.00100896, ..., 0.01121917,
         0.00895731, 0.00705704],
        [0.00040694, 0.0005393 , 0.00070907, ..., 0.00883872,
         0.00705704, 0.00556009]]])
Coordinates:
  * time     (time) datetime64[ns] 2022-04-01
  * y        (y) float64 -2.0 -1.918 -1.837 -1.755 ... 1.755 1.837 1.918 2.0
  * x        (x) float64 -2.0 -1.918 -1.837 -1.755 ... 1.755 1.837 1.918 2.0
```

Since we do not have a dt in this dataset, we can not use the `get_spacings()`-utility this time and need to calculate the `dxy` spacing manually:

[9]:
```
dxy = assym.diff("x")
```

Finally, we choose a threshold in the datarange and apply the feature detection with the four `position_threshold` flags - 'center' - 'extreme' - 'weighted_diff' - 'weighted_abs'

[10]:
```
%%capture

threshold = 0.2
features_center = tobac.themes.tobac_v1.feature_detection_multithreshold(
    assym, dxy, threshold, position_threshold="center"
)
features_extreme = tobac.themes.tobac_v1.feature_detection_multithreshold(
    assym, dxy, threshold, position_threshold="extreme"
)
features_diff = tobac.themes.tobac_v1.feature_detection_multithreshold(
    assym, dxy, threshold, position_threshold="weighted_diff"
)
features_abs = tobac.themes.tobac_v1.feature_detection_multithreshold(
    assym, dxy, threshold, position_threshold="weighted_abs"
)
```

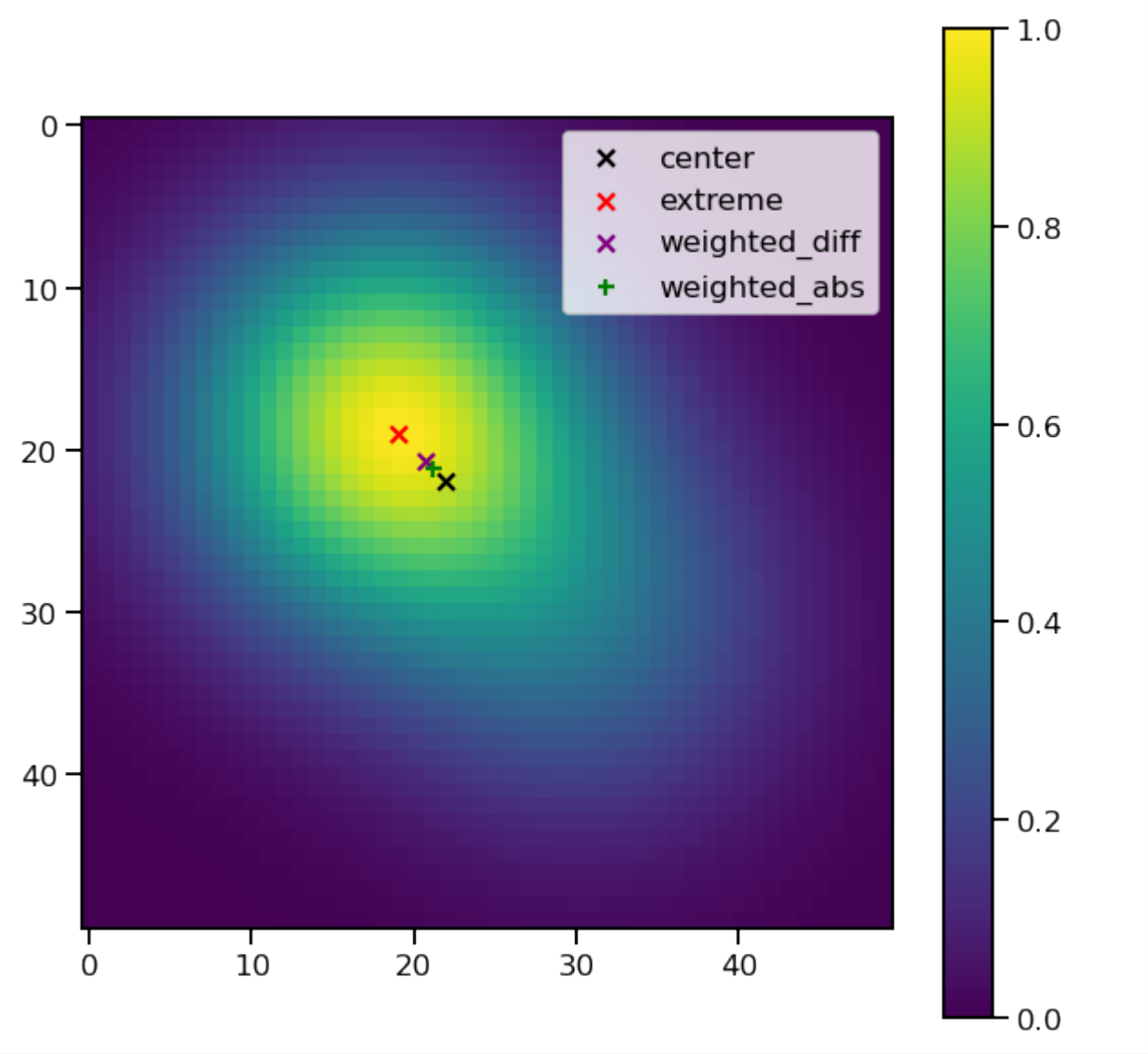
[11]:
```
plt.figure(figsize=(10, 10))
plt.imshow(assym[0])
plt.scatter(
    features_center["hdim_2"],
    features_center["hdim_1"],
    color="black",
    marker="x",
    label="center",
)
plt.scatter(
    features_extreme["hdim_2"],
    features_extreme["hdim_1"],
    color="red",
    marker="x",
    label="extreme",
```

(continues on next page)

(continued from previous page)

```
)
plt.scatter(
    features_diff["hdim_2"],
    features_diff["hdim_1"],
    color="purple",
    marker="x",
    label="weighted_diff",
)
plt.scatter(
    features_abs["hdim_2"],
    features_abs["hdim_1"],
    color="green",
    marker="+",
    label="weighted_abs",
)
plt.colorbar()
plt.legend()
```

[11]:
```
<matplotlib.legend.Legend at 0x7f49d16ecb50>
```



As you can see this parameter specifies how the postion of the feature is defined. These are the descriptions given in the code:

- extreme: get position as max/min position inside the identified region
- center : get position as geometrical centre of identified region
- weighted_diff: get position as centre of identified region, weighted by difference from the threshold
- weighted_abs: get position as centre of identified region, weighted by absolute values if the field

4.3 Sigma Parameter for Smoothing of Noisy Data

Before the features are searched a gausian filter is applied to the data in order to smooth it. So let's import the filter used by tobac for a demonstration:

[12]:
```
from scipy.ndimage import gaussian_filter
```

This filter works performing a convolution of a (in our case 2-dimensional) gaussian function

$$h(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

with our data and with this parameter we set the value of σ.

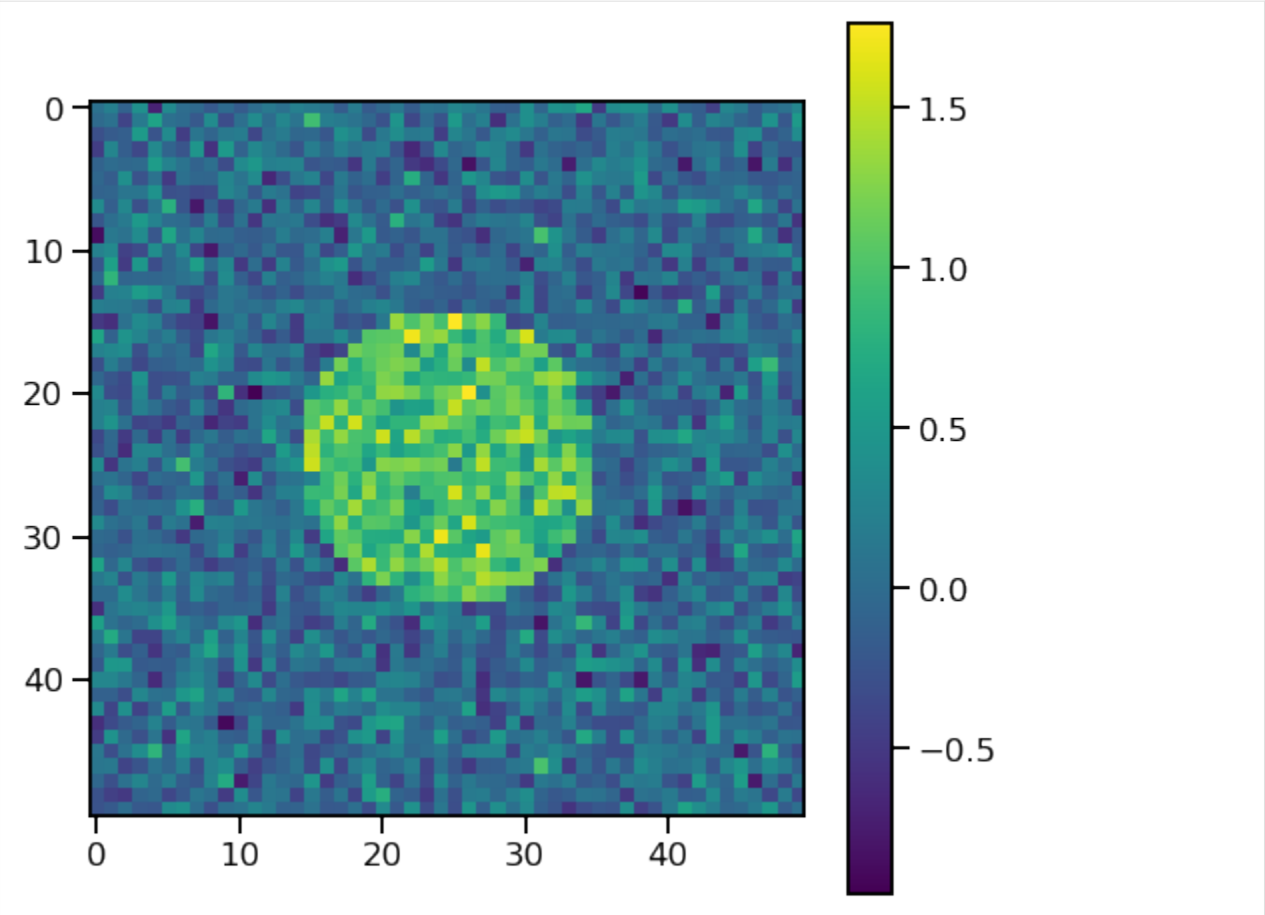
The effect of this filter can best be demonstrated on very sharp edges in the input. Therefore we create an array from a boolean mask of another 2d-Gaussian, which has only values of 0 or 1:

[13]:
```
x = np.linspace(-2, 2)
y = np.linspace(-2, 2)
xx, yy = np.meshgrid(x, y)

exp = np.exp(-(xx**2 + yy**2))

gaussian_data = np.expand_dims(exp, axis=0)
```

and we add some random noise to it:

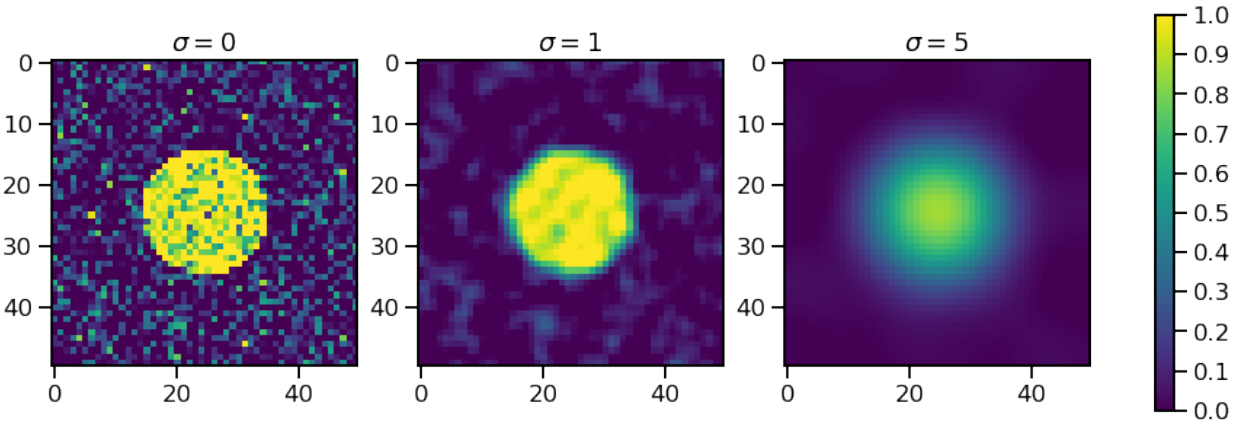
[14]:
```
np.random.seed(54321)

noise = 0.3 * np.random.randn(*gaussian_data.shape)
data_sharp = np.array(gaussian_data > 0.5, dtype="float32")

data_sharp += noise

plt.figure(figsize=(8, 8))
plt.imshow(data_sharp[0])
plt.colorbar()
```

[14]:
```
<matplotlib.colorbar.Colorbar at 0x7f49d15c81c0>
```



If we apply this filter to the data with increasing sigmas, increasingly smoothed data will be the result:

[15]:
```
non_smooth_data = gaussian_filter(data_sharp, sigma=0)
smooth_data = gaussian_filter(data_sharp, sigma=1)
smoother_data = gaussian_filter(data_sharp, sigma=5)

fig, axes = plt.subplots(ncols=3, figsize=(16, 10))

im0 = axes[0].imshow(non_smooth_data[0], vmin=0, vmax=1)
axes[0].set_title(r"$\sigma = 0$")

im1 = axes[1].imshow(smooth_data[0], vmin=0, vmax=1)
axes[1].set_title(r"$\sigma = 1$")

im2 = axes[2].imshow(smoother_data[0], vmin=0, vmax=1)
axes[2].set_title(r"$\sigma = 5$")

cbar = fig.colorbar(im1, ax=axes.tolist(), shrink=0.5)
cbar.set_ticks(np.linspace(0, 1, 11))
```



This is what happens in the background, when the `feature_detection_multithreshold()` function is called. The default value of `sigma_threshold` is 0.5. The next step is trying to detect features of the dataset with these `sigma_threshold` values. We first need an xarray DataArray again:

[16]:
```
date = np.datetime64("2022-04-01T00:00")
input_data = xr.DataArray(
    data=data_sharp, coords={"time": np.expand_dims(date, axis=0), "y": y, "x": x}
)
```

Now we set a threshold and detect the features:

[17]:
```
%%capture

threshold = 0.9
features_sharp = tobac.themes.tobac_v1.feature_detection_multithreshold(
    input_data, dxy, threshold, sigma_threshold=0
)
features_smooth = tobac.themes.tobac_v1.feature_detection_multithreshold(
    input_data, dxy, threshold, sigma_threshold=1
)
features_smoother = tobac.themes.tobac_v1.feature_detection_multithreshold(
    input_data, dxy, threshold, sigma_threshold=5
)
```

Attempting to plot the results

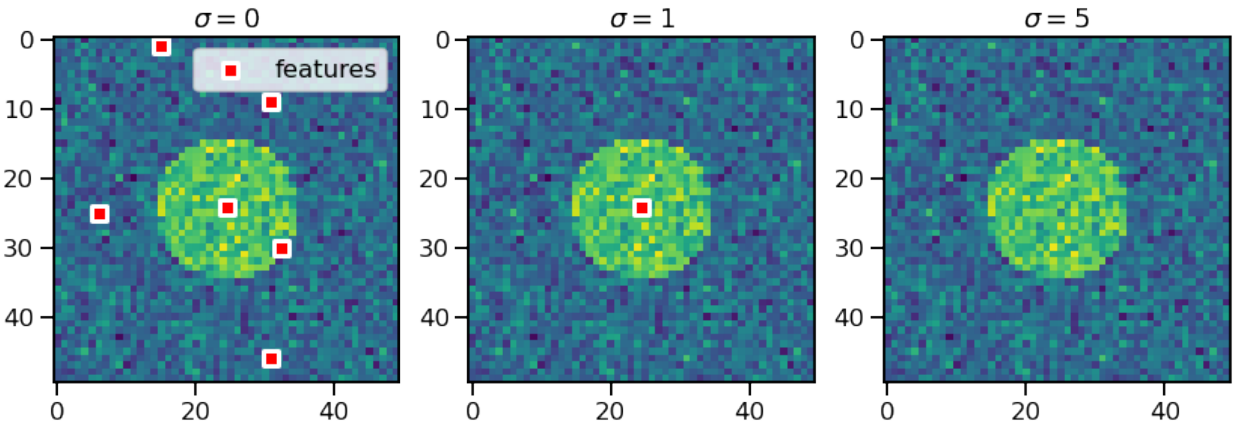
[18]:
```
fig, axes = plt.subplots(ncols=3, figsize=(14, 10))
plot_kws = dict(
    color="red",
    marker="s",
    s=100,
    edgecolors="w",
    linewidth=3,
)

im0 = axes[0].imshow(input_data[0])
axes[0].set_title(r"$\sigma = 0$")
axes[0].scatter(
    features_sharp["hdim_2"], features_sharp["hdim_1"], label="features", **plot_kws
```

(continues on next page)

(continued from previous page)

```
)
axes[0].legend()

im0 = axes[1].imshow(input_data[0])
axes[1].set_title(r"$\sigma = 1$")
axes[1].scatter(features_smooth["hdim_2"], features_smooth["hdim_1"], **plot_kws)

im0 = axes[2].imshow(input_data[0])
axes[2].set_title(r"$\sigma = 5$")
try:
    axes[2].scatter(
        features_smoother["hdim_2"], features_smoother["hdim_1"], **plot_kws
    )
except:
    print("WARNING: No Feature Detected!")
```

```
WARNING: No Feature Detected!
```



Noise may cause some false detections (left panel) that are significantly reduced when a suitable smoothing parameter is chosen (middle panel).

4.4 Band-Pass Filter for Input Fields via Parameter `wavelength_filtering`

This parameter can be understood best, when looking at real instead of snythethic data. An example of usage is given *here*

CHAPTER

FIVE

METHODS AND PARAMETERS FOR FEATURE DETECTION: PART 2

In this notebook, we will contninue to look in detail at tobac's feature detection and examine the remaining parameters.

We will treat:

- *Object Erosion Parameter*
- *Minimum Object Pair Distance*

[1]:
```
import tobac
import matplotlib.pyplot as plt
import numpy as np
import xarray as xr

import seaborn as sns

sns.set_context("talk")

%matplotlib inline
```

5.1 Object Erosion Parameter `n_erosion_threshold`

To understand this parameter we have to look at one varibale of the feature-Datasets we did not mention so far: *num*

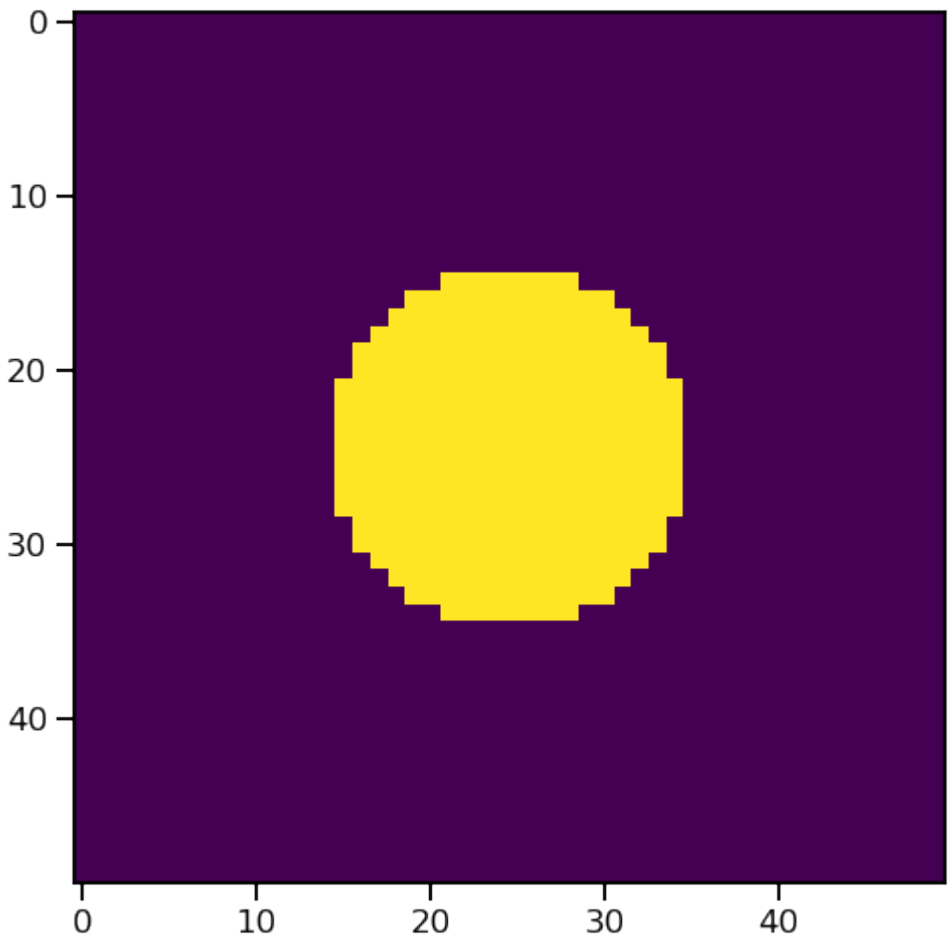
The value of *num* for a specific feature tells us the number of datapoints exceeding the threshold. *n_erosion_threshold* reduces this number by eroding the mask of the feature on its boundary. Supose we are looking at the gaussian data again and we set a treshold of 0.5. The resulting mask of our feature will look like this:

[2]:
```
x = np.linspace(-2, 2)
y = np.linspace(-2, 2)
xx, yy = np.meshgrid(x, y)

exp = np.exp(-(xx**2 + yy**2))

gaussian_data = np.expand_dims(exp, axis=0)
threshold = 0.5

mask = gaussian_data > threshold
mask = mask[0]

plt.figure(figsize=(8, 8))
```

(continues on next page)

(continued from previous page)

```
plt.imshow(mask)
plt.show()
```



The erosion algorithm used by tobac is imported from *skimage.morphology*:

[3]:
```
from skimage.morphology import binary_erosion
```

Applying this algorithm requires a quadratic matrix. The size of this matrix is provided by the *n_erosion_threshold* parameter. For a quick demonstration we can create the matrix by hand and apply the erosion for different values:

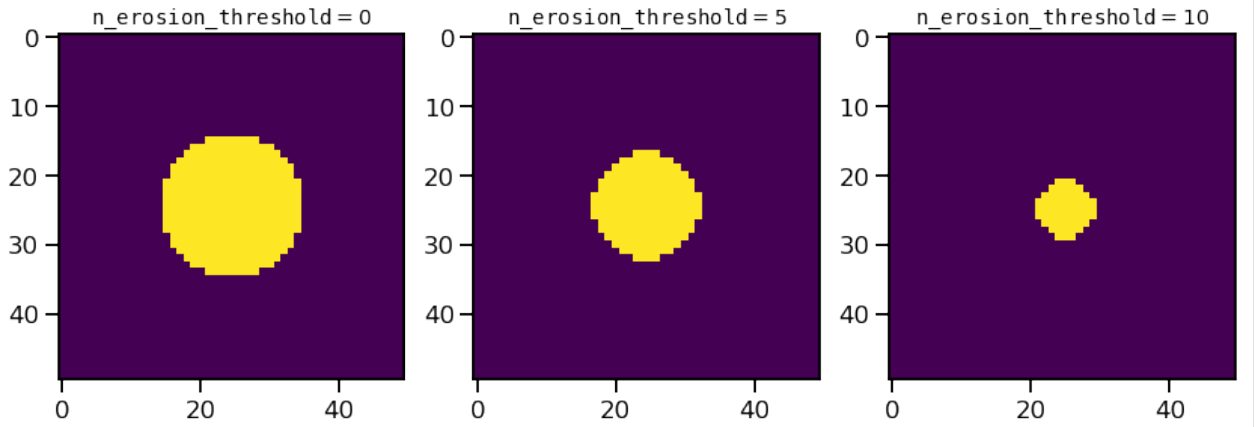
[4]:
```
fig, axes = plt.subplots(ncols=3, figsize=(14, 10))

im0 = axes[0].imshow(mask)
axes[0].set_title(r"$\mathtt{n\_erosion\_threshold} = 0$", fontsize=14)

n_erosion_threshold = 5
selem = np.ones((n_erosion_threshold, n_erosion_threshold))
mask_er = binary_erosion(mask, selem).astype(np.int64)

im1 = axes[1].imshow(mask_er)
axes[1].set_title(r"$\mathtt{n\_erosion\_threshold} = 5$", fontsize=14)
```

(continues on next page)

(continued from previous page)

```
n_erosion_threshold = 10
selem = np.ones((n_erosion_threshold, n_erosion_threshold))
mask_er_more = binary_erosion(mask, selem).astype(np.int64)

im2 = axes[2].imshow(mask_er_more)
axes[2].set_title(r"$\mathtt{n\_erosion\_threshold} = 10$", fontsize=14)
```

[4]:
```
Text(0.5, 1.0, '$\\mathtt{n\\_erosion\\_threshold} = 10$')
```



This means by using increasing values of *n_erosion_threshold* for a feature detection we will get lower values of *num*, which will match the number of **True**-values in the masks above:

[5]:
```
%%capture

date = np.datetime64("2022-04-01T00:00")
input_data = xr.DataArray(
    data=gaussian_data, coords={"time": np.expand_dims(date, axis=0), "y": y, "x": x}
)
dxy = input_data["x"][1] - input_data["x"][0]

features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    input_data, dxy, threshold, n_erosion_threshold=0
)
features_eroded = tobac.themes.tobac_v1.feature_detection_multithreshold(
    input_data, dxy, threshold, n_erosion_threshold=5
)
features_eroded_more = tobac.themes.tobac_v1.feature_detection_multithreshold(
    input_data, dxy, threshold, n_erosion_threshold=10
)
```

[6]:
```
features["num"].data[0]
```

[6]:
```
332
```

[7]:
```
mask.sum()
```

[7]:
```
332
```

[8]:
```
features_eroded["num"].data[0]
```
[8]:
```
188
```

[9]:
```
mask_er.sum()
```
[9]:
```
188
```

[10]:
```
features_eroded_more["num"].data[0]
```
[10]:
```
57
```

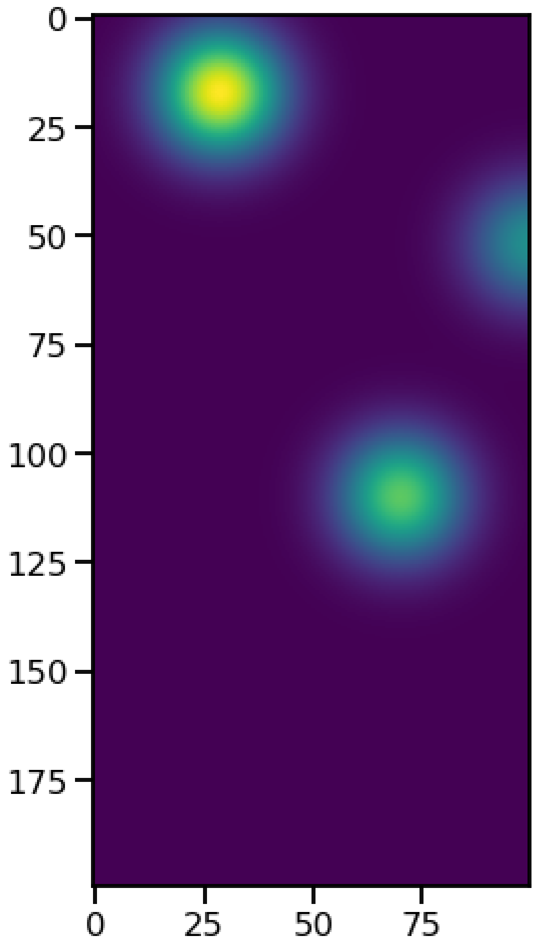
[11]:
```
mask_er_more.sum()
```
[11]:
```
57
```

This can be used to simplify the geometry of complex features.

5.2 Minimum Object Size Parameter `n_min_threshold`

With *n_min_threshold* parameter we can exclude smaller features by setting a minimum of datapoints that have to exceed the threshold for one feature. If we again detect the three blobs and check their *num* value at frame 50, we can see that one of them contains fewer pixels.

[12]:
```
data = tobac.testing.make_sample_data_2D_3blobs()

plt.figure(figsize=(8, 8))
plt.imshow(data[50])
plt.show()
```



[13]:
```
%%capture
threshold = 9
dxy, dt = tobac.utils.get_spacings(data)
features = tobac.themes.tobac_v1.feature_detection_multithreshold(data, dxy, threshold)
```

[14]:
```
features.where(features["frame"] == 50)["num"]
```

[14]:
```
<xarray.DataArray 'num' (index: 102)>
array([ nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan, 501.,  30., 325.,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan])
Coordinates:
  * index    (index) int64 0 1 2 3 4 5 6 7 8 9 ... 93 94 95 96 97 98 99 100 101
```

Obviously, the feature with only 30 datapoints is the rightmost feature that has almost left the imaging area. If we now

use an *n_min-threshold* above or equal to 30, we will not detect this small feature:

[15]:
```
%%capture
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, threshold, n_min_threshold=30
)
```

[16]:
```
features.where(features["frame"] == 50)["num"]
```

[16]:
```
<xarray.DataArray 'num' (index: 100)>
array([ nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan, 501., 325.,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
        nan])
Coordinates:
  * index    (index) int64 0 1 2 3 4 5 6 7 8 9 ... 90 91 92 93 94 95 96 97 98 99
```

Noisy data can be cleaned using this method by ignoring smaller features of no significance or, as here, features that leave the detection range.

5.3 Minimum Object Pair Distance `min_distance`

Another way of getting rid of this specific feature is the *min_distance* parameter. It sets a minimal distance of our features. Lets detect the features as before:

[17]:
```
%%capture

data = tobac.testing.make_sample_data_2D_3blobs()

threshold = 9
dxy, dt = tobac.utils.get_spacings(data)
features = tobac.themes.tobac_v1.feature_detection_multithreshold(data, dxy, threshold)
```

A quick look at frame 50 tells us the found features and their indices:

[18]:
```
n = 50
mask = features["frame"] == n
features_frame = features.where(mask).dropna("index").to_dataframe()
features_frame.index.values
```

[18]:
```
array([60, 61, 62])
```

Notice that *to_dataframe()* was used to convert the Dataset to a pandas dataframe, which is required to use the *calculate_distance* function of the analysis module. The distances bewteen our features are:

[19]:
```
tobac.analysis.analysis.calculate_distance(
    features_frame.loc[60], features_frame.loc[61], method_distance="xy"
)
```

[19]:
```
77307.67873610597
```

[20]:
```
tobac.analysis.analysis.calculate_distance(
    features_frame.loc[62], features_frame.loc[61], method_distance="xy"
)
```

[20]:
```
64289.48419281164
```

[21]:
```
tobac.analysis.analysis.calculate_distance(
    features_frame.loc[62], features_frame.loc[60], method_distance="xy"
)
```

[21]:
```
101607.57596570993
```

With this knowledge we can set reasonable values for *min_distance* to exclude the small feature:

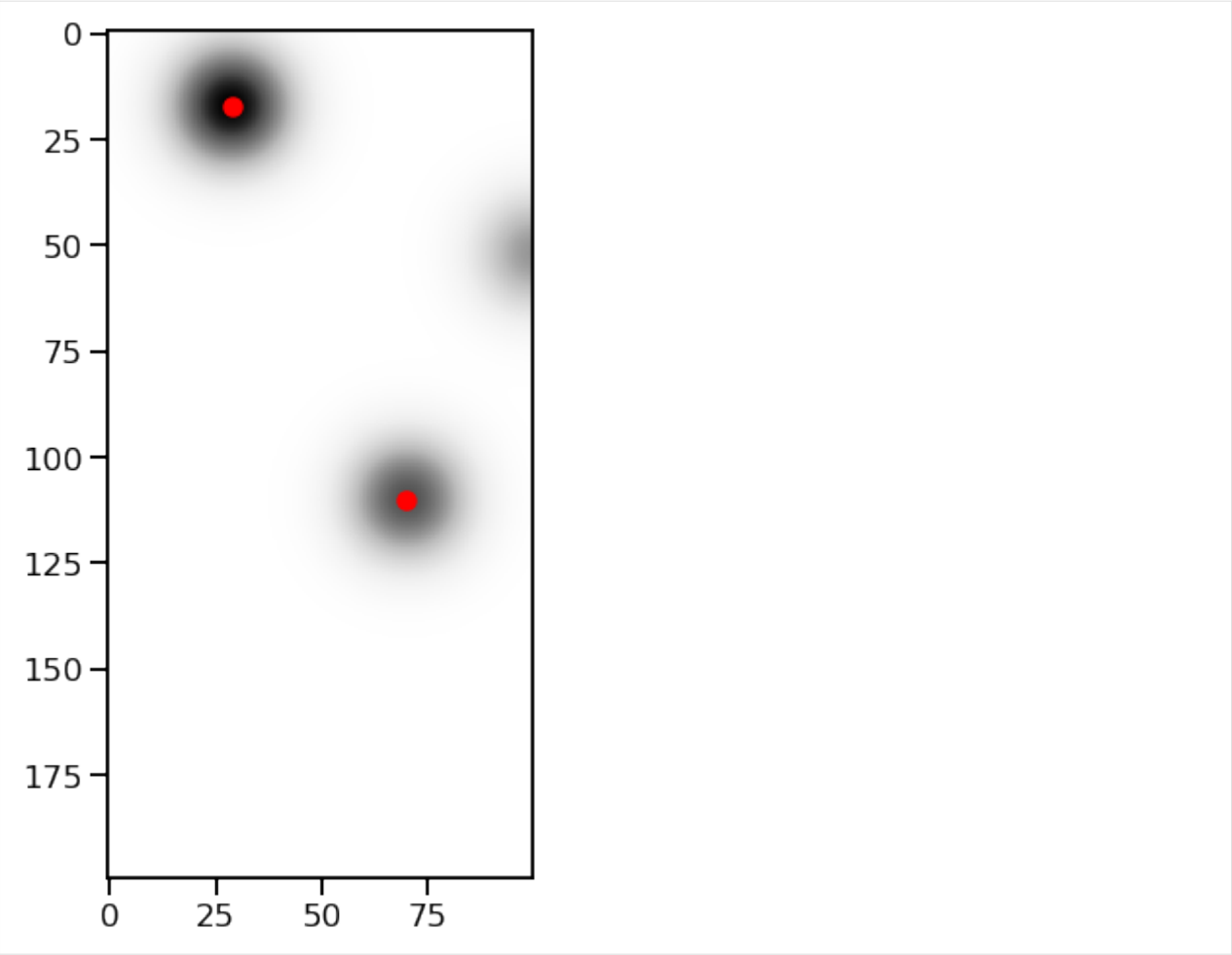
[22]:
```
min_distance = 70000
```

and perform the feature detection as usual:

[23]:
```
%%capture

data = tobac.testing.make_sample_data_2D_3blobs()

thresholds = [10]
dxy, dt = tobac.utils.get_spacings(data)
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, thresholds, min_distance=min_distance
)

n = 50
mask = features["frame"] == n
```

Plotting the result shows us that we now exclude the expected feature.

[24]:
```
fig, ax1 = plt.subplots(ncols=1, figsize=(8, 8))
ax1.imshow(data.data[50], cmap="Greys")

im1 = ax1.scatter(
    features.where(mask)["hdim_2"], features.where(mask)["hdim_1"], color="red"
)
```



If the features have the same threshold, tobac keeps the feature with the larger area. Otherwise the feature with the higher treshold is kept.

CHAPTER

SIX

METHODS AND PARAMETERS FOR SEGMENTATION

This notebook explores the segmentation function of `tobac` and its parameters:

- *Required Inputs*
- *Different Thresholds*
- *Choosing Method and Target*
- *Setting a maximum Distance*
- *Handling 3d-Data*

We start with the usual imports:

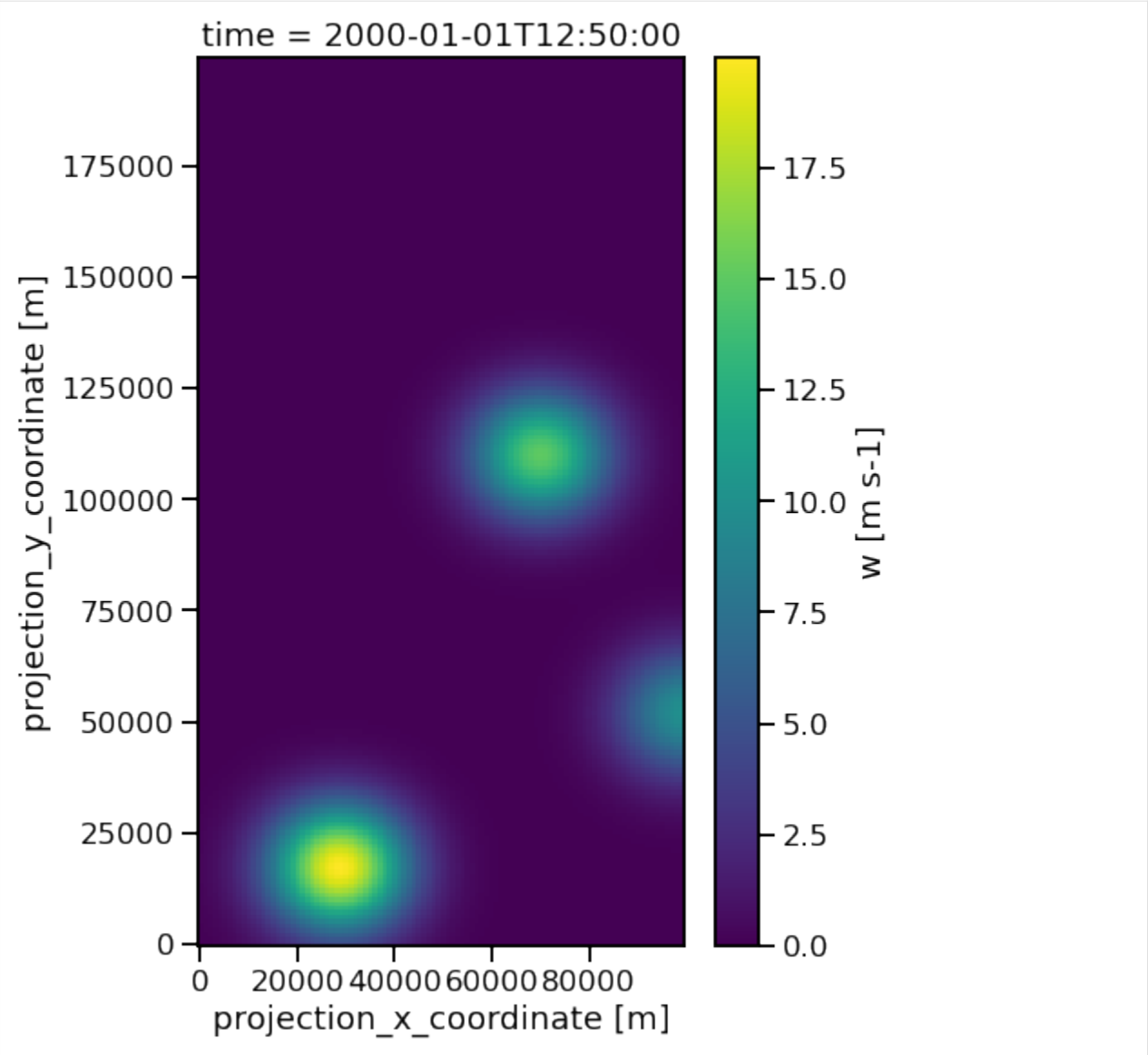
[1]:
```
import tobac
import matplotlib.pyplot as plt
import numpy as np
import xarray as xr

%matplotlib inline

import seaborn as sns

sns.set_context("talk")
```

6.1 Required Inputs

To perform a segmentation we need a dataset with already detected features. Therefore, we take advantage of the `testing.make_sample_data_2D_3blobs_inv()`-utility and detect features with different thresholds:

[2]:
```
data = tobac.testing.make_sample_data_2D_3blobs_inv()
dxy, dt = tobac.utils.get_spacings(data)

plt.figure(figsize=(6, 9))
data.isel(time=50).plot(x="x", y="y")
```

[2]:
```
<matplotlib.collections.QuadMesh at 0x7f928889f510>
```



[3]:
```
%%capture
thresholds = [9, 14, 17]
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, thresholds, position_threshold="weighted_abs"
)
```

The resulting dataset can now be used as argument for the `segmentation()`-function. The other required inputs are the original dataset, the spacing and a threshold.

[4]:
```
mask, features_mask = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=9
)
```

```
<xarray.DataArray 'w' (time: 100, x: 100, y: 200)>
[2000000 values with dtype=float64]
Coordinates:
```

(continues on next page)

(continued from previous page)

```
  * time       (time) datetime64[ns] 2000-01-01T12:00:00 ... 2000-01-01T13:39:00
  * x          (x) float64 0.0 1e+03 2e+03 3e+03 ... 9.7e+04 9.8e+04 9.9e+04
  * y          (y) float64 0.0 1e+03 2e+03 3e+03 ... 1.97e+05 1.98e+05 1.99e+05
    latitude   (x, y) float64 ...
    longitude  (x, y) float64 ...
Attributes:
    units:    m s-1
```

The created segments are provided as mask, which is the first returned object of the function. The second output is the features-dataset again, but with the additional *ncells*-variable, which gives us the number of datapoints belonging to the feature:

[5]:
```
features_mask["ncells"][1]
```

[5]:
```
<xarray.DataArray 'ncells' ()>
array(67.)
Coordinates:
    index    int64 1
```

Notice that this number can be deviate from the *num*-value, because watershedding works differently from just detecting the values exceeeding the threshold. For example, for the second feature *ncells* contains one additional datapoint compared to the original feature detection:

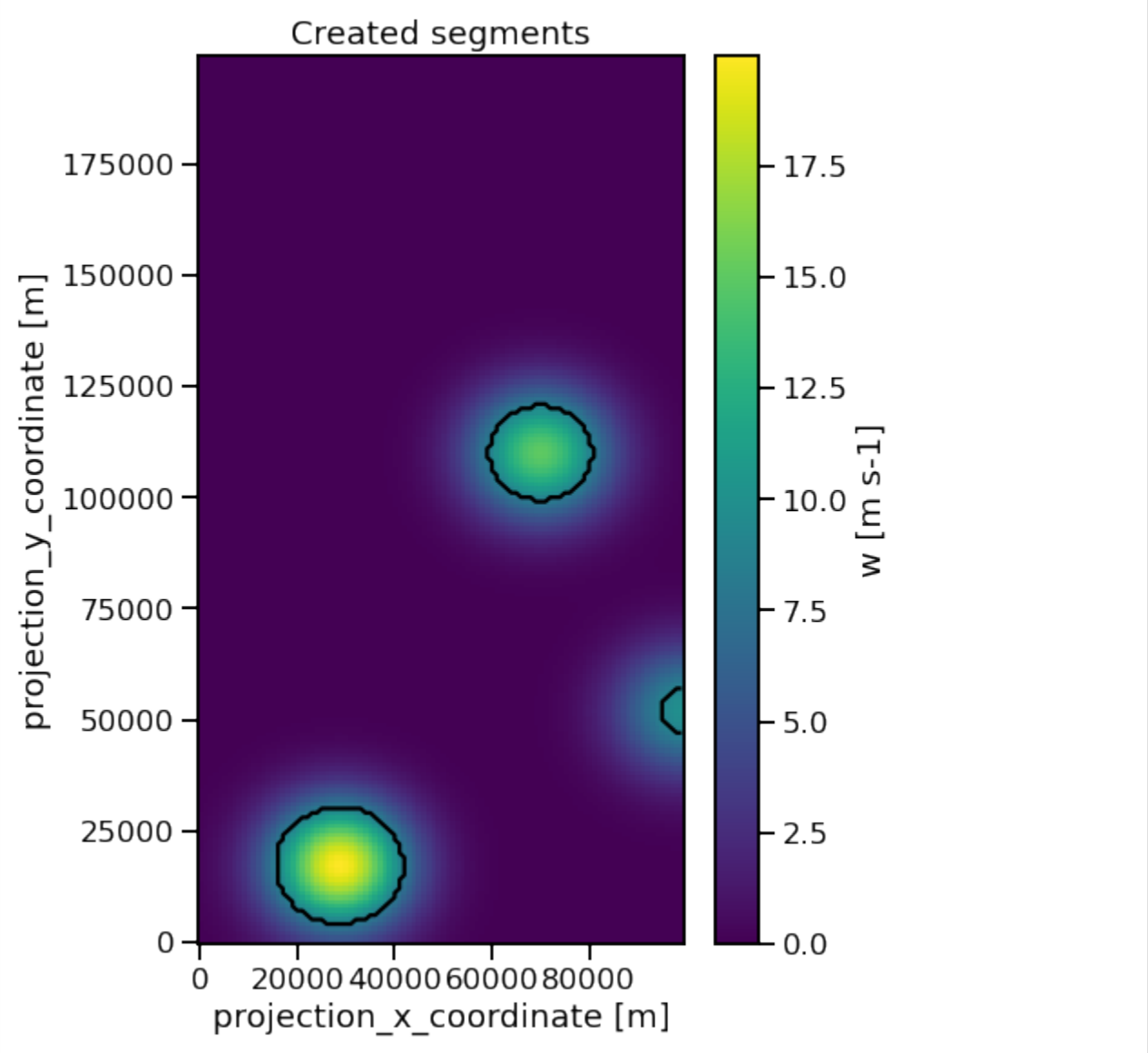
[6]:
```
features_mask["num"][1]
```

[6]:
```
<xarray.DataArray 'num' ()>
array(66)
Coordinates:
    index    int64 1
```

The created segments can be visualized with a contour plot of tha mask:

[7]:
```
plt.figure(figsize=(6, 9))
data.isel(time=50).plot(x="x", y="y")
mask.isel(time=50).plot.contour(levels=[0.5], x="x", y="y", colors="k")
plt.title("Created segments")
```

[7]:
```
Text(0.5, 1.0, 'Created segments')
```



6.2 Different Thresholds

It is important to highlight that (in contrast to the feature detection), segmentation is only possible with **single threshold values**. Because of that, we have to call the function multiple times with different `threshold` values to explore the influence of this argument:

[8]:
```
%%capture
mask_1, features_mask_1 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=9
)
mask_2, features_mask_2 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=14
)
mask_3, features_mask_3 = tobac.themes.tobac_v1.segmentation(
```

(continues on next page)

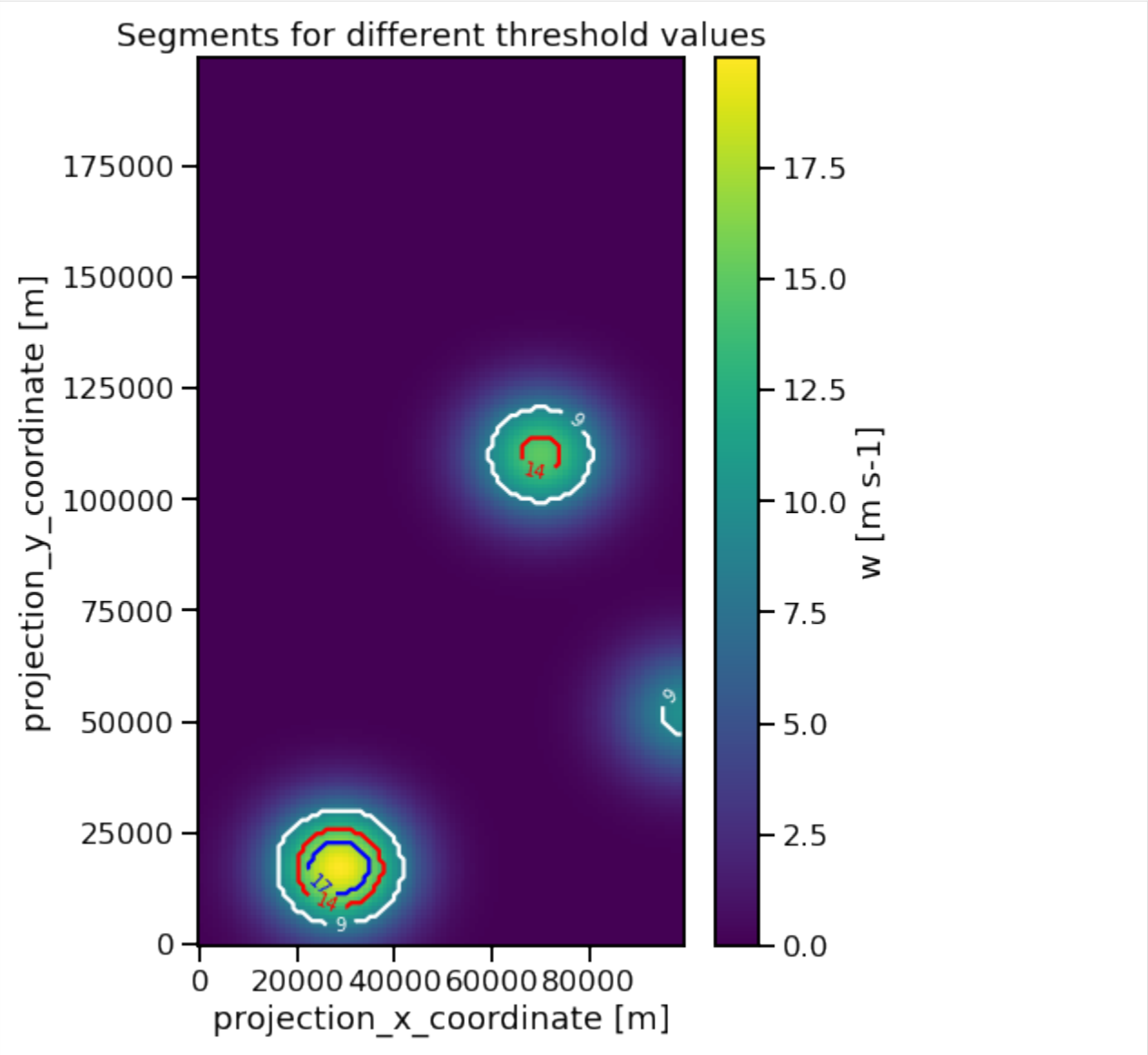
(continued from previous page)

```
    features, data, dxy, threshold=17
)
```

To visualize the segments we can use contour-plots of the masks:

[9]:
```
thresholds = [9, 14, 17]
masks = [mask_1, mask_2, mask_3]
colors = ["w", "r", "b"]

fig, ax = plt.subplots(ncols=1, figsize=(6, 9))
data.isel(time=50).plot(ax=ax, x="x", y="y")

for n, mask, color in zip(thresholds, masks, colors):

    contour = mask.isel(time=50).plot.contour(levels=[n], x="x", y="y", colors=color)
    ax.clabel(contour, inline=True, fontsize=10)

ax.set_title("Segments for different threshold values")
```

[9]:
```
Text(0.5, 1.0, 'Segments for different threshold values')
```



Obviously, a lower threshold value prodcuces a larger segment and if a feature does not exceed the value at all, no segment is associated.

6.3 Choosing Method and Target

The segmentation uses certain techniques to associate areas or volumes to each identified feature. Watershedding is the default and the only implemented option at the moment, but in future realeases the method will be selected by the `method`-keyword:

[10]:
```
%%capture
mask_1, features_mask_1 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=9, method="watershed"
)
```

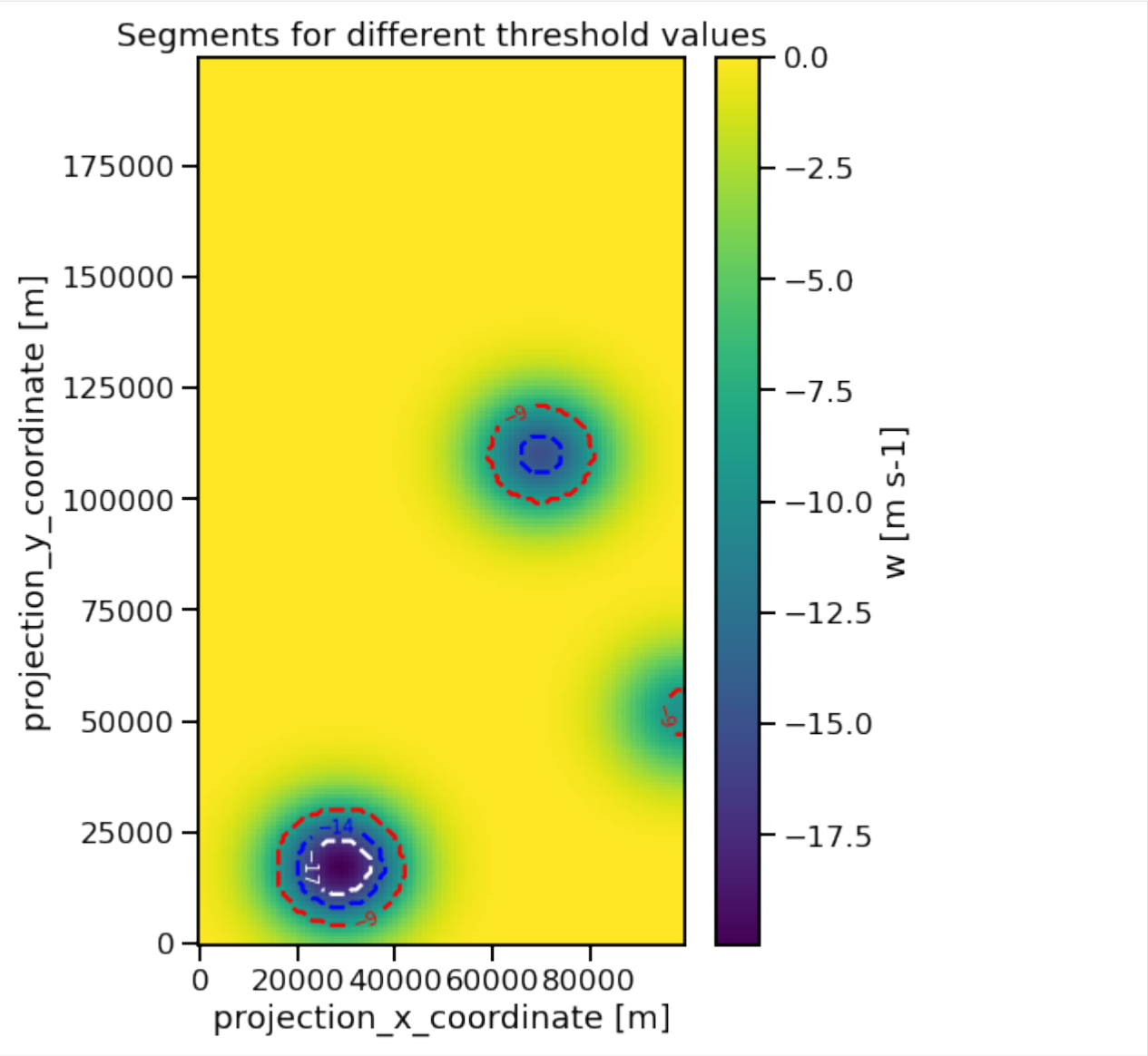
Analogous to the feature detection, it is also possible to apply the segmentation to minima by changing the *target*

keyword:

[11]:
```
%%capture

data = -tobac.testing.make_sample_data_2D_3blobs_inv()
dxy, dt = tobac.utils.get_spacings(data)
thresholds = [-9, -14, -17]
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, thresholds, target="minimum"
)

mask_1, features_mask_1 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=-9, target="minimum"
)
mask_2, features_mask_2 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=-14, target="minimum"
)
mask_3, features_mask_3 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=-17, target="minimum"
)
```

[12]:
```
masks = [mask_1, mask_2, mask_3]
colors = ["r", "b", "w"]
thresholds = [-9, -14, -17]

fig, ax = plt.subplots(ncols=1, figsize=(6, 9))
data.isel(time=50).plot(ax=ax, x="x", y="y")

for n, mask, color in zip(thresholds, masks, colors):

    contour = (
        (n * mask).isel(time=50).plot.contour(levels=[n], colors=color, x="x", y="y")
    )
    ax.clabel(contour, inline=True, fontsize=10)

ax.set_title("Segments for different threshold values")
```

[12]:
```
Text(0.5, 1.0, 'Segments for different threshold values')
```



6.4 Setting a maximum Distance

Another way of determining the size of our segments is the `max_distance`-parameter. It defines a maximum distance the segment can have from the coordinates of feature (in meters). This enables us, for example, to ensure that the segments of different features do not touch each other when we use a very low threshold value:

[13]:
```
%%capture

data = tobac.testing.make_sample_data_2D_3blobs_inv()
dxy, dt = tobac.utils.get_spacings(data)
thresh = 0.1

features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, threshold=3
```

(continues on next page)

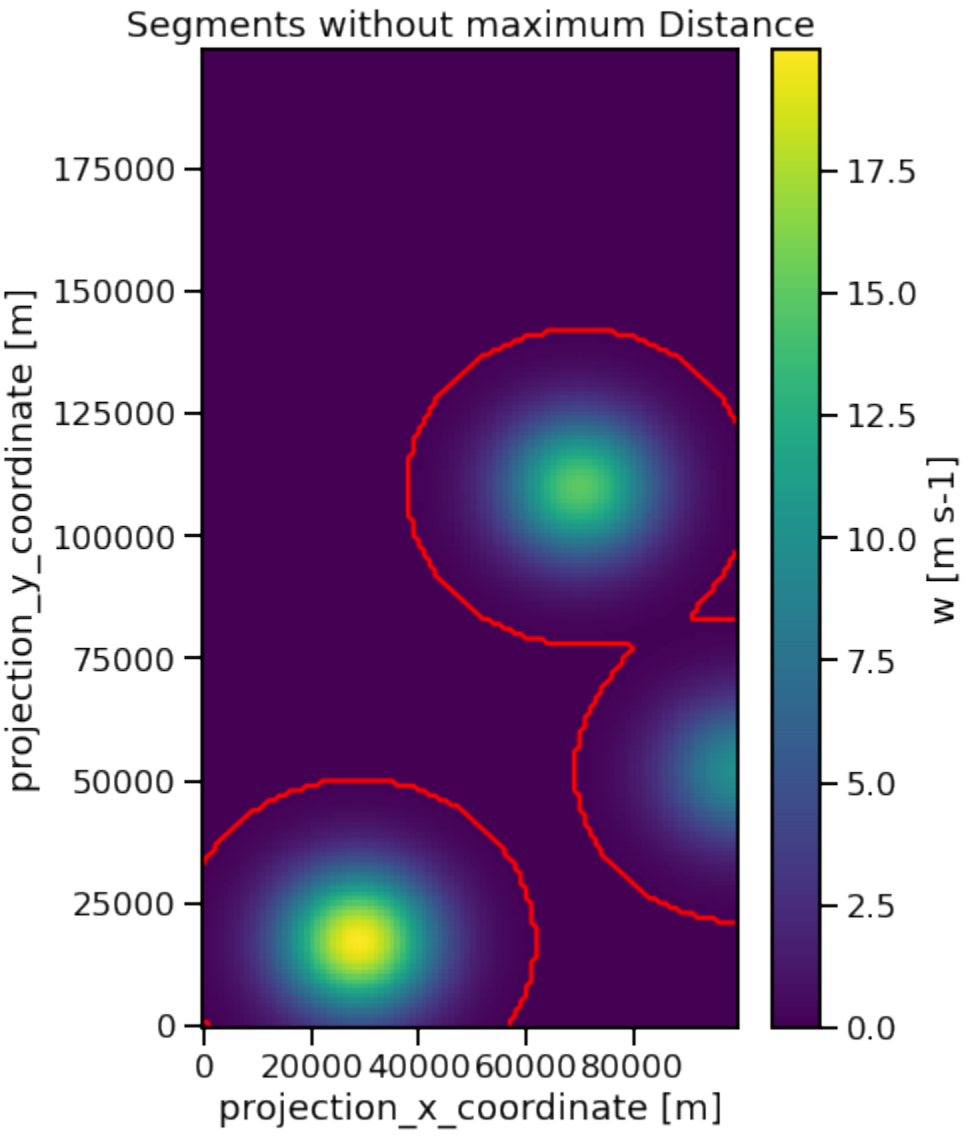
(continued from previous page)

```
)
mask_0, features_0 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=thresh
)
```

As you can see the threshold value was set to a value of 0.1. The result is that the segments of the two upper features will touch:

[14]:
```
fig, ax = plt.subplots(figsize=(6, 9))
data.isel(time=50).plot(ax=ax, x="x", y="y")
mask_0.isel(time=50).plot.contour(levels=[0.5], ax=ax, colors="r", x="x", y="y")
ax.set_title("Segments without maximum Distance")
```

[14]:
```
Text(0.5, 1.0, 'Segments without maximum Distance')
```

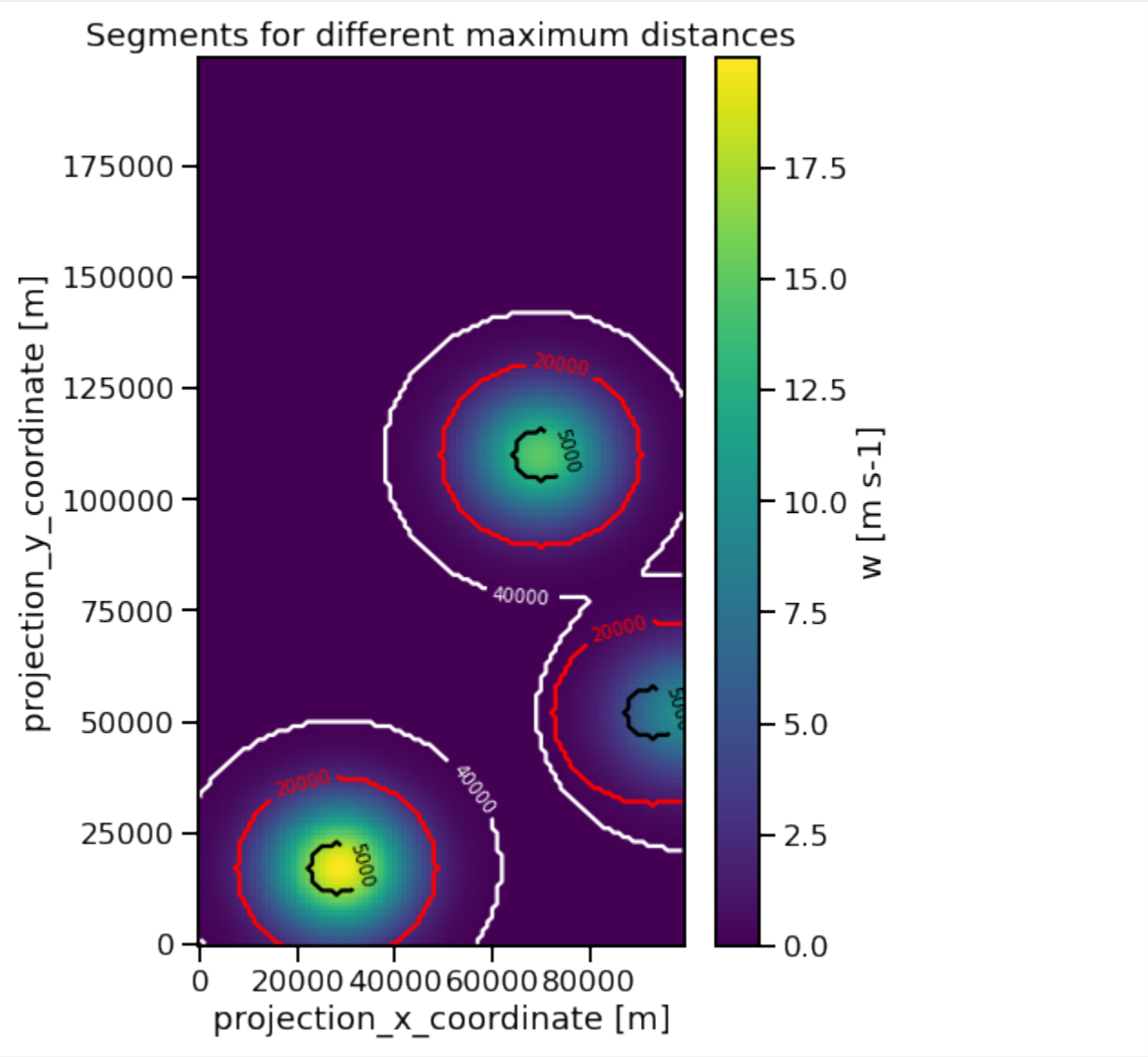


We can prevent this from happening by using the `max_distance` parameter to specify a maximum distance the border of the segment can have from the feature in meter:

[15]:
```
%%capture

mask_1, features_mask_1 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=thresh, max_distance=40000
)
mask_2, features_mask_2 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=thresh, max_distance=20000
)
mask_3, features_mask_3 = tobac.themes.tobac_v1.segmentation(
    features, data, dxy, threshold=thresh, max_distance=5000
)
```

[16]:
```
masks = [mask_1, mask_2, mask_3]
colors = ["w", "r", "k"]
distances = [4e4, 2e4, 5e3]

fig, ax = plt.subplots(ncols=1, figsize=(6, 9))
data.isel(time=50).plot(ax=ax, x="x", y="y")

for n, mask, color in zip(distances, masks, colors):

    contour = (
        (n * mask).isel(time=50).plot.contour(levels=[n], colors=color, x="x", y="y")
    )
    ax.clabel(contour, inline=True, fontsize=10)

ax.set_title("Segments for different maximum distances")
```

[16]:
```
Text(0.5, 1.0, 'Segments for different maximum distances')
```



6.5 Handling 3d-Data

The remaining parameters `level` and `vertical_coord` are useful only for the segemtation of 3-dimensional inputs and will be covered in the notebook for 3d-data

CHAPTER

SEVEN

METHODS AND PARAMETERS FOR LINKING

Linking assigns the detected features and segments in each timestep to trajectories, to enable an analysis of their time evolution. We will cover the topics:

- *Understanding the Linking Output*
- *Defining a Search Radius*
- *Working with the Subnetwork*
- *Timescale of the Tracks*
- *Gappy Tracks*
- *Other Parameters*

We start by loading the usual libraries:

[1]:
```
import tobac
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline

import seaborn as sns

sns.set_context("talk")
```

7.1 Understanding the Linking Output

Since it has a time dimension, the sample data from the testing utilities is also suitable for a demonstration of this step. The chosen method creates 2-dimensional sample data with up to 3 moving blobs at the same. So, our expectation is to obtain up to 3 tracks.

At first, loading in the data and performing the usual feature detection is required:

[2]:
```
%%capture

data = tobac.testing.make_sample_data_2D_3blobs_inv()
dxy, dt = tobac.utils.get_spacings(data)
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, threshold=1, position_threshold="weighted_abs"
)
```

Now the linking via trackpy can be performed. Notice that the temporal spacing is also a required input this time. Additionally, it is necessary to provide either a maximum speed *v_max* or a maximum search range *d_max*. Here we use a maximum speed of 100 m/s:

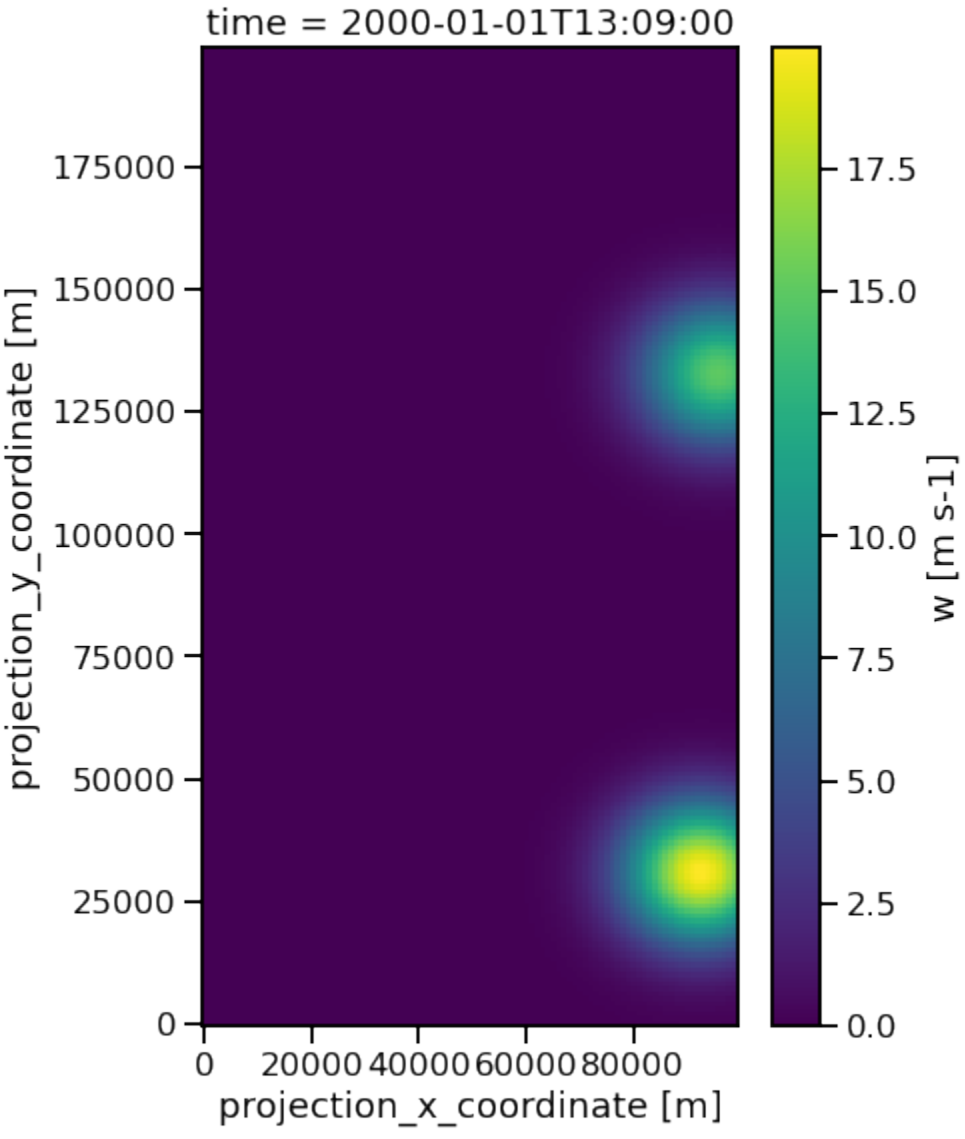
[3]:
```
track = tobac.themes.tobac_v1.linking_trackpy(features, data, dt=dt, dxy=dxy, v_max=100)
```

```
Frame 69: 2 trajectories present.
```

The output tells us, that in the final frame 69 two trajecteries where still present. If we checkout this frame via `xarray.plot` method, we can see that this corresponds to the two present features there, which are assigned to different trajectories.

[4]:
```
plt.figure(figsize=(6, 9))
data.isel(time=69).plot(x="x", y="y")
```

[4]:
```
<matplotlib.collections.QuadMesh at 0x7f064377b6d0>
```



The track dataset contains two new variables, *cell* and *time_cell*. The first assigns the features to one of the found trajectories by an integer and the second specifies how long the feature has been present already in the data.

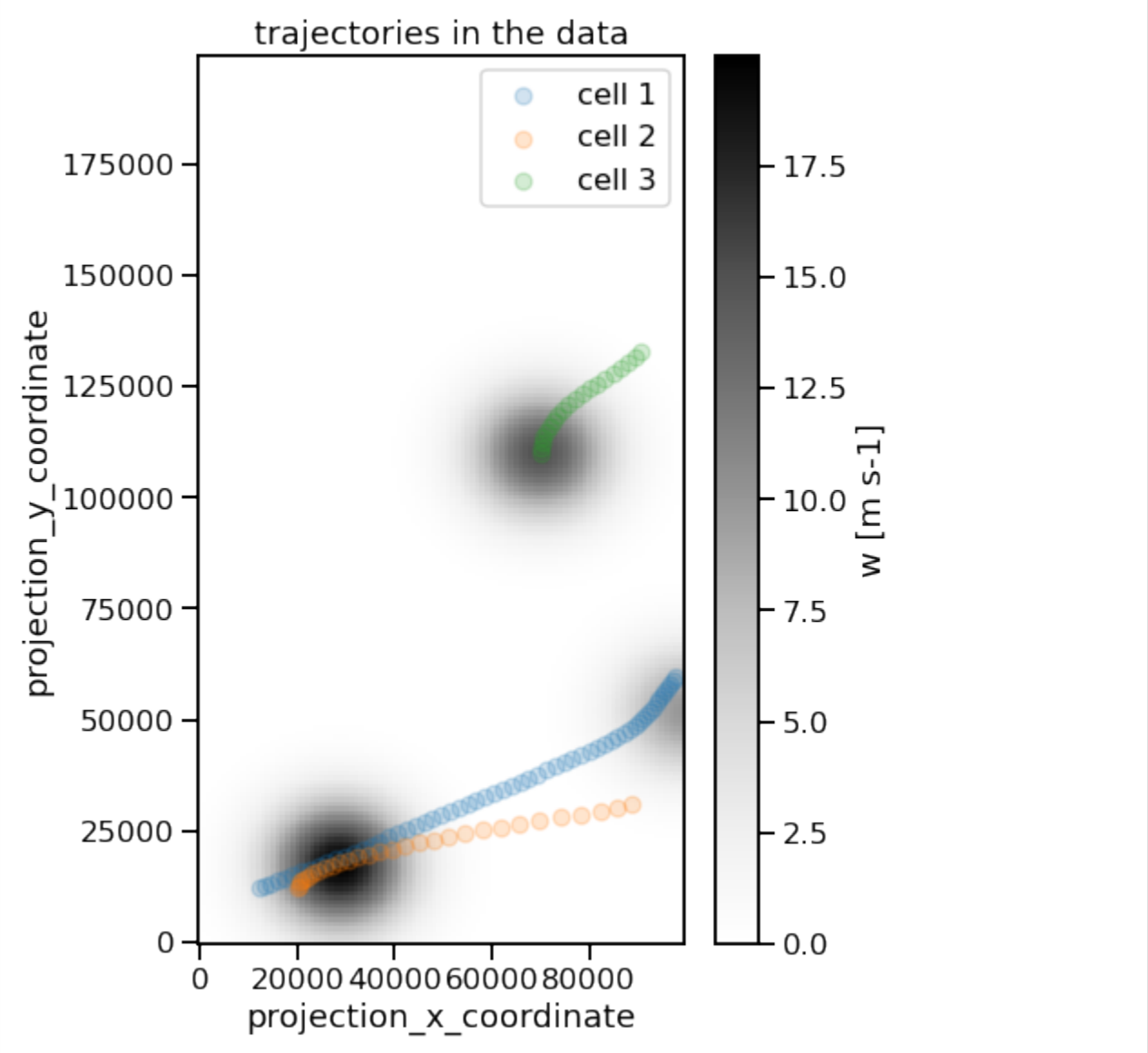
[5]:
```
track[["cell", "time_cell"]]
```

[5]:
```
<xarray.Dataset>
Dimensions:    (index: 110)
Coordinates:
  * index      (index) int64 0 1 2 3 4 5 6 7 ... 102 103 104 105 106 107 108 109
Data variables:
    cell       (index) float64 1.0 1.0 1.0 1.0 1.0 1.0 ... 3.0 2.0 3.0 2.0 3.0
    time_cell  (index) timedelta64[ns] 00:00:00 00:01:00 ... 00:29:00 00:19:00
```

Since we know that this dataset contains 3 features, we can visualize the found tracks with masks created from the *cell* variable:

[6]:
```
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 9))

data.isel(time=50).plot(ax=ax, x="x", y="y", cmap="Greys")

for i, cell_track in track.groupby("cell"):

    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax,
        marker="o",
        alpha=0.2,
        label="cell {0}".format(int(i)),
    )


ax.set_title("trajectories in the data")
plt.legend()
```

[6]:
```
<matplotlib.legend.Legend at 0x7f064d306590>
```



The cells have been specified with **different velocities** that also change in time. `tobac` retrieves the following values for our test data:

[7]:
```
vel = tobac.analysis.calculate_velocity(track)
```

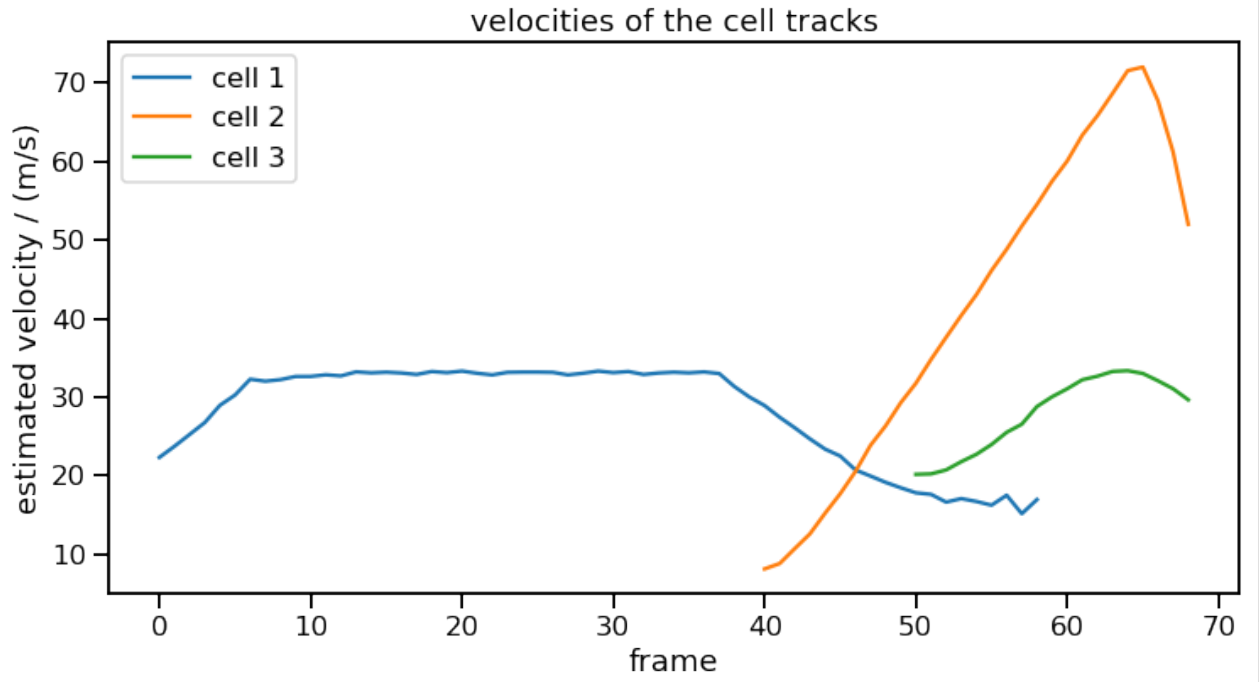
[8]:
```
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(12, 6))

for i, vc in vel.groupby("cell"):

    # get velocity as xarray Dataset
    v = vc.to_xarray()["v"]
    v = v.assign_coords({"frame": vc.frame})

    v.plot(x="frame", ax=ax, label="cell {0}".format(int(i)))

    ax.set_ylabel("estimated velocity / (m/s)")
```

(continues on next page)

(continued from previous page)

```
ax.set_title("velocities of the cell tracks")
plt.legend()
```

[8]:
```
<matplotlib.legend.Legend at 0x7f0642a4b490>
```



Interpretation:

- "cell 1" is actually specified with constant velocity. The initial increase and the final decrease come from edge effects, i.e. that the blob corresponding to "cell 1" has parts that that go beyond the border.
- "cell 2" and "cell 3" are specified with linearily increasing x-component of the velocity. The initial speed-up is due to the square-root behavior of the velocity magnitude. The final decrease however come again from edge effects.

The starting point of the cell index is of course arbitrary. If we want to change it from 1 to 0 we can do this with the `cell_number_start` parameter:

[9]:
```
track = tobac.themes.tobac_v1.linking_trackpy(
    features, data, dt=dt, dxy=dxy, v_max=100, cell_number_start=0
)

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 9))

data.isel(time=50).plot(ax=ax, x="x", y="y", cmap="Greys")

for i, cell_track in track.groupby("cell"):

    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax,
```

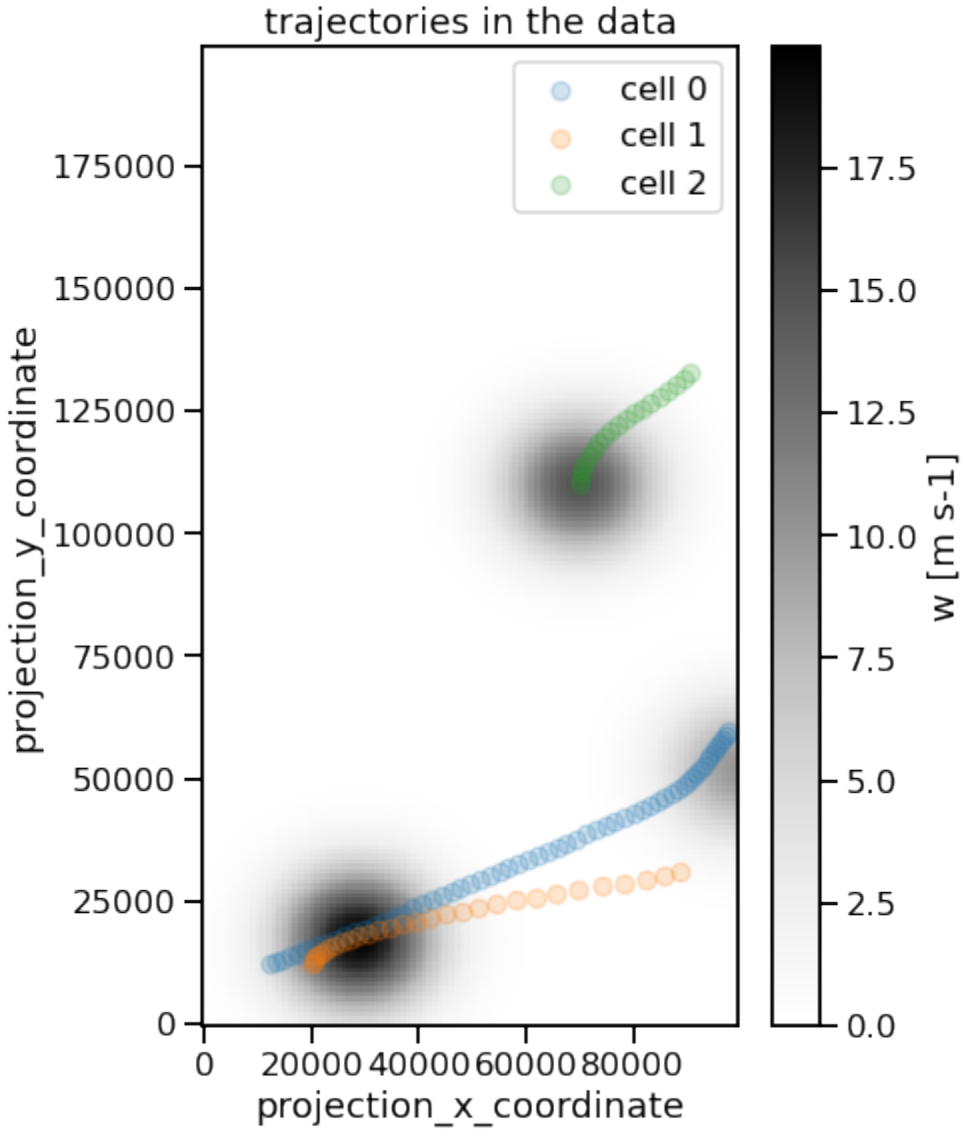
(continues on next page)

(continued from previous page)

```
        marker="o",
        alpha=0.2,
        label="cell {0}".format(int(i)),
    )

ax.set_title("trajectories in the data")
plt.legend()
```

```
Frame 69: 2 trajectories present.
```

[9]:
```
<matplotlib.legend.Legend at 0x7f0642918cd0>
```



The linking in `tobac` is performed with methods of the trackpy-library. Currently, there are two methods implemented, 'random' and 'predict'. These can be selected with the `method` keyword. The default is 'random'.

[10]:
```
track_p = tobac.themes.tobac_v1.linking_trackpy(
    features, data, dt=dt, dxy=dxy, v_max=100, method_linking="predict"
```

(continues on next page)

(continued from previous page)

```
)
```

```
Frame 69: 2 trajectories present.
```

7.2 Defining a Search Radius

If you paid attention to the input of the `linking_trackpy()` function you noticed that we used the parameter `v_max` there. The reason for this is that it would take a lot of computation time to check the whole data of a frame for the next position of a feature from the previous frame with the Crocker-Grier linking algorithm. Therefore the search is restricted to a circle around a position predicted by different methods implemented in *trackpy*.

The speed we defined before specifies such a radius timestep via

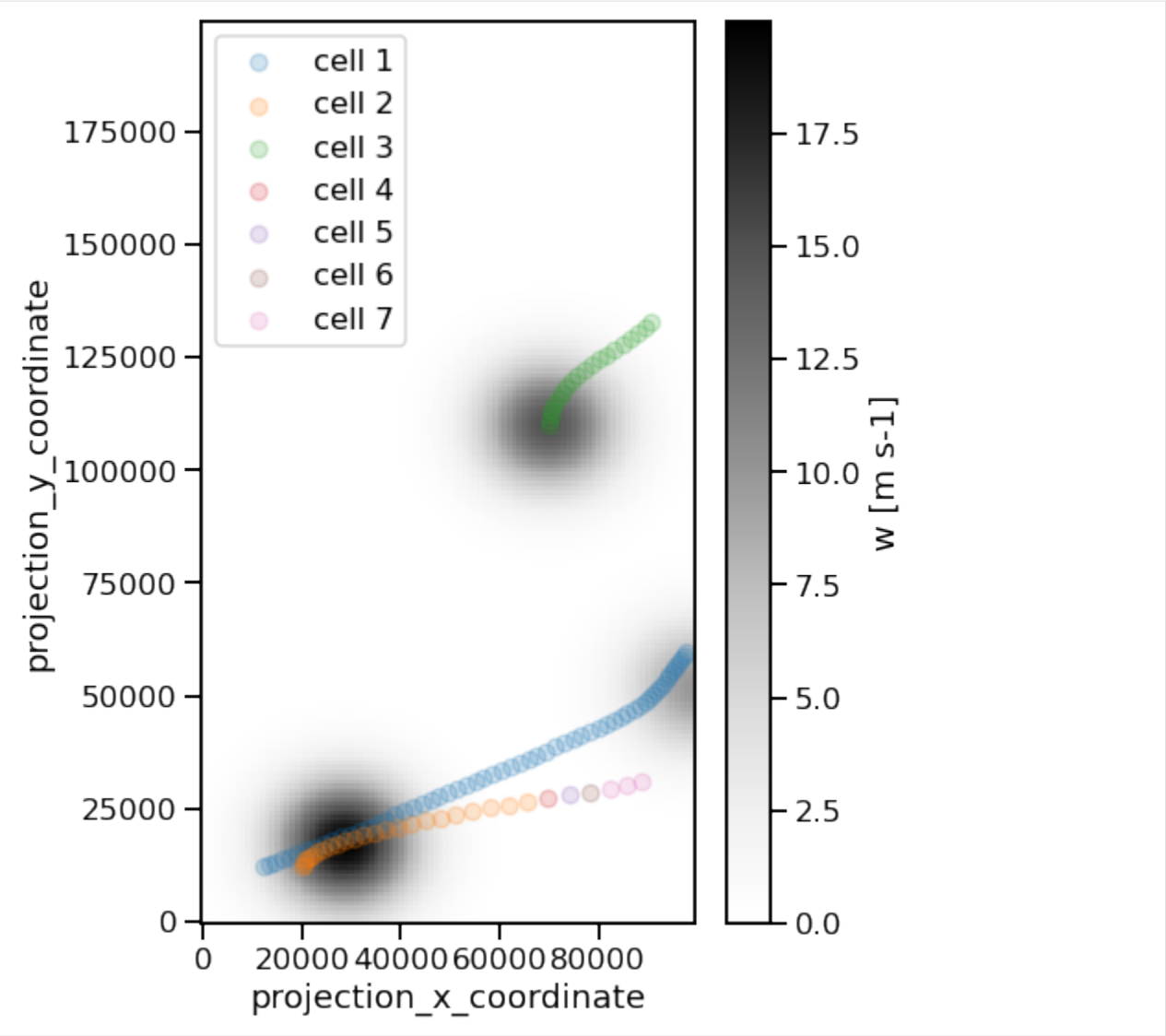
$$r_{max} = v_{max} \Delta t.$$

By reducing the **maximum speed** from 100 m/s to 70 m/s we will find more cell tracks, because a feature that is not assigned to an existing track will be used as a starting point for a new one:

[11]:
```
track = tobac.themes.tobac_v1.linking_trackpy(features, data, dt=dt, dxy=dxy, v_max=70)

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 9))

data.isel(time=50).plot(ax=ax, x="x", y="y", cmap="Greys")

for i, cell_track in track.groupby("cell"):
    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax,
        marker="o",
        alpha=0.2,
        label="cell {0}".format(int(i)),
    )

plt.legend()
```

```
Frame 69: 2 trajectories present.
```

[11]:
```
<matplotlib.legend.Legend at 0x7f0642957790>
```



Above you see that previously orange track is *split into five parts* because the considered cell moves so fast.

The same effect can be achieved by directly reducing the **maximum search range** with the `d_max` parameter to

$$d_{max} = 4200m$$

because

$$v_{max}\Delta t = 4200m$$

for $v_{max} = 70$ m/s in our case.

[12]:
```
track = tobac.themes.tobac_v1.linking_trackpy(
    features, data, dt=dt, dxy=dxy, d_max=4200
)

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 9))
```

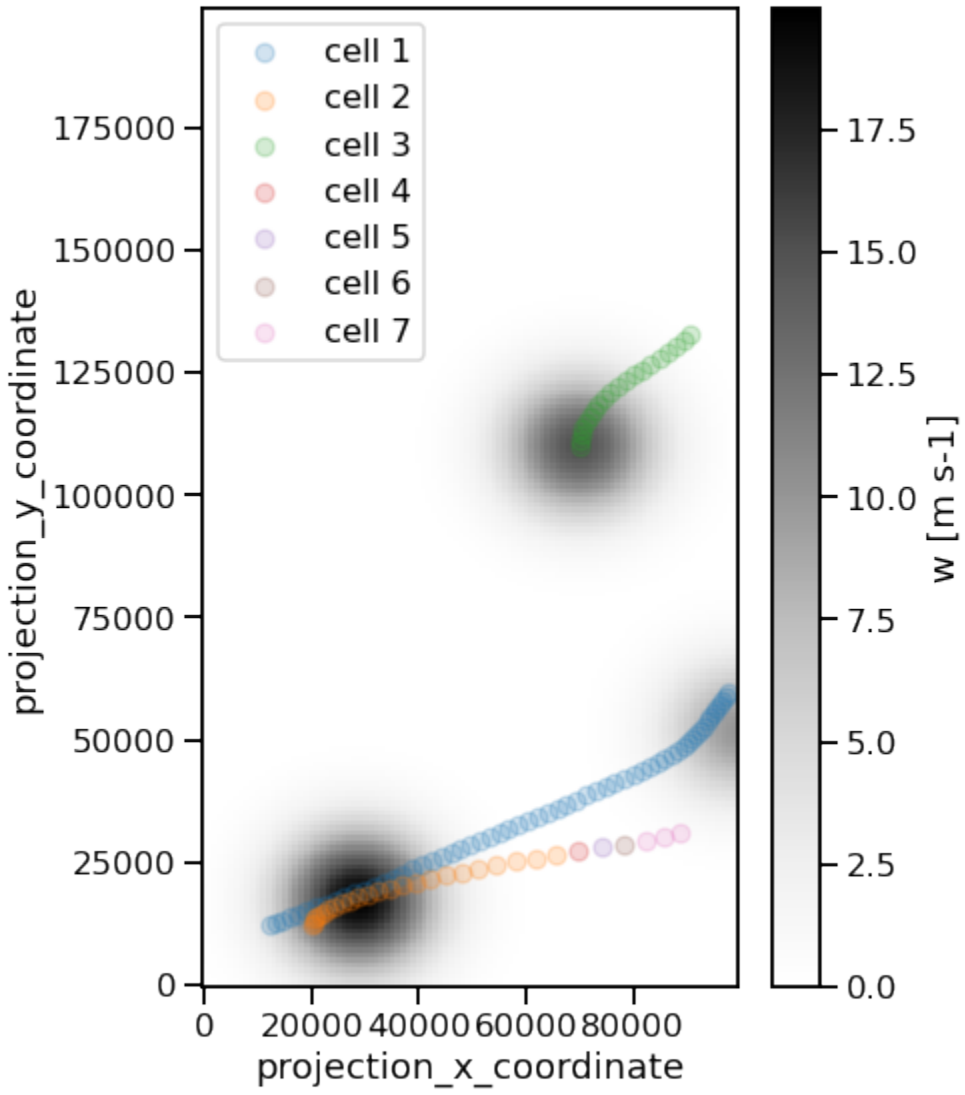
(continues on next page)

(continued from previous page)

```
data.isel(time=50).plot(ax=ax, x="x", y="y", cmap="Greys")

for i, cell_track in track.groupby("cell"):
    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax,
        marker="o",
        alpha=0.2,
        label="cell {0}".format(int(i)),
    )

plt.legend()
```

```
Frame 69: 2 trajectories present.
```

[12]:
```
<matplotlib.legend.Legend at 0x7f0641dbe510>
```



v_max and *d_max* should not be used together, because they both specify the same qunatity, namely the search range, but it is neccery to set at least one of these parameters.

7.3 Working with the Subnetwork

If we increase *v_max* or *d_max* it can happen that more than one feature of the previous timestep is in the search range. The selection of the best matching feature from the set of possible features (which is called the subnetwork, for a more in depth explanation have a look here) is the most time consuming part of the linking process. For this reason, it is possible to limit the number of features in the search range via the `subnetwork_size` parameter. The tracking will result in a **trackpy.SubnetOversizeException** if more features than specified there are in the search range. We can simulate this situation by setting a really high value for `v_max` and 1 for the `subnetwork_size`:

[13]:
```
from trackpy import SubnetOversizeException

try:

    track = tobac.themes.tobac_v1.linking_trackpy(
        features, data, dt=dt, dxy=dxy, v_max=1000, subnetwork_size=1
    )
    print("tracking successful")

except SubnetOversizeException as e:

    print("Tracking not possible because the {}".format(e))
```

```
Frame 57: 3 trajectories present.
Tracking not possible because the Subnetwork contains 2 points
```

The default value for this parameter is 30 and can be accessed via:

[14]:
```
import trackpy as tp

tp.linking.Linker.MAX_SUB_NET_SIZE
```

[14]:
```
1
```

It is important to highlight that if the `linking_trackpy()`-function is called with a `subnetwork_size` different from 30, this number will be the new default. This is the reason, why the value is 1 at this point.

7.3.1 Adaptive Search

A way of dealing with **SubnetOversizeException**s is adaptive search. An extensive description can be found here.

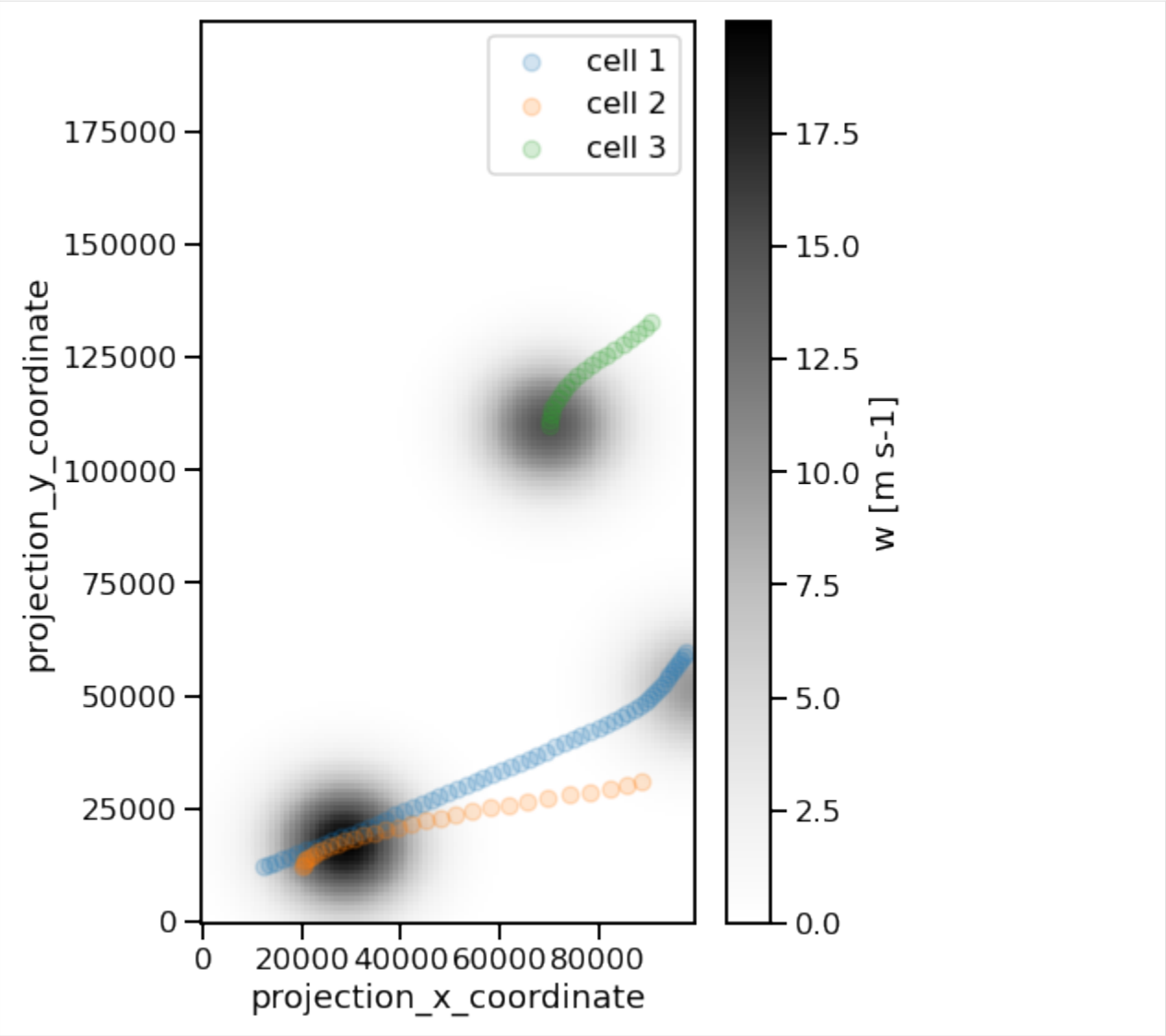
If the subnetwork is too large, the search range is reduced iteratively by multiplying it with the `adaptive_step`. The `adaptive_stop`-parameter is the lower limit of the search range. If it is reached and the subnetwork is still too large, a **SubnetOversizeException** is thrown. Notice that as soon as adaptive search is used, a different limit of the subnetwork size applies, which needs to be specified by hand at the moment:

[15]:
```
tp.linking.Linker.MAX_SUB_NET_SIZE_ADAPTIVE = 1
```

Now the tracking can be performed and will no longer result in an exception:

[16]:
```
try:

    track = tobac.themes.tobac_v1.linking_trackpy(
        features, data, dt=dt, dxy=dxy, v_max=1000, adaptive_stop=10, adaptive_step=0.9
    )
    print("tracking successful!")

    fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 9))

    data.isel(time=50).plot(ax=ax, x="x", y="y", cmap="Greys")

    for i, cell_track in track.groupby("cell"):
        cell_track.plot.scatter(
            x="projection_x_coordinate",
            y="projection_y_coordinate",
            ax=ax,
            marker="o",
            alpha=0.2,
            label="cell {0}".format(int(i)),
        )

    plt.legend()

except SubnetOversizeException as e:

    print("Tracking not possible because the {}".format(e))
```

```
Frame 69: 2 trajectories present.
tracking successful!
```



If `adaptive_stop` is specified, `adaptive_step` needs to be specified as well.

7.4 Timescale of the Tracks

If we want to limit our analysis to long living features of the dataset it is possible to exclude tracks which only cover few timesteps. This is done by setting a lower bound for those via the `stups` parameter. In our test data, only the first cell exceeds 50 time steps:

[17]:
```
track = tobac.themes.tobac_v1.linking_trackpy(
    features, data, dt=dt, dxy=dxy, v_max=100, stubs=50
)

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 9))

data.isel(time=50).plot(ax=ax, x="x", y="y", cmap="Greys")
```

(continues on next page)

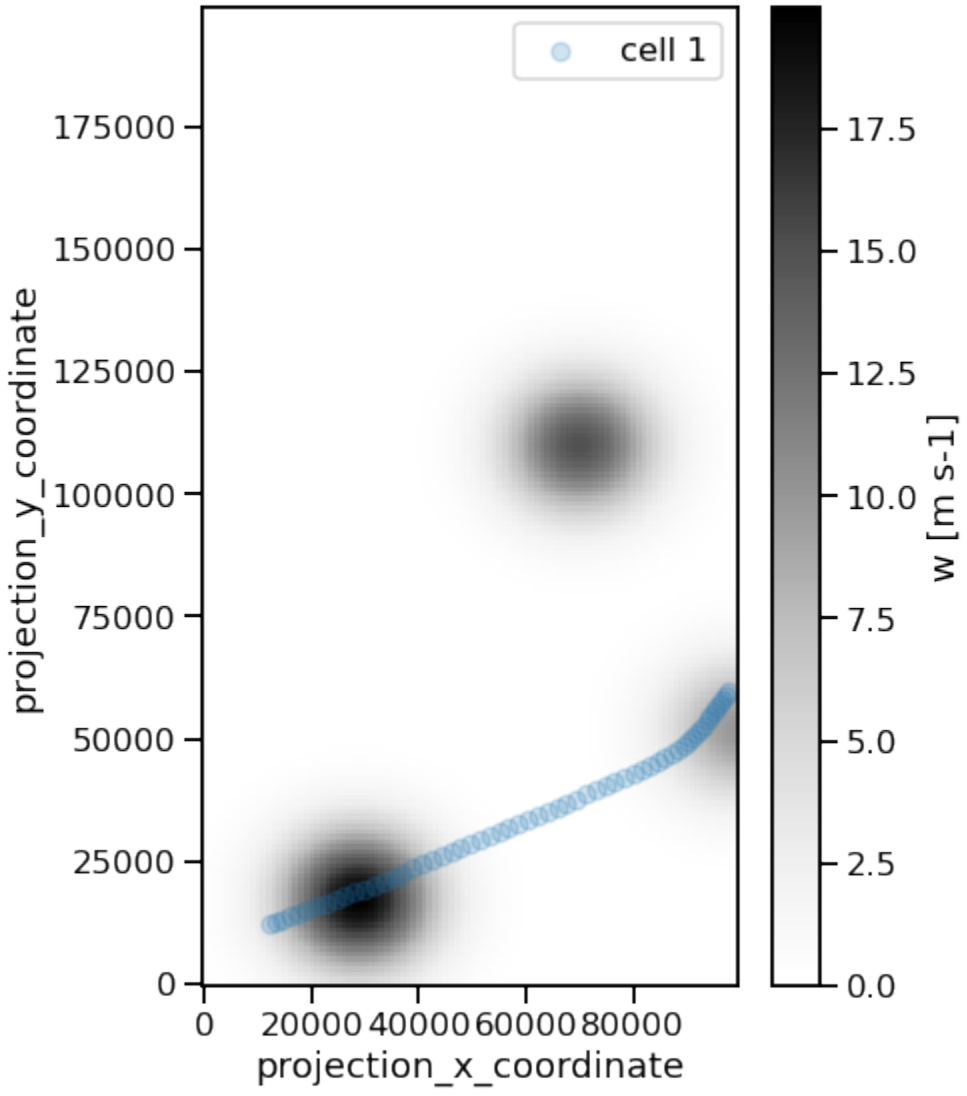
(continued from previous page)

```
for i, cell_track in track.groupby("cell"):
    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax,
        marker="o",
        alpha=0.2,
        label="cell {0}".format(int(i)),
    )

plt.legend()
```

```
Frame 69: 2 trajectories present.
```

[17]:

```
<matplotlib.legend.Legend at 0x7f0641d4d990>
```



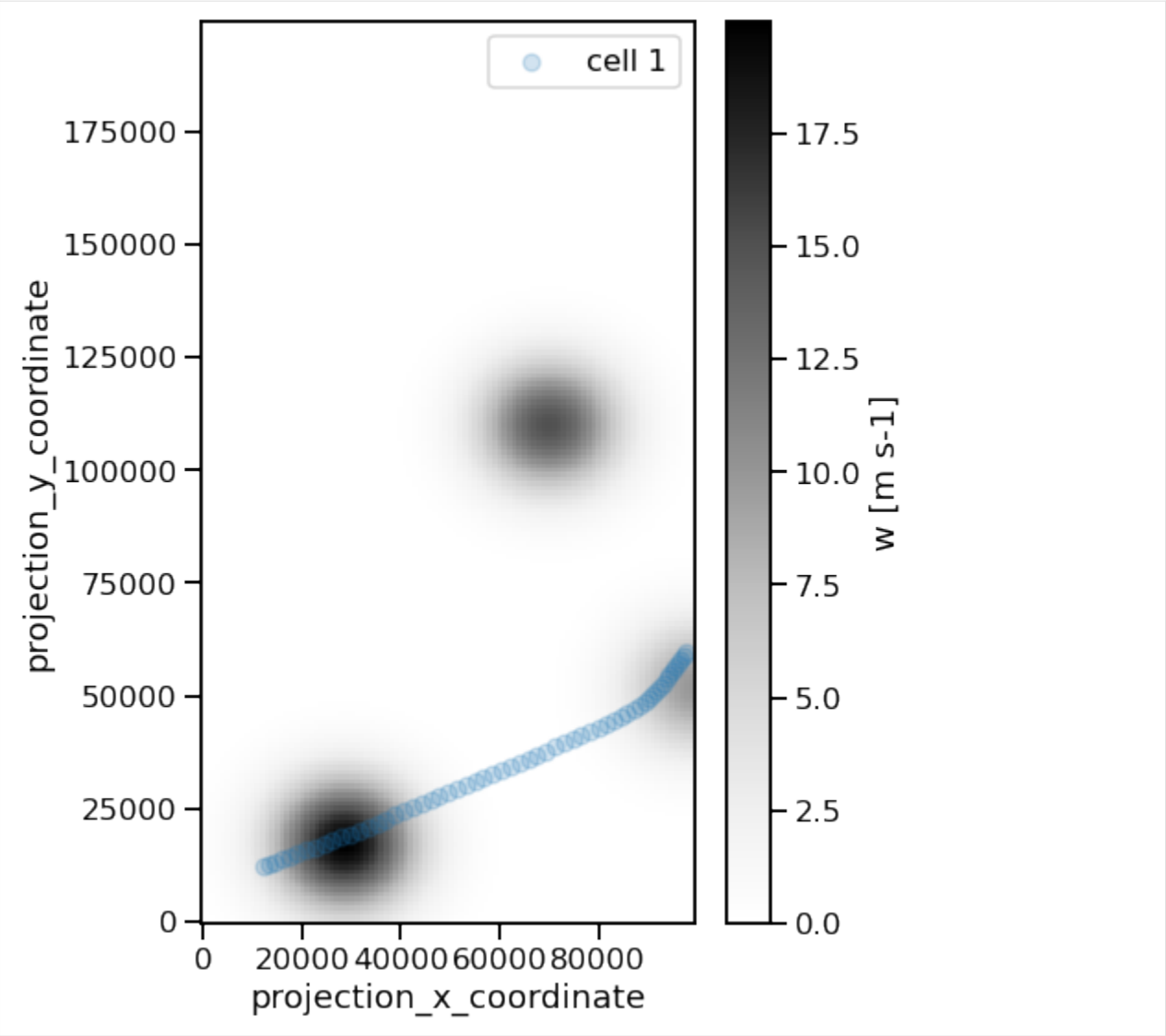
Analogius to the search range, there is a second option for that called *time_cell_min*. This defines a minimum of time

in seconds for a cell to appear as tracked:

[18]:
```
track = tobac.themes.tobac_v1.linking_trackpy(
    features,
    data,
    dt=dt,
    dxy=dxy,
    v_max=100,
    time_cell_min=60 * 50,  # means 50 steps of 60 seconds
)

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 9))

data.isel(time=50).plot(ax=ax, x="x", y="y", cmap="Greys")

for i, cell_track in track.groupby("cell"):
    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax,
        marker="o",
        alpha=0.2,
        label="cell {0}".format(int(i)),
    )

plt.legend()
```

```
Frame 69: 2 trajectories present.
```

[18]:
```
<matplotlib.legend.Legend at 0x7f06412030d0>
```



7.5 Gappy Tracks

If the data is noisy, an object may disappear for a few frames and then reappear. Such features can still be linked by setting the `memory` parameter with an integer. This defines the number of timeframes a feature can vanish and still get tracked as one cell. We demonstrate this on a simple dataset with only one feature:

[19]:
```
data = tobac.testing.make_simple_sample_data_2D()
dxy, dt = tobac.utils.get_spacings(data)
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, threshold=1
)
track = tobac.themes.tobac_v1.linking_trackpy(
    features, data, dt=dt, dxy=dxy, v_max=1000
)
```

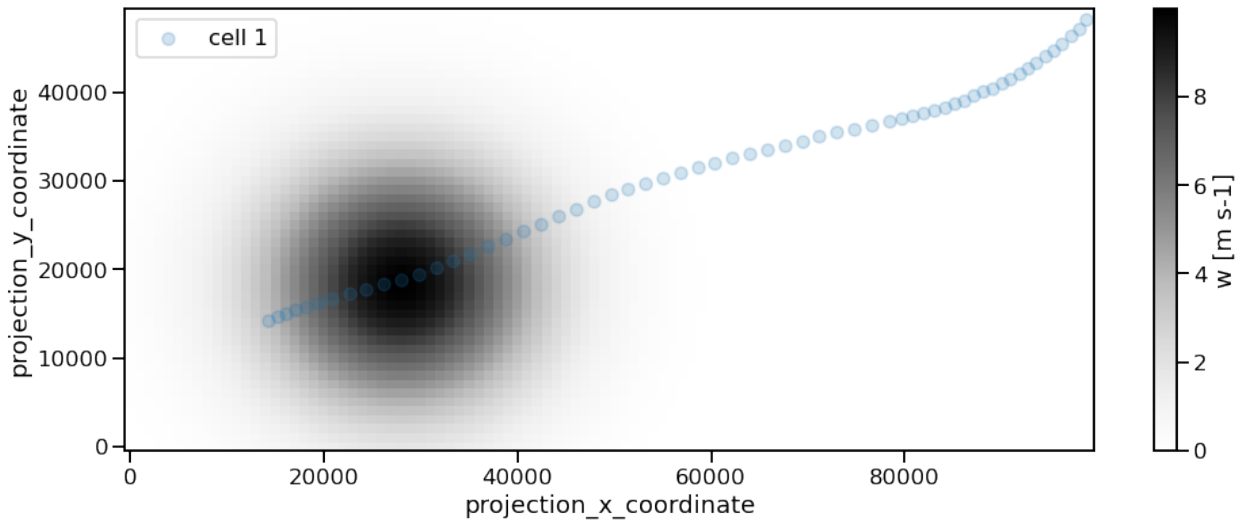
(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(16, 6))

data.isel(time=10).plot(ax=ax, x="x", y="y", cmap="Greys")

for i, cell_track in track.groupby("cell"):
    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax,
        marker="o",
        alpha=0.2,
        label="cell {0}".format(int(i)),
    )

plt.legend()
```

```
Frame 59: 1 trajectories present.
```

[19]:
```
<matplotlib.legend.Legend at 0x7f0641ccabd0>
```



To simulate a gap in the data we set 5 timeframes to 0:

[20]:
```
data[30:35] = data[30] * 0
```

If we apply feature detection and linking to this modified dataset, we will now get two cells:

[21]:
```
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, threshold=1
)
track = tobac.themes.tobac_v1.linking_trackpy(
    features, data, dt=dt, dxy=dxy, v_max=1000
)

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(16, 6))

data.isel(time=10).plot(ax=ax, x="x", y="y", cmap="Greys")
```

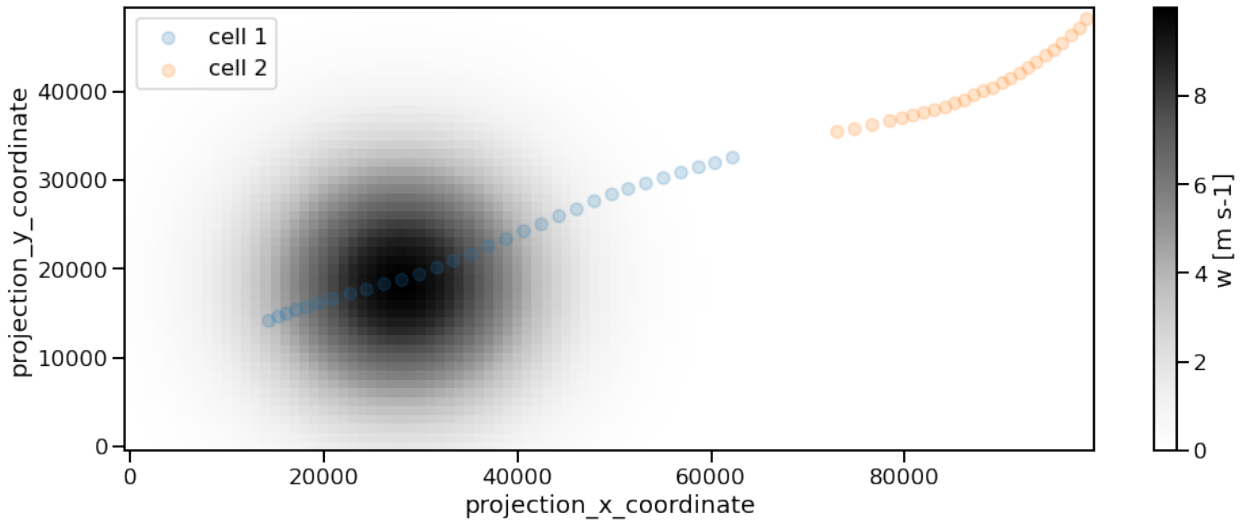
(continues on next page)

(continued from previous page)

```
for i, cell_track in track.groupby("cell"):
    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
        ax=ax,
        marker="o",
        alpha=0.2,
        label="cell {0}".format(int(i)),
    )

plt.legend()
```

```
Frame 59: 1 trajectories present.
```

[21]:
```
<matplotlib.legend.Legend at 0x7f064d10b110>
```



We can avoid that by setting `memory` to a sufficiently high number. 5 is of course sufficient in our case as the situation is artificially created, but with real data this would require some fine tuning. Keep in mind that the search radius needs to be large enough to reach the next feature position. This is the reason for setting `v_max` to 1000 in the linking process.

[22]:
```
features = tobac.themes.tobac_v1.feature_detection_multithreshold(
    data, dxy, threshold=1
)
track = tobac.themes.tobac_v1.linking_trackpy(
    features, data, dt=dt, dxy=dxy, v_max=1000, memory=5
)

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(16, 6))

data.isel(time=10).plot(ax=ax, cmap="Greys")

for i, cell_track in track.groupby("cell"):
    cell_track.plot.scatter(
        x="projection_x_coordinate",
        y="projection_y_coordinate",
```

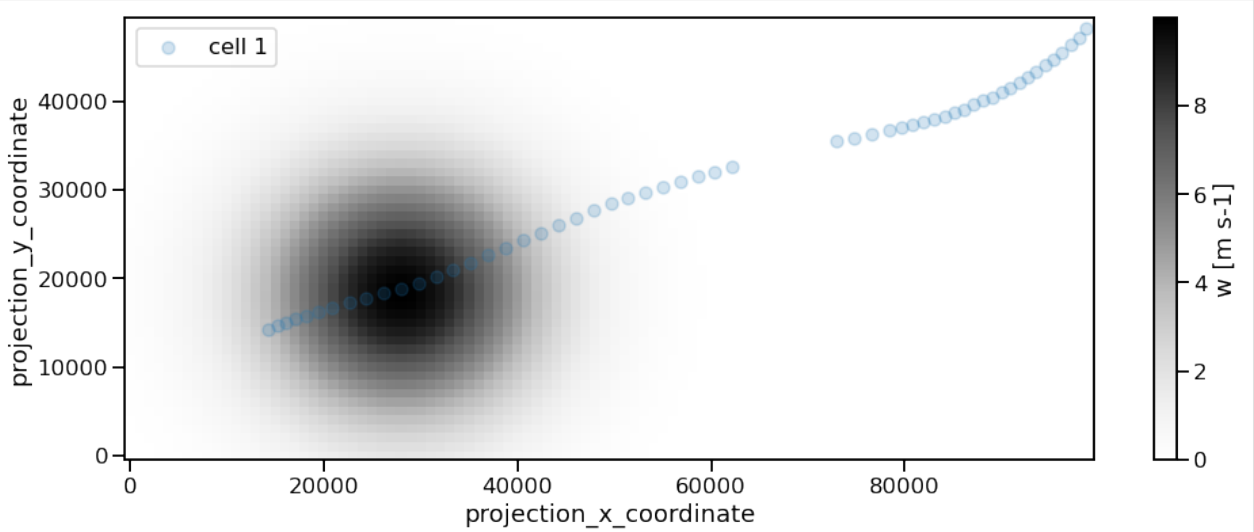
(continues on next page)

(continued from previous page)

```
        ax=ax,
        marker="o",
        alpha=0.2,
        label="cell {0}".format(int(i)),
    )

plt.legend()
```

```
Frame 59: 1 trajectories present.
```

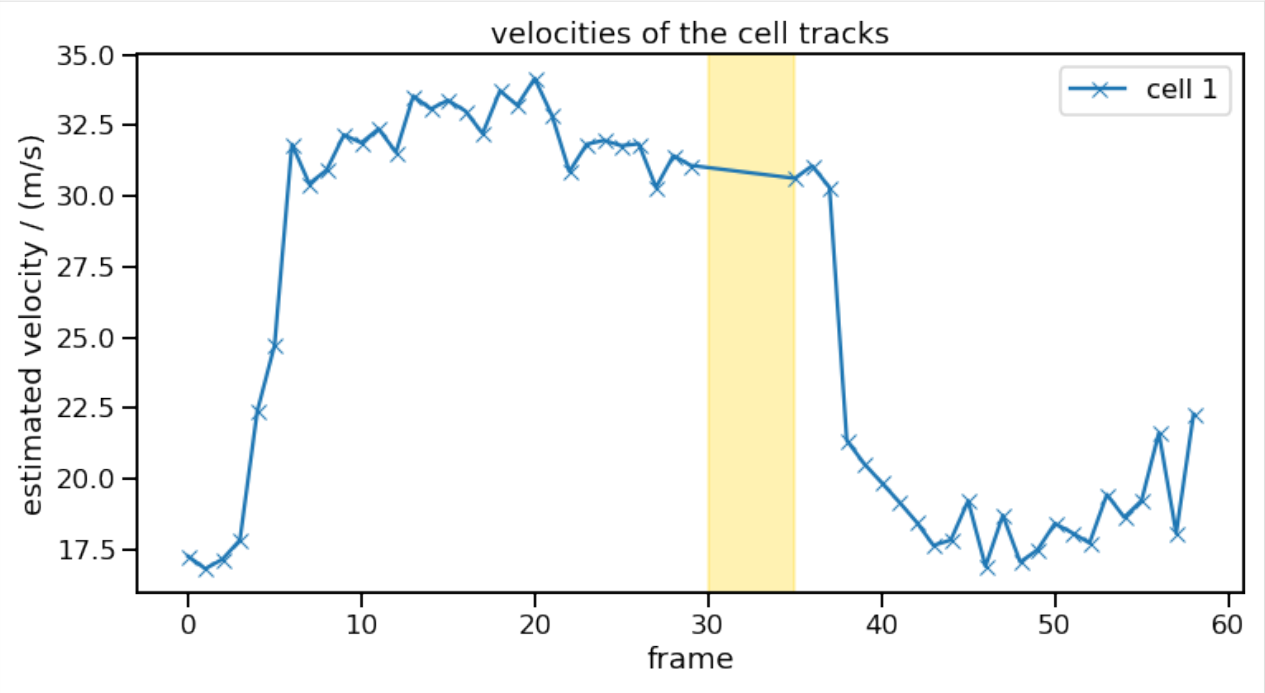
[22]:
```
<matplotlib.legend.Legend at 0x7f064d07b9d0>
```



[23]:
```
vel = tobac.analysis.calculate_velocity(track)
```

[24]:
```
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(12, 6))

for i, vc in vel.groupby("cell"):

    # get velocity as xarray Dataset
    v = vc.to_xarray()["v"]
    v = v.assign_coords({"frame": vc.frame})

    v.plot(x="frame", ax=ax, label="cell {0}".format(int(i)), marker="x")

    ax.set_ylabel("estimated velocity / (m/s)")

ax.axvspan(30, 35, color="gold", alpha=0.3)
ax.set_title("velocities of the cell tracks")
plt.legend()
```

[24]:
```
<matplotlib.legend.Legend at 0x7f0642a91650>
```



Velocity calculations work well across the gap (marked with yellow above). The initial and final drop in speed comes again from edge effects.

7.6 Other Parameters

The parameters `d_min_`, `extrapolate` and `order` do not have a working implmentation in tobac V2 at the moment.

CHAPTER

EIGHT

TOBAC EXAMPLE: TRACKING DEEP CONVECTION BASED ON OLR FROM GEOSTATIONARY SATELLITE RETRIEVALS

This example notebook demonstrates the use of tobac to track isolated deep convective clouds based on outgoing longwave radiation (OLR) calculated based on a combination of two different channels of the GOES-13 imaging instrument.

The data used in this example is downloaded from "zenodo link" automatically as part of the notebooks (This only has to be done once for all the tobac example notebooks).

[3]:
```
# Import libraries:
import xarray
import numpy as np
import pandas as pd
import os
from six.moves import urllib
from glob import glob

import matplotlib.pyplot as plt
%matplotlib inline
```

[1]:
```
# Import tobac itself:
import tobac
```

[4]:
```
# Disable a few warnings:
import warnings
warnings.filterwarnings('ignore', category=UserWarning, append=True)
warnings.filterwarnings('ignore', category=RuntimeWarning, append=True)
warnings.filterwarnings('ignore', category=FutureWarning, append=True)
warnings.filterwarnings('ignore',category=pd.io.pytables.PerformanceWarning)
```

Download example data:

This has to be done only once for all tobac examples.

[5]:
```
data_out='../'
```

[5]:
```
# # Download the data: This only has to be done once for all tobac examples and can take␣
↪a while
# file_path='https://zenodo.org/record/3195910/files/climate-processes/tobac_example_data-
```

(continues on next page)

(continued from previous page)

```
↪v1.0.1.zip'
# tempfile='temp.zip'
# print('start downloading data')
# request=urllib.request.urlretrieve(file_path,tempfile)
# print('start extracting data')
# zf = zipfile.ZipFile(tempfile)
# zf.extractall(data_out)
# print('example data saved in')
```

Load data:

[6]:
```
data_file=os.path.join(data_out,'*','data','Example_input_OLR_satellite.nc')
data_file = glob(data_file)[0]
```

[10]:
```
# Load Data from downloaded file:
OLR=xarray.open_dataset(data_file)['olr']
```

[9]:
```
# Display information about the input data cube:
display(OLR)
```

```
<iris 'Cube' of OLR / (W m^-2) (time: 54; latitude: 131; longitude: 184)>
```

[12]:
```
#Set up directory to save output and plots:
savedir='Save'
if not os.path.exists(savedir):
    os.makedirs(savedir)
plot_dir="Plot"
if not os.path.exists(plot_dir):
    os.makedirs(plot_dir)
```

Feature identification:

Identify features based on OLR field and a set of threshold values

[13]:
```
# Determine temporal and spatial sampling of the input data:
dxy,dt=tobac.utils.get_spacings(OLR,grid_spacing=4000)
```

```
(<xarray.DataArray 'olr' (time: 54, lat: 131, lon: 184)>
[1301616 values with dtype=float64]
Coordinates:
  * time     (time) datetime64[ns] 2013-06-19T19:02:22 ... 2013-06-20T02:45:18
  * lat      (lat) float64 28.03 28.07 28.11 28.15 ... 32.88 32.92 32.96 32.99
  * lon      (lon) float64 -94.99 -94.95 -94.91 -94.87 ... -88.08 -88.04 -88.01
Attributes:
    long_name:  OLR
    units:      W m^-2,)
{'grid_spacing': 4000}
converting xarray to iris and back
(<iris 'Cube' of OLR / (W m^-2) (time: 54; latitude: 131; longitude: 184)>,)
{'grid_spacing': 4000}
(4000, 476)
```

[14]:
```
# Keyword arguments for the feature detection step
parameters_features={}
parameters_features['position_threshold']='weighted_diff'
parameters_features['sigma_threshold']=0.5
parameters_features['min_num']=4
parameters_features['target']='minimum'
parameters_features['threshold']=[250,225,200,175,150]
```

[15]:
```
# Feature detection and save results to file:
print('starting feature detection')
Features=tobac.themes.tobac_v1.feature_detection_multithreshold(OLR,dxy,**parameters_
↪features)
Features.to_netcdf(os.path.join(savedir,'Features.nc'))
print('feature detection performed and saved')
```

```
starting feature detection
(      frame  idx      hdim_1      hdim_2  num  threshold_value  feature
0         0    2    0.737149  177.294423    4              250        1
1         0    3    3.354292  162.980569    9              250        2
2         0    5   32.000000  138.000000    1              250        3
3         0    8   37.479003  146.971379    6              250        4
4         0   13   65.790337    2.067482    3              250        5
...     ...  ...         ...         ...  ...              ...      ...
2733     53   24   64.980999   91.292535   16              225     2734
2734     53   25   83.297251  157.068621   89              225     2735
2735     53   26  103.000000    1.000000    1              225     2736
2736     53   27  129.853687   11.461736    4              225     2737
2737     53   29   46.832556   29.433805   24              200     2738

[2738 rows x 7 columns], <iris 'Cube' of OLR / (W m^-2) (time: 54; latitude: 131;␣
↪longitude: 184)>)
{}
      frame  idx      hdim_1      hdim_2  num  threshold_value  feature  \
0         0    2    0.737149  177.294423    4              250        1
1         0    3    3.354292  162.980569    9              250        2
2         0    5   32.000000  138.000000    1              250        3
3         0    8   37.479003  146.971379    6              250        4
4         0   13   65.790337    2.067482    3              250        5
...     ...  ...         ...         ...  ...              ...      ...
2733     53   24   64.980999   91.292535   16              225     2734
2734     53   25   83.297251  157.068621   89              225     2735
2735     53   26  103.000000    1.000000    1              225     2736
2736     53   27  129.853687   11.461736    4              225     2737
2737     53   29   46.832556   29.433805   24              200     2738

                    time             timestr   latitude  longitude
0     2013-06-19 19:02:22  2013-06-19 19:02:22  28.060909 -88.223109
1     2013-06-19 19:02:22  2013-06-19 19:02:22  28.160780 -88.769332
2     2013-06-19 19:02:22  2013-06-19 19:02:22  29.253914 -89.722603
3     2013-06-19 19:02:22  2013-06-19 19:02:22  29.462995 -89.380251
4     2013-06-19 19:02:22  2013-06-19 19:02:22  30.543369 -94.909851
...                   ...                  ...        ...        ...
```

(continues on next page)

(continued from previous page)

```
2733  2013-06-20 02:45:18  2013-06-20 02:45:18  30.512484 -91.504982
2734  2013-06-20 02:45:18  2013-06-20 02:45:18  31.211441 -88.994935
2735  2013-06-20 02:45:18  2013-06-20 02:45:18  31.963307 -94.950587
2736  2013-06-20 02:45:18  2013-06-20 02:45:18  32.988057 -94.551362
2737  2013-06-20 02:45:18  2013-06-20 02:45:18  29.819931 -93.865540

[2738 rows x 11 columns]
      frame  idx      hdim_1      hdim_2  num  threshold_value  feature  \
0         0    2    0.737149  177.294423    4              250        1
1         0    3    3.354292  162.980569    9              250        2
2         0    5   32.000000  138.000000    1              250        3
3         0    8   37.479003  146.971379    6              250        4
4         0   13   65.790337    2.067482    3              250        5
...     ...  ...         ...         ...  ...              ...      ...
2733     53   24   64.980999   91.292535   16              225     2734
2734     53   25   83.297251  157.068621   89              225     2735
2735     53   26  103.000000    1.000000    1              225     2736
2736     53   27  129.853687   11.461736    4              225     2737
2737     53   29   46.832556   29.433805   24              200     2738

                    time              timestr   latitude  longitude
0     2013-06-19 19:02:22  2013-06-19 19:02:22  28.060909 -88.223109
1     2013-06-19 19:02:22  2013-06-19 19:02:22  28.160780 -88.769332
2     2013-06-19 19:02:22  2013-06-19 19:02:22  29.253914 -89.722603
3     2013-06-19 19:02:22  2013-06-19 19:02:22  29.462995 -89.380251
4     2013-06-19 19:02:22  2013-06-19 19:02:22  30.543369 -94.909851
...                   ...                  ...        ...        ...
2733  2013-06-20 02:45:18  2013-06-20 02:45:18  30.512484 -91.504982
2734  2013-06-20 02:45:18  2013-06-20 02:45:18  31.211441 -88.994935
2735  2013-06-20 02:45:18  2013-06-20 02:45:18  31.963307 -94.950587
2736  2013-06-20 02:45:18  2013-06-20 02:45:18  32.988057 -94.551362
2737  2013-06-20 02:45:18  2013-06-20 02:45:18  29.819931 -93.865540

[2738 rows x 11 columns]
['time', 'lat', 'lon']
feature detection performed and saved
```

Segmentation:
Segmentation is performed based on the OLR field and a threshold value to determine the cloud areas.

[16]:
```
# Keyword arguments for the segmentation step:
parameters_segmentation={}
parameters_segmentation['target']='minimum'
parameters_segmentation['method']='watershed'
parameters_segmentation['threshold']=250
```

[18]:
```
# Perform segmentation and save results to files:
Mask_OLR,Features_OLR=tobac.themes.tobac_v1.segmentation(Features,OLR,dxy,**parameters_
↪segmentation)
```

(continues on next page)

(continued from previous page)

```
print('segmentation OLR performed, start saving results to files')
Mask_OLR.to_netcdf(os.path.join(savedir,'Mask_Segmentation_OLR.nc'))
Features_OLR.to_netcdf(os.path.join(savedir,'Features_OLR.nc'))
print('segmentation OLR performed and saved')
```

```
<xarray.DataArray 'olr' (time: 54, lat: 131, lon: 184)>
array([[[306.847749, 306.847749, ..., 263.987914, 279.675285],
        [306.847749, 306.847749, ..., 284.331736, 285.586608],
        ...,
        [290.318092, 290.318092, ..., 294.654012, 294.654012],
        [290.318092, 290.318092, ..., 294.654012, 297.020815]],

       [[306.847749, 304.470854, ..., 272.836495, 272.836495],
        [304.470854, 304.470854, ..., 281.993749, 285.586608],
        ...,
        [290.318092, 290.318092, ..., 297.020815, 294.654012],
        [290.318092, 290.318092, ..., 297.020815, 294.654012]],

       ...,

       [[305.300037, 305.300037, ..., 294.250334, 294.250334],
        [306.847749, 306.847749, ..., 294.250334, 294.250334],
        ...,
        [261.385907, 248.390063, ..., 290.903128, 288.574062],
        [258.106908, 258.106908, ..., 290.903128, 288.574062]],

       [[306.847749, 306.847749, ..., 294.250334, 294.250334],
        [306.847749, 306.847749, ..., 294.250334, 294.250334],
        ...,
        [260.32182 , 254.87151 , ..., 290.903128, 288.574062],
        [258.106908, 255.910795, ..., 290.903128, 288.574062]]])
Coordinates:
  * time     (time) datetime64[ns] 2013-06-19T19:02:22 ... 2013-06-20T02:45:18
  * lat      (lat) float64 28.03 28.07 28.11 28.15 ... 32.88 32.92 32.96 32.99
  * lon      (lon) float64 -94.99 -94.95 -94.91 -94.87 ... -88.08 -88.04 -88.01
Attributes:
    long_name:  OLR
    units:      W m^-2
segmentation OLR performed, start saving results to files
segmentation OLR performed and saved
```

Trajectory linking:

The detected features are linked into cloud trajectories using the trackpy library (http://soft-matter.github.io/trackpy). This takes the feature positions determined in the feature detection step into account but does not include information on the shape of the identified objects.

[19]:
```
# keyword arguments for linking step
parameters_linking={}
parameters_linking['v_max']=20
```

(continues on next page)

(continued from previous page)

```
parameters_linking['stubs']=2
parameters_linking['order']=1
parameters_linking['extrapolate']=1
parameters_linking['memory']=0
parameters_linking['adaptive_stop']=0.2
parameters_linking['adaptive_step']=0.95
parameters_linking['subnetwork_size']=100
parameters_linking['method_linking']= 'predict'
```

[20]:
```
# Perform linking and save results to file:
Track=tobac.themes.tobac_v1.linking_trackpy(Features,OLR,dt=dt,dxy=dxy,**parameters_
↪linking)
Track.to_netcdf(os.path.join(savedir,'Track.nc'))
```

```
Frame 53: 19 trajectories present.
```

Visualisation:

[19]:
```
# Set extent of maps created in the following cells:
axis_extent=[-95,-89,28,32]
```

[20]:
```
# Plot map with all individual tracks:
import cartopy.crs as ccrs
fig_map,ax_map=plt.subplots(figsize=(10,10),subplot_kw={'projection': ccrs.PlateCarree()}
↪)
ax_map=tobac.plot.map_tracks(Track,axis_extent=axis_extent,axes=ax_map)
```

[21]:
```
# Create animation of tracked clouds and outlines with OLR as a background field
animation_test_tobac=tobac.plot.animation_mask_field(Track,Features,OLR,Mask_OLR,
                                          axis_extent=axis_extent,#figsize=figsize,
↪orientation_colorbar='horizontal',pad_colorbar=0.2,
                                          vmin=80,vmax=330,cmap='Blues_r',
                                          plot_outline=True,plot_marker=True,marker_
↪track='x',plot_number=True,plot_features=True)
```

[23]:
```
# Display animation:
from IPython.display import HTML, Image, display
HTML(animation_test_tobac.to_html5_video())
```

[23]:
```
<IPython.core.display.HTML object>
```

[ ]:
```
# # Save animation to file:
# savefile_animation=os.path.join(plot_dir,'Animation.mp4')
# animation_test_tobac.save(savefile_animation,dpi=200)
# print(f'animation saved to {savefile_animation}')
```

[22]:
```
# Lifetimes of tracked clouds:
fig_lifetime,ax_lifetime=plt.subplots()
tobac.plot.plot_lifetime_histogram_bar(Track,axes=ax_lifetime,bin_edges=np.arange(0,200,
↪20),density=False,width_bar=10)
ax_lifetime.set_xlabel('lifetime (min)')
ax_lifetime.set_ylabel('counts')
```

[22]:
```
Text(0, 0.5, 'counts')
```

[ ]:
```

```

CHAPTER

NINE

TOBAC EXAMPLE: TRACKING DEEP CONVECTION BASED ON VIS FROM GEOSTATIONARY SATELLITE RETRIEVALS

This example notebook demonstrates the use of tobac to track isolated deep convective clouds based on radiances within the VIS using channel 2 (red light - 600 nm) of the GOES-16 imaging instrument in 5-min resolution. The study area is loacted within the CONUS extent of the GOES-E for investigating the formation of deep convection over the Carribean, following the EUREC4A initiative (http://eurec4a.eu/).

The data used in this example is saved on the cloud of the Amazon Web Services, providing an efficient way of processing satellite data without facing the need of downloading the data.

In this example, the Cloud and Moisture Imagery data (ABI-L2-CMIPC) data set is used for the cloud tracking. The product contains one or more Earth-view images with pixel values identifying brightness values that are scaled to support visual analysis. A resolution of 500 m in the VIS channels of the imager ensures a majority of features to be detectable. Also, the data includes quality information as well as information on the solar zenith angle.

Further information on the AWS and provided data sets can be found here.

9.1 Accessing GOES data

Configurations for conducting the example:

[14]:
```
# Import libraries:
import requests
import netCDF4
import boto3
from botocore import UNSIGNED
from botocore.config import Config

import xarray
import rioxarray
import rasterio
import numpy as np
import pandas as pd
import os
from six.moves import urllib
from glob import glob

import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# Import tobac itself:
import tobac
```

[3]:
```
# Disable a few warnings:
import warnings
warnings.filterwarnings('ignore', category=UserWarning, append=True)
warnings.filterwarnings('ignore', category=RuntimeWarning, append=True)
warnings.filterwarnings('ignore', category=FutureWarning, append=True)
warnings.filterwarnings('ignore',category=pd.io.pytables.PerformanceWarning)
```

Build access to Amazon Web Server where GOES-16 data are stored:

[4]:
```
# For accessing data from AWS bucket first define specifics:
bucket_name = 'noaa-goes16'
product_name = 'ABI-L2-CMIPC'
year = 2020
day_of_year = 45
hour = 18
band = 2
```

[5]:
```
# Initialize an s3 client:
s3_client = boto3.client('s3', config=Config(signature_version=UNSIGNED))
```

[6]:
```
# Function for generating file name keys for the S3 bucket:
def get_s3_keys(bucket, s3_client, prefix = ''):
    """
    Generate the keys in an S3 bucket.

    :param bucket: Name of the S3 bucket.
    :param prefix: Only fetch keys that start with this prefix (optional).
    """

    kwargs = {'Bucket': bucket}

    if isinstance(prefix, str):
        kwargs['Prefix'] = prefix

    while True:
        resp = s3_client.list_objects_v2(**kwargs)
        for obj in resp['Contents']:
            key = obj['Key']
            if key.startswith(prefix):
                yield key

        try:
            kwargs['ContinuationToken'] = resp['NextContinuationToken']
        except KeyError:
            break
```

[7]:
```
# Retrieve the keys for of the file names:
keys = get_s3_keys(bucket_name,
                   s3_client,
```

(continues on next page)

(continued from previous page)

```
                    prefix = f'{product_name}/{year}/{day_of_year:03.0f}/{hour:02.0f}/OR_
↪{product_name}-M6C{band:02.0f}'
                   )


keys = [key for key in keys][0:6]
```

Request data from AWS S3 server using the keys and store all files to one data set:

[8]:
```
for k in range(len(keys)):
    resp = requests.get(f'https://{bucket_name}.s3.amazonaws.com/{keys[k]}')
    file_name = keys[k].split('/')[-1].split('.')[0]
    nc4_ds = netCDF4.Dataset(file_name, memory = resp.content)
    store = xarray.backends.NetCDF4DataStore(nc4_ds)
    if k == 0:
        DS = xarray.open_dataset(store)
    else:
        DS2 = xarray.open_dataset(store)
        DS = xarray.combine_nested([DS, DS2], concat_dim=["t"], combine_attrs = "override
↪")
```

9.2 Processing GOES data

Reprojecting image coordinates:

[9]:
```
# Extract properties to define a projection string:
geos_p = DS['goes_imager_projection']
proj_name = DS.goes_imager_projection.grid_mapping_name[0]
h = DS.goes_imager_projection.perspective_point_height
a = DS.goes_imager_projection.semi_major_axis
b = DS.goes_imager_projection.semi_minor_axis
rf = DS.goes_imager_projection.inverse_flattening
lat_0 = DS.goes_imager_projection.latitude_of_projection_origin
lon_0 = DS.goes_imager_projection.longitude_of_projection_origin
sweep = DS.goes_imager_projection.sweep_angle_axis
pstr = '+proj=geos +h=%f +a=%f +b=%f +rf=%f +lat_0=%f +lon_0=%f' \
    % (h, a, b, rf, lat_0, lon_0)
```

[10]:
```
DS = DS.rio.write_crs(pstr, inplace=True)
```

[11]:
```
# Multiply the original coordinates by the satellite height:
DS["x"] = DS["x"]*h
DS["y"] = DS["y"]*h
```

[12]:
```
# Reduce dataset to only variables needed for further analysis:
DS = DS[["CMI"]]
```

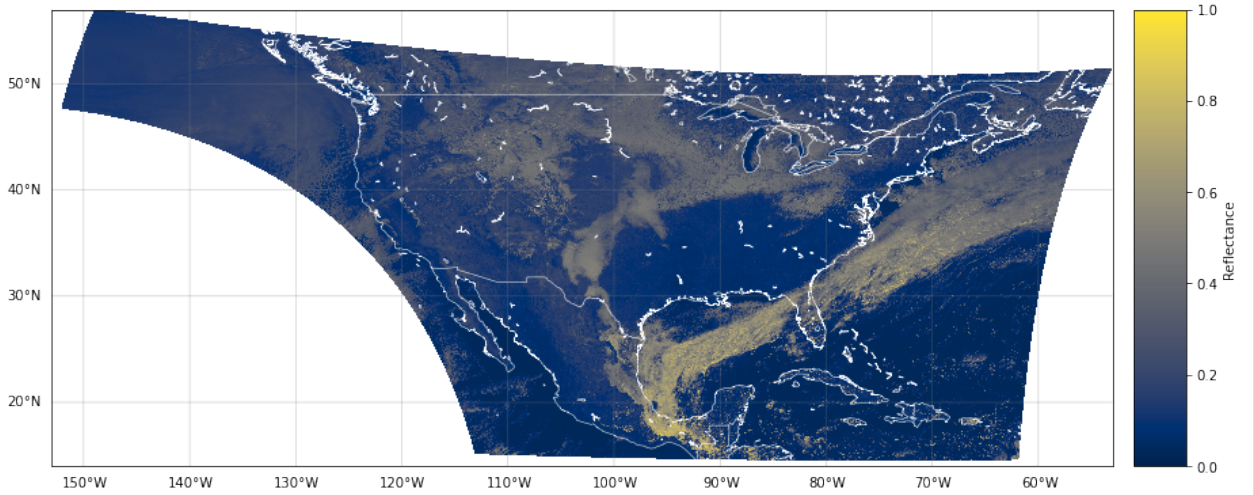
[13]:
```
# Reproject to WGS84 (EPSG: 4326) using rasterio:
DS_R = DS.rio.reproject("epsg:4326")
```

Plot CONUS extent:

[16]:
```
from mpl_toolkits.basemap import Basemap
```

[17]:
```
fig = plt.figure(figsize=(15,15))
ax_0 = fig.add_subplot(1, 1, 1)

cmap = Basemap(projection='cyl',llcrnrlon=-153,llcrnrlat=14,urcrnrlon=-53,urcrnrlat=57,␣
↪resolution='i', ax=ax_0)
con = cmap.pcolormesh(DS_R.x,DS_R.y,DS_R.CMI[0], cmap='cividis', vmin=0,vmax=1)
cmap.drawcountries(color='white', linewidth=.5, ax=ax_0)
cmap.drawcoastlines(linewidth=.5, color='white', ax=ax_0)

cmap.drawparallels(np.arange(10,60,10),labels=[1,0,0,0], linewidth=.5, color="grey")
cmap.drawmeridians(np.arange(-160,-50,10),labels=[0,0,0,1], linewidth=.5, color="grey")

cmap.colorbar(con, shrink=0.6, label='Reflectance', ax=ax_0)
```

[17]:
```
<matplotlib.colorbar.Colorbar at 0x7fa55d020670>
```



Crop the image to a smaller extent, here focussing on the isle of Jamaica:

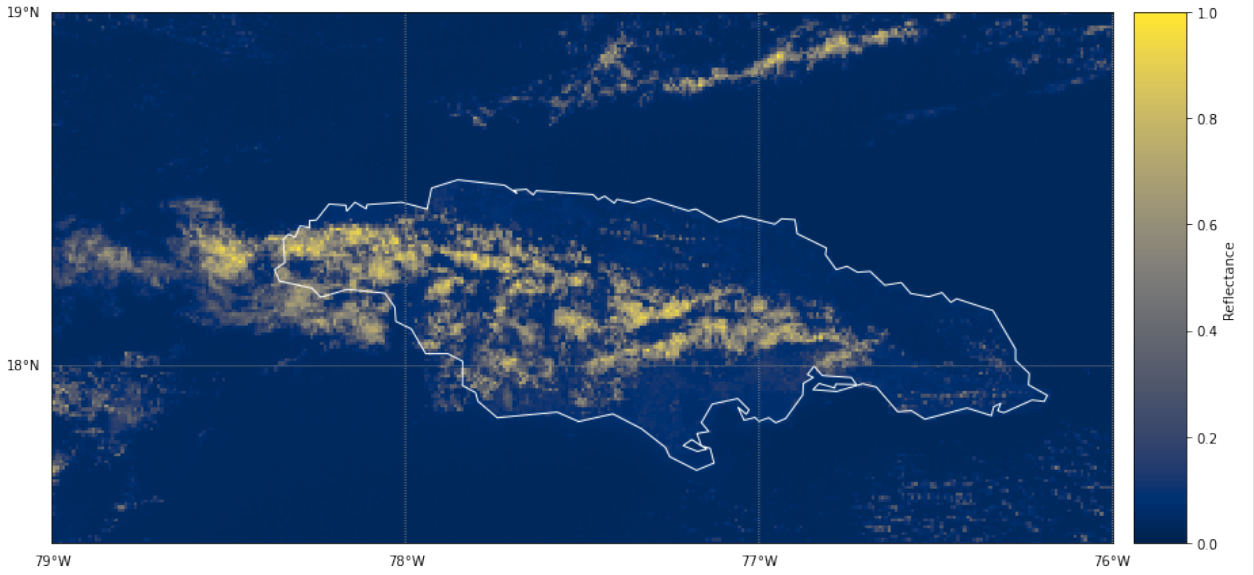
[18]:
```
# Define boundaries of mask used for cropping:
min_lon = -80.0
min_lat = 15.0
max_lon = -75.0
max_lat = 20.0

mask_lon = (DS_R.x >= min_lon) & (DS_R.x <= max_lon)
mask_lat = (DS_R.y >= min_lat) & (DS_R.y <= max_lat)
```

[19]:
```
cropped_ds = DS_R.where(mask_lon & mask_lat, drop=True)
```

Plot cropped image showing Jamaica:

[20]:
```
fig = plt.figure(figsize=(15,15))
ax_1 = fig.add_subplot(1, 1, 1)

bmap = Basemap(projection='cyl',llcrnrlon=-79,llcrnrlat=17.5,urcrnrlon=-76,urcrnrlat=19,␣
↪resolution='i', ax=ax_1)
dat = bmap.pcolormesh(cropped_ds.x,cropped_ds.y,cropped_ds.CMI[0], cmap='cividis',␣
↪vmin=0,vmax=1)
bmap.drawcountries(color='white', linewidth=1, ax=ax_1)
bmap.drawcoastlines(linewidth=1, color='white', ax=ax_1)

bmap.drawparallels(np.arange(10,20,1),labels=[1,0,0,0], linewidth=1, color="grey")
bmap.drawmeridians(np.arange(-80,-70,1),labels=[0,0,0,1], linewidth=1, color="grey")

bmap.colorbar(dat, ax=ax_1, shrink=0.6, label='Reflectance')
```

[20]:
```
<matplotlib.colorbar.Colorbar at 0x7fa55cd97910>
```



9.3 Tobac Cloud Tracking

[21]:
```
# Set up directory to save output and plots:
savedir='Save'
if not os.path.exists(savedir):
    os.makedirs(savedir)
plot_dir="Plot"
if not os.path.exists(plot_dir):
    os.makedirs(plot_dir)
```

[22]:
```
# Change standard_name to be a regular argument of cf-conventions:
cropped_ds.CMI.attrs["standard_name"] = "toa_bidirectional_reflectance"
```

[23]:
```
# Assign former coordinates as variables to avoid dupplicate naming:
new_data = cropped_ds.reset_coords(names = ["y_image", "x_image"])
```

Feature identification: Identify features based on VIS field and a set of threshold values.

[24]:
```
# Determine temporal and spatial sampling of the input data:
dxy,dt=tobac.utils.get_spacings(new_data.CMI,grid_spacing=500)
```

[26]:
```
# Keyword arguments for the feature detection step:
parameters_features={}
parameters_features['position_threshold']='weighted_diff'
parameters_features['sigma_threshold']=0.5
parameters_features['min_num']=8
parameters_features['n_min_threshold']=8
parameters_features['target']='maximum'
parameters_features['threshold']=[0.2,0.4,0.6]
```

[27]:
```
# Feature detection and save results to file:
print('starting feature detection')
Features=tobac.themes.tobac_v1.feature_detection_multithreshold(new_data.CMI,dxy,
↪**parameters_features)
Features.to_netcdf(os.path.join(savedir,'Features.nc'))
print('feature detection performed and saved')
```

```
starting feature detection
      frame   idx      hdim_1      hdim_2  num  threshold_value  feature  \
0         0     6    3.967704  199.359137   13              0.2        1
1         0     9    1.384956  308.325801   11              0.2        2
2         0    11    0.973372  340.422488   25              0.2        3
3         0    14    1.055909  479.695983   18              0.2        4
4         0    27    5.538348    3.698972   32              0.2        5
...     ...   ...         ...         ...  ...              ...      ...
1487      5  1966  335.889404  285.967179    9              0.6     1488
1488      5  1967  422.424521   34.334552   24              0.6     1489
1489      5  1975  472.002232   31.821198   16              0.6     1490
1490      5  1986  522.294278  142.710449   15              0.6     1491
1491      5  2002  538.749685  148.998903   18              0.6     1492

                    time              timestr   latitude  longitude  \
0     2020-02-14 18:02:17  2020-02-14 18:02:17  19.958121 -78.153419
1     2020-02-14 18:02:17  2020-02-14 18:02:17  19.981960 -77.147658
2     2020-02-14 18:02:17  2020-02-14 18:02:17  19.985759 -76.851406
3     2020-02-14 18:02:17  2020-02-14 18:02:17  19.984997 -75.565914
4     2020-02-14 18:02:17  2020-02-14 18:02:17  19.943624 -79.959360
...                   ...                  ...        ...        ...
1487  2020-02-14 18:27:17  2020-02-14 18:27:17  16.894488 -77.354028
1488  2020-02-14 18:27:17  2020-02-14 18:27:17  16.095770 -79.676594
1489  2020-02-14 18:27:17  2020-02-14 18:27:17  15.638168 -79.699792
1490  2020-02-14 18:27:17  2020-02-14 18:27:17  15.173973 -78.676286
1491  2020-02-14 18:27:17  2020-02-14 18:27:17  15.022090 -78.618244

      goes_imager_projection
0                 -78.153419
```

(continues on next page)

(continued from previous page)

```
1                -77.147658
2                -76.851406
3                -75.565914
4                -79.959360
...                     ...
1487             -77.354028
1488             -79.676594
1489             -79.699792
1490             -78.676286
1491             -78.618244

[1492 rows x 12 columns]
['x', 'y', 't', 'goes_imager_projection']
feature detection performed and saved
```

Segmentation:
Segmentation is performed based on the OLR field and a threshold value to determine the cloud areas.

[28]:
```
# Keyword arguments for the segmentation step:
parameters_segmentation={}
parameters_segmentation['target']='maximum'
parameters_segmentation['method']='watershed'
parameters_segmentation['threshold']=0.2
```

[29]:
```
# Perform segmentation and save results to files:
Mask_VIS,Features_VIS=tobac.themes.tobac_v1.segmentation(Features,new_data.CMI,dxy,
↪**parameters_segmentation)
print('segmentation VIS performed, start saving results to files')
Features_VIS.to_netcdf(os.path.join(savedir,'Features_VIS.nc'))
Mask_VIS.to_netcdf(os.path.join(savedir,'Mask_Segmentation_VIS.nc'))
print('segmentation VIS performed and saved')
```

```
<xarray.DataArray 'CMI' (t: 6, y: 542, x: 542)>
array([[[0.17650777, 0.15174589, 0.09904753, ..., 0.07682532,
         0.0873015 , 0.08920626],
        [0.3238092 , 0.05301582, 0.11460306, ..., 0.0809523 ,
         0.08603166, 0.0873015 ],
        [0.1206348 , 0.047619  , 0.05142852, ..., 0.09587292,
         0.0904761 , 0.08222214],
        ...,
        [0.0396825 , 0.03746028, 0.0380952 , ..., 0.04507932,
         0.04253964, 0.04222218],
        [0.03714282, 0.03682536, 0.03841266, ..., 0.04412694,
         0.04539678, 0.04571424],
        [0.03904758, 0.03873012, 0.03936504, ..., 0.04063488,
         0.0412698 , 0.04031742]],

       [[0.11492053, 0.16380937, 0.06952374, ..., 0.07396818,
         0.08825389, 0.09301578],
```

(continues on next page)

(continued from previous page)

```
        [0.14698398, 0.10730148, 0.06285708, ..., 0.07587294,
         0.0825396 , 0.08444437],
        [0.07015866, 0.0714285 , 0.05047614, ..., 0.09079356,
         0.08380944, 0.08285706],
...
        [0.03746028, 0.0380952 , 0.03746028, ..., 0.0587301 ,
         0.03873012, 0.04507932],
        [0.03619044, 0.03841266, 0.03873012, ..., 0.04158726,
         0.04095234, 0.04063488],
        [0.03682536, 0.0365079 , 0.03714282, ..., 0.1158729 ,
         0.03777774, 0.0428571 ]],

       [[0.04507932, 0.04349202, 0.05650788, ..., 0.07015866,
         0.07999992, 0.0825396 ],
        [0.04158726, 0.04380948, 0.04507932, ..., 0.07396818,
         0.07873008, 0.08190469],
        [0.0412698 , 0.0428571 , 0.0412698 , ..., 0.08634912,
         0.08317453, 0.07587294],
        ...,
        [0.04222218, 0.03841266, 0.03746028, ..., 0.0984126 ,
         0.0730158 , 0.0412698 ],
        [0.0380952 , 0.04634916, 0.0682539 , ..., 0.03873012,
         0.03999996, 0.0380952 ],
        [0.0365079 , 0.03714282, 0.03873012, ..., 0.03873012,
         0.04031742, 0.03714282]]], dtype=float32)
Coordinates:
  * x                      (x) float64 -79.99 -79.98 -79.98 ... -75.01 -75.0
  * y                      (y) float64 19.99 19.99 19.98 ... 15.02 15.01 15.0
  * t                      (t) datetime64[ns] 2020-02-14T18:02:17.194589056 ...
    goes_imager_projection  int64 0
Attributes:
    long_name:              ABI L2+ Cloud and Moisture Imagery reflectance fa...
    standard_name:          toa_bidirectional_reflectance
    sensor_band_bit_depth:  12
    valid_range:            [   0 4095]
    units:                  1
    resolution:             y: 0.000014 rad x: 0.000014 rad
    grid_mapping:           goes_imager_projection
    cell_methods:           t: point area: point
    ancillary_variables:    DQF
segmentation VIS performed, start saving results to files
segmentation VIS performed and saved
```

Trajectory linking: The detected features are linked into cloud trajectories using the trackpy library (http://soft-matter.github.io/trackpy). This takes the feature positions determined in the feature detection step into account but does not include information on the shape of the identified objects.

[30]:
```
# Keyword arguments for linking step:
parameters_linking={}
parameters_linking['v_max']=20
parameters_linking['stubs']=2
parameters_linking['order']=1
```

(continues on next page)

(continued from previous page)

```
parameters_linking['extrapolate']=1
parameters_linking['memory']=0
parameters_linking['adaptive_stop']=0.2
parameters_linking['adaptive_step']=0.95
parameters_linking['subnetwork_size']=100
parameters_linking['method_linking']= 'predict'
```

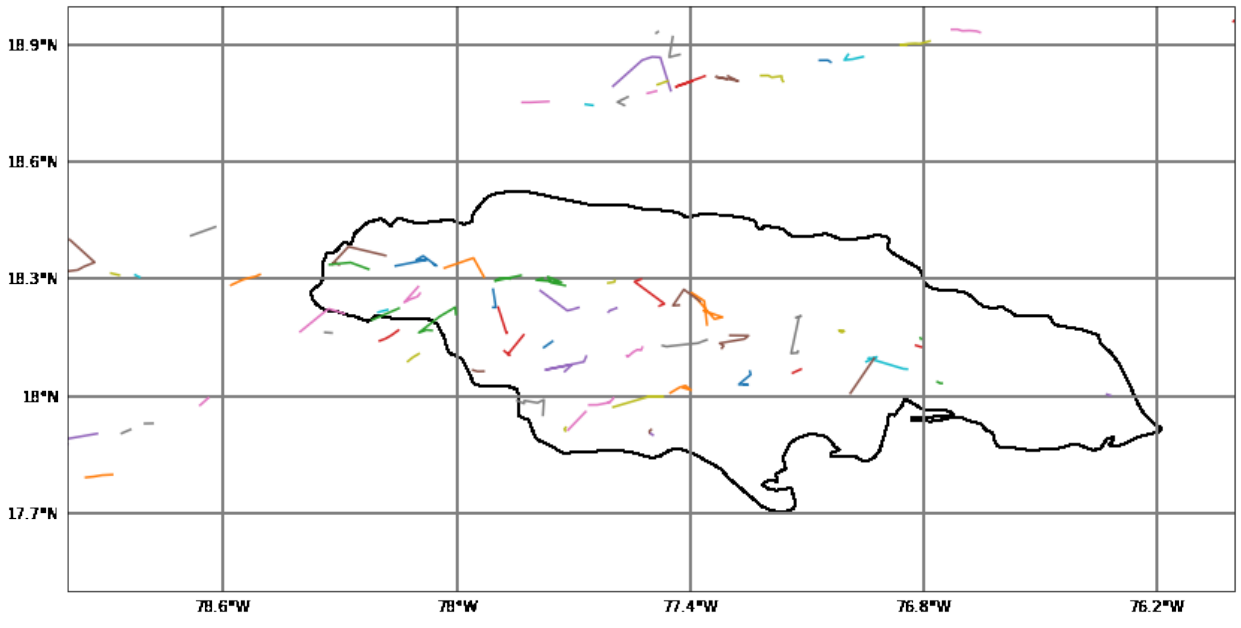
[31]:
```
# Perform linking and save results to file:
Track=tobac.themes.tobac_v1.linking_trackpy(Features,new_data.CMI,dt=dt,dxy=dxy,
↪**parameters_linking)
Track.to_netcdf(os.path.join(savedir,'Track.nc'))
```

```
Frame 5: 257 trajectories present.
```

Visualisation of the tracks:

[32]:
```
axis_extent=[-79,-76,17.5,19] #xmin,xmax,ymin,ymax
```

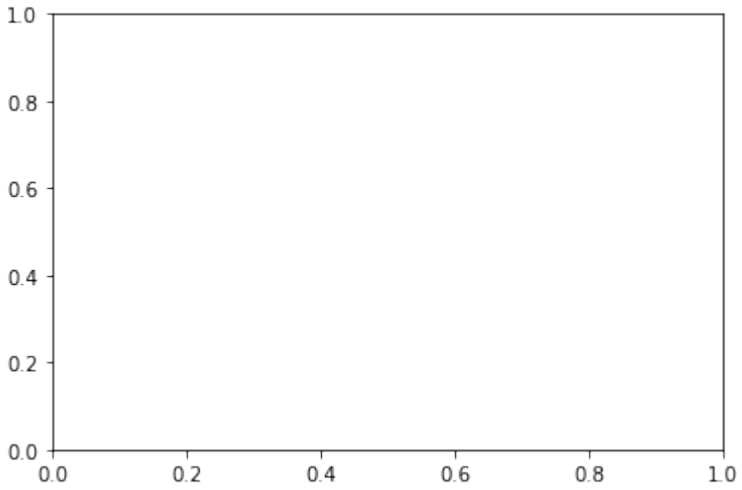
[33]:
```
import cartopy.crs as ccrs

fig_map,ax_map=plt.subplots(figsize=(14,8),subplot_kw={'projection': ccrs.PlateCarree()})
ax_map=tobac.plot.map_tracks(Track,axis_extent=axis_extent,axes=ax_map)
```



[34]:
```
# Create animation of tracked clouds and outlines with VIS as a background field:
animation_test_tobac=tobac.plot.animation_mask_field(Track,Features,new_data.CMI,Mask_
↪VIS,
                                                     axis_extent=axis_extent,figsize=(14,8),
↪#orientation_colorbar='horizontal',pad_colorbar=0.2,
                                                     vmin=0,vmax=1,cmap='Blues_r',linewidth_
↪contour=1,
                                                     plot_outline=True,plot_marker=True,marker_
↪track='x',plot_number=True,plot_features=True)
```

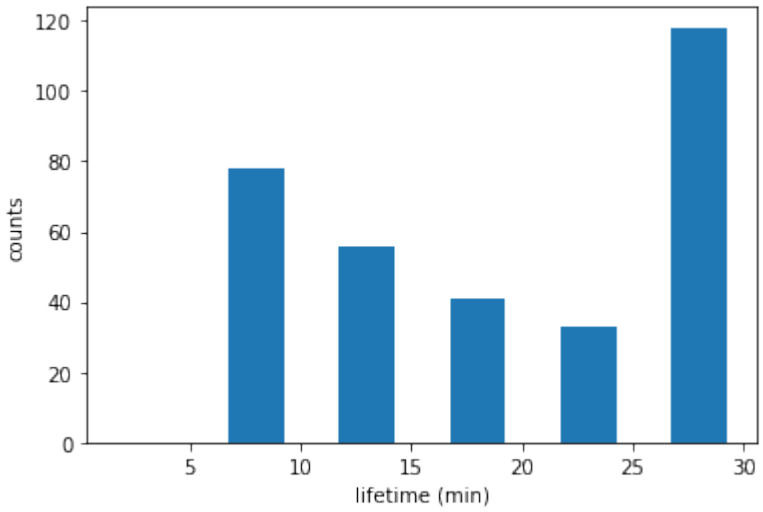
[35]:
```
# Display animation:
from IPython.display import HTML, Image, display
HTML(animation_test_tobac.to_html5_video())
```

[35]:
```
<IPython.core.display.HTML object>
```



[36]:
```
# Lifetimes of tracked clouds:
fig_lifetime,ax_lifetime=plt.subplots()
tobac.plot.plot_lifetime_histogram_bar(Track,axes=ax_lifetime,bin_edges=np.arange(0,35,
↪5),density=False,width_bar=2.5)
ax_lifetime.set_xlabel('lifetime (min)')
ax_lifetime.set_ylabel('counts')
```

[36]:
```
Text(0, 0.5, 'counts')
```



CHAPTER

TEN

TOBAC EXAMPLE: TRACKING OF DEEP CONVECTION BASED ON OLR FROM CONVECTION PERMITTING MODEL SIMULATIONS

This example notebook demonstrates the use of tobac to track deep convection based on the outgoing longwave radiation (OLR) from convection permitting simulations.

The simulation results used in this example were performed as part of the ACPC deep convection intercomparison case study (http://acpcinitiative.org/Docs/ACPC_DCC_Roadmap_171019.pdf) with WRF using the Morrison microphysics scheme. Simulations were performed with a horizontal grid spacing of 4.5 km.

The data used in this example is downloaded from "zenodo link" automatically as part of the notebooks (This only has to be done once for all the tobac example notebooks).

Import libraries:

[9]:
```
# Import a range of python libraries used in this notebook:
import xarray
import numpy as np
import pandas as pd
import os,sys
import shutil
import datetime
from six.moves import urllib
from glob import glob

import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# Import tobac itself:
import tobac
```

[4]:
```
# Disable a few warnings:
import warnings
warnings.filterwarnings('ignore', category=UserWarning, append=True)
warnings.filterwarnings('ignore', category=RuntimeWarning, append=True)
warnings.filterwarnings('ignore', category=FutureWarning, append=True)
warnings.filterwarnings('ignore',category=pd.io.pytables.PerformanceWarning)
```

Download example data:

The actual download is only necessary once for all example notebooks.

[5]:
```
data_out='../'
```

[7]:
```
# # Download the data: This only has to be done once for all tobac examples and can take␣
↪a while
# file_path='https://zenodo.org/record/3195910/files/climate-processes/tobac_example_data-
↪v1.0.1.zip'
# #file_path='http://zenodo..'
# tempfile='temp.zip'
# print('start downloading data')
# request=urllib.request.urlretrieve(file_path,tempfile)
# print('start extracting data')
# shutil.unpack_archive(tempfile,data_out)
# #zf = zipfile.ZipFile(tempfile)
# #zf.extractall(data_out)
# os.remove(tempfile)
# print('data extracted')
```

```
start downloading data
start extracting data
data extracted
```

[6]:
```
data_file=os.path.join(data_out,'*','data','Example_input_OLR_model.nc')
data_file = glob(data_file)[0]
```

[10]:
```
#Load Data from downloaded file:
OLR=xarray.open_dataset(data_file)['OLR']
```

[11]:
```
#Set up directory to save output and plots:
savedir='Save'
if not os.path.exists(savedir):
    os.makedirs(savedir)
plot_dir="Plot"
if not os.path.exists(plot_dir):
    os.makedirs(plot_dir)
```

Feature detection:

Feature detection is performed based on OLR field and a set of thresholds.

[12]:
```
# Determine temporal and spatial sampling:
dxy,dt=tobac.utils.get_spacings(OLR)
```

```
(<xarray.DataArray 'OLR' (time: 96, south_north: 110, west_east: 132)>
[1393920 values with dtype=float32]
Coordinates:
  * time         (time) datetime64[ns] 2013-06-19T19:05:00 ... 2013-06-20T03:00:00
  * south_north  (south_north) int64 156 157 158 159 160 ... 261 262 263 264 265
  * west_east    (west_east) int64 201 202 203 204 205 ... 328 329 330 331 332
    latitude     (south_north, west_east) float32 ...
    longitude    (south_north, west_east) float32 ...
```

(continues on next page)

(continued from previous page)

```
    x            (west_east) float64 ...
    y            (south_north) float64 ...
    x_0          (west_east) int64 ...
    y_0          (south_north) int64 ...
Attributes:
    units:    W m-2,)
{}
converting xarray to iris and back
(<iris 'Cube' of OLR / (W m-2) (time: 96; south_north: 110; west_east: 132)>,)
{}
(4500.0, 300)
```

[14]:
```
# Dictionary containing keyword arguments for feature detection step (Keywords could
↪also be given directly in the function call).
parameters_features={}
parameters_features['position_threshold']='weighted_diff'
parameters_features['sigma_threshold']=0.5
parameters_features['min_num']=4
parameters_features['target']='minimum'
parameters_features['threshold']=[250,225,200,175,150]
```

[15]:
```
# Perform feature detection:
print('starting feature detection')
Features=tobac.themes.tobac_v1.feature_detection_multithreshold(OLR,dxy,**parameters_
↪features)
Features.to_netcdf(os.path.join(savedir,'Features.netcdf'))
print('feature detection performed and saved')
```

```
starting feature detection
(      frame  idx     hdim_1      hdim_2  num  threshold_value  feature
0         0    1  48.000000  104.999952    3              250        1
1         0    2  49.000000  100.000000    1              250        2
2         0    4  59.462142   84.000000    2              250        3
3         0    7  65.000000   67.000000    1              250        4
4         0    9  67.000000   62.346732    2              250        5
...     ...  ...        ...         ...  ...              ...      ...
2716     95   28  67.000000  110.000000    1              175     2717
2717     95   29  71.666555  122.473032   62              175     2718
2718     95   30  69.802409   15.037454   10              175     2719
2719     95   31  75.380948  109.634011   19              175     2720
2720     95   34  88.139481  113.625911   68              150     2721

[2721 rows x 7 columns], <iris 'Cube' of OLR / (W m-2) (time: 96; south_north: 110; west_
↪east: 132)>)
{}
      frame  idx     hdim_1      hdim_2  num  threshold_value  feature  \
0         0    1  48.000000  104.999952    3              250        1
1         0    2  49.000000  100.000000    1              250        2
2         0    4  59.462142   84.000000    2              250        3
3         0    7  65.000000   67.000000    1              250        4
4         0    9  67.000000   62.346732    2              250        5
```

(continues on next page)

(continued from previous page)

```
...     ...  ...        ...         ...  ...             ...      ...
2716     95   28  67.000000  110.000000    1             175     2717
2717     95   29  71.666555  122.473032   62             175     2718
2718     95   30  69.802409   15.037454   10             175     2719
2719     95   31  75.380948  109.634011   19             175     2720
2720     95   34  88.139481  113.625911   68             150     2721

                    time              timestr  south_north   west_east  \
0     2013-06-19 19:05:00  2013-06-19 19:05:00   204.000000  305.999952
1     2013-06-19 19:05:00  2013-06-19 19:05:00   205.000000  301.000000
2     2013-06-19 19:05:00  2013-06-19 19:05:00   215.462142  285.000000
3     2013-06-19 19:05:00  2013-06-19 19:05:00   221.000000  268.000000
4     2013-06-19 19:05:00  2013-06-19 19:05:00   223.000000  263.346732
...                   ...                  ...          ...         ...
2716  2013-06-20 03:00:00  2013-06-20 03:00:00   223.000000  311.000000
2717  2013-06-20 03:00:00  2013-06-20 03:00:00   227.666555  323.473032
2718  2013-06-20 03:00:00  2013-06-20 03:00:00   225.802409  216.037454
2719  2013-06-20 03:00:00  2013-06-20 03:00:00   231.380948  310.634011
2720  2013-06-20 03:00:00  2013-06-20 03:00:00   244.139481  314.625911

      projection_y_coordinate           y              latitude  \
0                9.202500e+05  204.000000   [29.56323250795596]
1                9.247500e+05  205.000000   [29.61300277709961]
2                9.718296e+05  215.462142   [30.06779582764317]
3                9.967500e+05  221.000000  [30.317230224609375]
4                1.005750e+06  223.000000   [30.40456474862914]
...                       ...         ...                   ...
2716             1.005750e+06  223.000000  [30.334190368652344]
2717             1.026749e+06  227.666555  [30.500885027226275]
2718             1.018361e+06  225.802409    [30.5510583635925]
2719             1.043464e+06  231.380948  [30.678966958276725]
2720             1.100878e+06  244.139481   [31.19504789023921]

                longitude  projection_x_coordinate           x
0      [-90.0455955414817]             1.379250e+06  305.999952
1      [-90.2796630859375]             1.356750e+06  301.000000
2     [-91.01834241734284]             1.284750e+06  285.000000
3     [-91.81805419921875]             1.208250e+06  268.000000
4     [-92.03704073277446]             1.187310e+06  263.346732
...                    ...                      ...         ...
2716  [-89.76840209960938]             1.401750e+06  311.000000
2717  [-89.16383151140309]             1.457879e+06  323.473032
2718  [-94.29081414933923]             9.744185e+05  216.037454
2719  [-89.76749015468663]             1.400103e+06  310.634011
2720  [-89.54780638125565]             1.418067e+06  314.625911

[2721 rows x 17 columns]
      frame  idx     hdim_1      hdim_2  num  threshold_value  feature  \
0         0    1  48.000000  104.999952    3              250        1
1         0    2  49.000000  100.000000    1              250        2
2         0    4  59.462142   84.000000    2              250        3
3         0    7  65.000000   67.000000    1              250        4
```

(continues on next page)

(continued from previous page)

```
4         0    9  67.000000   62.346732    2              250        5
...     ...  ...        ...         ...  ...              ...      ...
2716     95   28  67.000000  110.000000    1              175     2717
2717     95   29  71.666555  122.473032   62              175     2718
2718     95   30  69.802409   15.037454   10              175     2719
2719     95   31  75.380948  109.634011   19              175     2720
2720     95   34  88.139481  113.625911   68              150     2721

                    time              timestr  south_north   west_east  \
0     2013-06-19 19:05:00  2013-06-19 19:05:00   204.000000  305.999952
1     2013-06-19 19:05:00  2013-06-19 19:05:00   205.000000  301.000000
2     2013-06-19 19:05:00  2013-06-19 19:05:00   215.462142  285.000000
3     2013-06-19 19:05:00  2013-06-19 19:05:00   221.000000  268.000000
4     2013-06-19 19:05:00  2013-06-19 19:05:00   223.000000  263.346732
...                   ...                  ...          ...         ...
2716  2013-06-20 03:00:00  2013-06-20 03:00:00   223.000000  311.000000
2717  2013-06-20 03:00:00  2013-06-20 03:00:00   227.666555  323.473032
2718  2013-06-20 03:00:00  2013-06-20 03:00:00   225.802409  216.037454
2719  2013-06-20 03:00:00  2013-06-20 03:00:00   231.380948  310.634011
2720  2013-06-20 03:00:00  2013-06-20 03:00:00   244.139481  314.625911

      projection_y_coordinate           y              latitude  \
0                9.202500e+05  204.000000   [29.56323250795596]
1                9.247500e+05  205.000000   [29.61300277709961]
2                9.718296e+05  215.462142   [30.06779582764317]
3                9.967500e+05  221.000000  [30.317230224609375]
4                1.005750e+06  223.000000   [30.40456474862914]
...                       ...         ...                   ...
2716             1.005750e+06  223.000000  [30.334190368652344]
2717             1.026749e+06  227.666555  [30.500885027226275]
2718             1.018361e+06  225.802409    [30.5510583635925]
2719             1.043464e+06  231.380948  [30.678966958276725]
2720             1.100878e+06  244.139481   [31.19504789023921]

                longitude  projection_x_coordinate           x
0      [-90.0455955414817]             1.379250e+06  305.999952
1      [-90.2796630859375]             1.356750e+06  301.000000
2     [-91.01834241734284]             1.284750e+06  285.000000
3     [-91.81805419921875]             1.208250e+06  268.000000
4     [-92.03704073277446]             1.187310e+06  263.346732
...                    ...                      ...         ...
2716  [-89.76840209960938]             1.401750e+06  311.000000
2717  [-89.16383151140309]             1.457879e+06  323.473032
2718  [-94.29081414933923]             9.744185e+05  216.037454
2719  [-89.76749015468663]             1.400103e+06  310.634011
2720  [-89.54780638125565]             1.418067e+06  314.625911

[2721 rows x 17 columns]
['time', 'south_north', 'west_east', 'latitude', 'longitude', 'x', 'y', 'x_0', 'y_0']
feature detection performed and saved
```

Segmentation:
Segmentation is performed with watershedding based on the detected features and a single threshold value.

[16]:
```
# Dictionary containing keyword options for the segmentation step:
parameters_segmentation={}
parameters_segmentation['target']='minimum'
parameters_segmentation['method']='watershed'
parameters_segmentation['threshold']=250
```

[18]:
```
# Perform segmentation and save results:
print('Starting segmentation based on OLR.')
Mask_OLR,Features_OLR=tobac.themes.tobac_v1.segmentation(Features,OLR,dxy,**parameters_
↪segmentation)
print('segmentation OLR performed, start saving results to files')
Mask_OLR.to_netcdf(os.path.join(savedir,'Mask_Segmentation_OLR.nc'))
Features_OLR.to_netcdf(os.path.join(savedir,'Features_OLR.nc'))
print('segmentation OLR performed and saved')
```

```
Starting segmentation based on OLR.
<xarray.DataArray 'OLR' (time: 96, south_north: 110, west_east: 132)>
array([[[289.05252, 288.72162, ..., 280.26904, 280.21326],
        [289.42636, 289.08936, ..., 280.36798, 280.3152 ],
        ...,
        [289.93964, 295.0356 , ..., 283.53815, 274.44553],
        [296.27737, 296.89362, ..., 279.30847, 274.90115]],

       [[289.3221 , 288.9656 , ..., 280.25012, 280.19217],
        [289.7054 , 289.34625, ..., 280.35687, 280.3002 ],
        ...,
        [295.6821 , 286.90628, ..., 286.26834, 275.19684],
        [296.15982, 296.80145, ..., 281.44797, 275.65378]],

       ...,

       [[300.36115, 300.2792 , ..., 278.89587, 278.8872 ],
        [300.26105, 300.16174, ..., 278.63638, 278.67062],
        ...,
        [296.05872, 296.319  , ..., 288.2268 , 288.3182 ],
        [295.82617, 296.04742, ..., 287.981  , 287.89703]],

       [[300.3252 , 300.24365, ..., 278.6296 , 278.71307],
        [300.2927 , 300.2258 , ..., 278.47296, 278.5284 ],
        ...,
        [295.60107, 296.1242 , ..., 288.0909 , 288.09134],
        [295.24384, 295.66608, ..., 287.80396, 287.7296 ]]], dtype=float32)
Coordinates:
  * time         (time) datetime64[ns] 2013-06-19T19:05:00 ... 2013-06-20T03:00:00
  * south_north  (south_north) int64 156 157 158 159 160 ... 261 262 263 264 265
  * west_east    (west_east) int64 201 202 203 204 205 ... 328 329 330 331 332
    latitude     (south_north, west_east) float32 27.684788 ... 32.014114
    longitude    (south_north, west_east) float32 -95.00824 ... -88.65869
    x            (west_east) float64 9.068e+05 9.112e+05 ... 1.492e+06 1.496e+06
```

(continues on next page)

(continued from previous page)

```
    y            (south_north) float64 7.042e+05 7.088e+05 ... 1.195e+06
    x_0          (west_east) int64 201 202 203 204 205 ... 328 329 330 331 332
    y_0          (south_north) int64 156 157 158 159 160 ... 261 262 263 264 265
Attributes:
    units:    W m-2
segmentation OLR performed, start saving results to files
segmentation OLR performed and saved
```

Trajectory linking:

Features are linked into cloud trajectories using the trackpy library (http://soft-matter.github.io/trackpy). This takes the feature positions determined in the feature detection step into account but does not include information on the shape of the identified objects.**

[19]:
```
# Arguments for trajectory linking:
parameters_linking={}
parameters_linking['v_max']=20
parameters_linking['stubs']=2
parameters_linking['order']=1
parameters_linking['extrapolate']=1
parameters_linking['memory']=0
parameters_linking['adaptive_stop']=0.2
parameters_linking['adaptive_step']=0.95
parameters_linking['subnetwork_size']=100
parameters_linking['method_linking']= 'predict'
```

[20]:
```
# Perform linking and save results to file:
Track=tobac.themes.tobac_v1.linking_trackpy(Features,OLR,dt=dt,dxy=dxy,**parameters_
↪linking)
Track.to_netcdf(os.path.join(savedir,'Track.nc'))
```

```
Frame 95: 18 trajectories present.
```

Visualisation:

[17]:
```
# Set extent of maps created in the following cells:
axis_extent=[-95,-89,28,32]
```

[18]:
```
# Plot map with all individual tracks:
import cartopy.crs as ccrs
fig_map,ax_map=plt.subplots(figsize=(10,10),subplot_kw={'projection': ccrs.PlateCarree()}
↪)
ax_map=tobac.plot.map_tracks(Track,axis_extent=axis_extent,axes=ax_map)
```

[19]:
```
# Create animation of tracked clouds and outlines with OLR as a background field
animation_test_tobac=tobac.plot.animation_mask_field(Track,Features,OLR,Mask_OLR,
                                          axis_extent=axis_extent,#figsize=figsize,
↪orientation_colorbar='horizontal',pad_colorbar=0.2,
                                          vmin=80,vmax=330,
```

(continues on next page)

(continued from previous page)

```
                                                plot_outline=True,plot_marker=True,marker_
↪track='x',plot_number=True,plot_features=True)
```

[20]:
```
# Display animation:
from IPython.display import HTML, Image, display
HTML(animation_test_tobac.to_html5_video())
```

[20]:
```
<IPython.core.display.HTML object>
```

[ ]:
```
# # Save animation to file:
# savefile_animation=os.path.join(plot_dir,'Animation.mp4')
# animation_test_tobac.save(savefile_animation,dpi=200)
# print(f'animation saved to {savefile_animation}')
```

[21]:
```
# Lifetimes of tracked clouds:
fig_lifetime,ax_lifetime=plt.subplots()
tobac.plot.plot_lifetime_histogram_bar(Track,axes=ax_lifetime,bin_edges=np.arange(0,200,
↪20),density=False,width_bar=10)
ax_lifetime.set_xlabel('lifetime (min)')
ax_lifetime.set_ylabel('counts')
```

[21]:
```
Text(0, 0.5, 'counts')
```

[ ]:
```

```

CHAPTER

ELEVEN

TOBAC EXAMPLE: TRACKING OF PRECIPITATION FEATURES

This example notebook demonstrates the use of tobac to track precipitation features from isolated deep convective clouds.

The simulation results used in this example were performed as part of the ACPC deep convection intercomparison case study (http://acpcinitiative.org/Docs/ACPC_DCC_Roadmap_171019.pdf) with WRF using the Morrison microphysics scheme.

The data used in this example is downloaded from "zenodo link" automatically as part of the notebooks (This only has to be done once for all the tobac example notebooks).

Import necessary python libraries:

[1]:
```
### a development hack
### !pip install --upgrade ../../../../tobac/
```

[2]:
```
# Import libraries:
import xarray
import numpy as np
import pandas as pd
import os
from six.moves import urllib
from glob import glob

import matplotlib.pyplot as plt
%matplotlib inline
```

[3]:
```
# Import tobac itself
import tobac
```

[4]:
```
# Disable a few warnings:
import warnings
warnings.filterwarnings('ignore', category=UserWarning, append=True)
warnings.filterwarnings('ignore', category=RuntimeWarning, append=True)
warnings.filterwarnings('ignore', category=FutureWarning, append=True)
warnings.filterwarnings('ignore',category=pd.io.pytables.PerformanceWarning)
```

Download example data:

Actual download has to be performed only once for all example notebooks!

[5]:
```
data_out='../'
```

[6]:
```
# # Download the data: This only has to be done once for all tobac examples and can take␣
↪a while
# file_path='https://zenodo.org/record/3195910/files/climate-processes/tobac_example_data-
↪v1.0.1.zip'
# #file_path='http://zenodo..'
# tempfile='temp.zip'
# print('start downloading data')
# request=urllib.request.urlretrieve(file_path,tempfile)
# print('start extracting data')
# shutil.unpack_archive(tempfile,data_out)
# #zf = zipfile.ZipFile(tempfile)
# #zf.extractall(data_out)
# os.remove(tempfile)
# print('data extracted')
```

Load Data from downloaded file:

[7]:
```
data_file = os.path.join(data_out,'*','data','Example_input_Precip.nc')
data_file = glob(data_file)[0]
```

[8]:
```
Precip=xarray.open_dataset(data_file)['surface_precipitation_average']
```

[9]:
```
#display information about the iris cube containing the surface precipitation data:
display(Precip)
```

```
<xarray.DataArray 'surface_precipitation_average' (time: 47, south_north: 198, west_east:
↪ 198)>
[1842588 values with dtype=float32]
Coordinates:
  * time          (time) datetime64[ns] 2013-06-19T20:05:00 ... 2013-06-19T23:55:00
  * south_north   (south_north) int64 281 282 283 284 285 ... 474 475 476 477 478
  * west_east     (west_east) int64 281 282 283 284 285 ... 474 475 476 477 478
    latitude      (south_north, west_east) float32 ...
    longitude     (south_north, west_east) float32 ...
    x             (west_east) float64 ...
    y             (south_north) float64 ...
    x_0           (west_east) int64 ...
    y_0           (south_north) int64 ...
Attributes:
    long_name:  surface_precipitation_average
    units:      mm h-1
```

[10]:
```
#Set up directory to save output and plots:
savedir='Save'
if not os.path.exists(savedir):
    os.makedirs(savedir)
plot_dir="Plot"
if not os.path.exists(plot_dir):
    os.makedirs(plot_dir)
```

Feature detection:

Feature detection is perfomed based on surface precipitation field and a range of thresholds

[11]:
```
# Dictionary containing keyword options (could also be directly given to the function)
parameters_features={}
parameters_features['position_threshold']='weighted_diff'
parameters_features['sigma_threshold']=0.5
parameters_features['min_num']=3
parameters_features['min_distance']=0
parameters_features['sigma_threshold']=1
parameters_features['threshold']=[1,2,3,4,5,10,15] #mm/h
parameters_features['n_erosion_threshold']=0
parameters_features['n_min_threshold']=3
```

[12]:
```
# get temporal and spation resolution of the data
dxy,dt=tobac.utils.get_spacings(Precip)
```

[13]:
```
# Feature detection based on based on surface precipitation field and a range of␣
↪thresholds
print('starting feature detection based on multiple thresholds')
Features=tobac.themes.tobac_v1.feature_detection_multithreshold(Precip,dxy,**parameters_
↪features)
print('feature detection done')
Features.to_netcdf(os.path.join(savedir,'Features.nc'))
print('features saved')
```

```
starting feature detection based on multiple thresholds
      frame  idx      hdim_1      hdim_2  num  threshold_value  feature  \
0         0    1   50.065727  139.857477    9                1        1
1         0   15  120.527119  172.500325    4                1        2
2         0   18  126.779273  145.368401   15                1        3
3         0   34  111.611369  155.452030    4                2        4
4         0   35  111.765231  164.938866    8                2        5
...     ...  ...         ...         ...  ...              ...      ...
1124     46   90  163.334480   35.049366   47               15     1125
1125     46   91  177.119093    1.938903   50               15     1126
1126     46   92  183.889436  100.054108   36               15     1127
1127     46   93  196.870749  177.405664   10               15     1128
1128     46   94  196.936190  191.649040    4               15     1129

                    time              timestr  south_north   west_east  \
0     2013-06-19 20:05:00  2013-06-19 20:05:00   331.065727  420.857477
1     2013-06-19 20:05:00  2013-06-19 20:05:00   401.527119  453.500325
2     2013-06-19 20:05:00  2013-06-19 20:05:00   407.779273  426.368401
3     2013-06-19 20:05:00  2013-06-19 20:05:00   392.611369  436.452030
4     2013-06-19 20:05:00  2013-06-19 20:05:00   392.765231  445.938866
...                   ...                  ...          ...         ...
1124  2013-06-19 23:55:00  2013-06-19 23:55:00   444.334480  316.049366
1125  2013-06-19 23:55:00  2013-06-19 23:55:00   458.119093  282.938903
1126  2013-06-19 23:55:00  2013-06-19 23:55:00   464.889436  381.054108
1127  2013-06-19 23:55:00  2013-06-19 23:55:00   477.870749  458.405664
1128  2013-06-19 23:55:00  2013-06-19 23:55:00   477.936190  472.649040

      projection_y_coordinate           y              latitude  \
0               165782.863285  331.065727   [29.84636174574229]
1               201013.559414  401.527119   [30.16692908466767]
```

(continues on next page)

(continued from previous page)

```
2               204139.636582  407.779273  [30.196498834397495]
3               196555.684682  392.611369  [30.126870992779423]
4               196632.615461  392.765231  [30.127221395688906]
...                       ...         ...                   ...
1124            222417.240247  444.334480   [30.36594369270744]
1125            229309.546339  458.119093   [30.42915134198937]
1126            232694.717873  464.889436   [30.45864828495059]
1127            239185.374437  477.870749  [30.515366411726458]
1128            239218.094905  477.936190  [30.515049777191315]

                 longitude  projection_x_coordinate           x
0     [-94.17201509872564]            210678.738492  420.857477
1     [-93.99689187482306]            227000.162468  453.500325
2     [-94.13995972549196]            213434.200454  426.368401
3     [-94.08731650871061]            218476.015240  436.452030
4     [-94.03722567369418]            223219.433218  445.938866
...                    ...                       ...         ...
1124    [-94.722384409394]            158274.683059  316.049366
1125  [-94.89756987871436]            141719.451486  282.938903
1126   [-94.3777372958404]            190777.054179  381.054108
1127  [-93.96733386588102]            229452.832088  458.405664
1128  [-93.89182112542963]            236574.520064  472.649040

[1129 rows x 17 columns]
['time', 'south_north', 'west_east', 'latitude', 'longitude', 'x', 'y', 'x_0', 'y_0']
feature detection done
features saved
```

Segmentation:

Segmentation is performed based on a watershedding and a threshold value:

[14]:
```
# Dictionary containing keyword arguments for segmentation step:
parameters_segmentation={}
parameters_segmentation['method']='watershed'
parameters_segmentation['threshold']=1  # mm/h mixing ratio
```

[15]:
```
# Perform Segmentation and save resulting mask to NetCDF file:
print('Starting segmentation based on surface precipitation')
Mask,Features_Precip=tobac.themes.tobac_v1.segmentation(Features,Precip,dxy,**parameters_
↪segmentation)
print('segmentation based on surface precipitation performed, start saving results to␣
↪files')
Mask.to_netcdf(os.path.join(savedir,'Mask_Segmentation_precip.nc'))
Features_Precip.to_netcdf(os.path.join(savedir,'Features_Precip.nc'))
print('segmentation surface precipitation performed and saved')
```

```
Starting segmentation based on surface precipitation
<xarray.DataArray 'surface_precipitation_average' (time: 47, south_north: 198, west_east:
↪ 198)>
array([[[0.000000e+00, 0.000000e+00, ..., 0.000000e+00, 0.000000e+00],
        [0.000000e+00, 0.000000e+00, ..., 0.000000e+00, 0.000000e+00],
```

(continues on next page)

(continued from previous page)

```
         ...,
         [0.000000e+00, 0.000000e+00, ..., 0.000000e+00, 0.000000e+00],
         [0.000000e+00, 0.000000e+00, ..., 0.000000e+00, 0.000000e+00]],

        [[0.000000e+00, 0.000000e+00, ..., 0.000000e+00, 0.000000e+00],
         [0.000000e+00, 0.000000e+00, ..., 0.000000e+00, 0.000000e+00],
         ...,
         [0.000000e+00, 0.000000e+00, ..., 0.000000e+00, 0.000000e+00],
         [0.000000e+00, 0.000000e+00, ..., 0.000000e+00, 0.000000e+00]],

        ...,

        [[1.162095e-03, 1.237040e-03, ..., 0.000000e+00, 0.000000e+00],
         [1.077846e-03, 1.167705e-03, ..., 0.000000e+00, 0.000000e+00],
         ...,
         [0.000000e+00, 0.000000e+00, ..., 1.475261e-01, 7.602954e-02],
         [0.000000e+00, 0.000000e+00, ..., 1.065946e-01, 4.400021e-02]],

        [[3.334880e-05, 2.312288e-05, ..., 0.000000e+00, 0.000000e+00],
         [1.528859e-05, 8.571893e-06, ..., 0.000000e+00, 0.000000e+00],
         ...,
         [6.988572e-03, 1.184124e-03, ..., 1.278359e-01, 2.074599e-02],
         [3.475464e-04, 2.722412e-05, ..., 1.691269e-01, 1.051712e-02]]],
      dtype=float32)
Coordinates:
  * time         (time) datetime64[ns] 2013-06-19T20:05:00 ... 2013-06-19T23:55:00
  * south_north  (south_north) int64 281 282 283 284 285 ... 474 475 476 477 478
  * west_east    (west_east) int64 281 282 283 284 285 ... 474 475 476 477 478
    latitude     (south_north, west_east) float32 29.62032 ... 30.515106
    longitude    (south_north, west_east) float32 -94.90857 ... -93.863464
    x            (west_east) float64 1.408e+05 1.412e+05 ... 2.388e+05 2.392e+05
    y            (south_north) float64 1.408e+05 1.412e+05 ... 2.392e+05
    x_0          (west_east) int64 281 282 283 284 285 ... 474 475 476 477 478
    y_0          (south_north) int64 281 282 283 284 285 ... 474 475 476 477 478
Attributes:
    long_name:  surface_precipitation_average
    units:      mm h-1
```

```
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
```

(continues on next page)

(continued from previous page)

```
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
```

(continues on next page)

(continued from previous page)

```
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
```

(continues on next page)

(continued from previous page)

```
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
```

(continues on next page)

(continued from previous page)

```
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
```

```
segmentation based on surface precipitation performed, start saving results to files
segmentation surface precipitation performed and saved
```

Trajectory linking:

Trajectory linking is performed using the trackpy library (http://soft-matter.github.io/trackpy). This takes the feature positions determined in the feature detection step into account but does not include information on the shape of the identified objects.

[16]:
```
# Dictionary containing keyword arguments for the linking step:
parameters_linking={}
parameters_linking['method_linking']='predict'
parameters_linking['adaptive_stop']=0.2
parameters_linking['adaptive_step']=0.95
parameters_linking['extrapolate']=0
parameters_linking['order']=1
parameters_linking['subnetwork_size']=100
parameters_linking['memory']=0
parameters_linking['time_cell_min']=5*60
parameters_linking['method_linking']='predict'
parameters_linking['v_max']=10
parameters_linking['d_min']=2000
```

[17]:
```
# Perform trajectory linking using trackpy and save the resulting DataFrame:
Track=tobac.themes.tobac_v1.linking_trackpy(Features,Precip,dt=dt,dxy=dxy,**parameters_
↪linking)
Track.to_netcdf(os.path.join(savedir,'Track.nc'))
```

```
Frame 46: 18 trajectories present.
```

Visualistation:

[18]:
```
# Set extent for maps plotted in the following cells ( in the form [lon_min,lon_max,lat_
↪min,lat_max])
axis_extent=[-95,-93.8,29.5,30.6]
```

[19]:
```
# Plot map with all individual tracks:
import cartopy.crs as ccrs
fig_map,ax_map=plt.subplots(figsize=(10,10),subplot_kw={'projection': ccrs.PlateCarree()}
↪)
ax_map=tobac.plot.map_tracks(Track,axis_extent=axis_extent,axes=ax_map)
```

[20]:
```
# Create animation showing tracked cells with outline of precipitation features and the␣
↪and surface precipitation as a background field:
animation_tobac=tobac.plot.animation_mask_field(Track, Features, Precip, Mask,
                                                axis_extent=axis_extent,#figsize=figsize,
↪orientation_colorbar='horizontal',pad_colorbar=0.2,
                                                vmin=0,vmax=60,extend='both',cmap='Blues',
                                                interval=500,figsize=(10,10),
                                                plot_outline=True,plot_marker=True,marker_
↪track='x',plot_number=True,plot_features=True);
```

[21]:
```
# Display animation:
from IPython.display import HTML, Image, display
HTML(animation_tobac.to_html5_video())
```

[21]:
```
<IPython.core.display.HTML object>
```

[22]:
```
# # Save animation to file
# savefile_animation=os.path.join(plot_dir,'Animation.mp4')
# animation_tobac.save(savefile_animation,dpi=200)
# print(f'animation saved to {savefile_animation}')
```

[23]:
```
# Lifetimes of tracked features:
fig_lifetime,ax_lifetime=plt.subplots()
tobac.plot.plot_lifetime_histogram_bar(Track,axes=ax_lifetime,bin_edges=np.arange(0,120,
↪10),density=False,width_bar=8)
ax_lifetime.set_xlabel('lifetime (min)')
ax_lifetime.set_ylabel('counts')
```

[23]:
```
Text(0, 0.5, 'counts')
```

CHAPTER

TWELVE

TOBAC EXAMPLE: TRACKING ISOLATED CONVECTION BASED ON UPDRAFT VELOCITY AND TOTAL CONDENSATE

This example notebook demonstrates the use of tobac to track isolated deep convective clouds in cloud-resolving model simulation output based on vertical velocity and total condensate mixing ratio.

The simulation results used in this example were performed as part of the ACPC deep convection intercomparison case study (http://acpcinitiative.org/Docs/ACPC_DCC_Roadmap_171019.pdf) with WRF using the Morrison microphysics scheme.

The data used in this example is downloaded from "zenodo link" automatically as part of the notebooks (This only has to be done once for all the tobac example notebooks).

Import libraries:

[1]:
```
# Import libraries:
import xarray
import numpy as np
import pandas as pd
import os
from six.moves import urllib
from glob import glob

import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# Import tobac itself:
import tobac
```

[3]:
```
#Disable a couple of warnings:
import warnings
warnings.filterwarnings('ignore', category=UserWarning, append=True)
warnings.filterwarnings('ignore', category=RuntimeWarning, append=True)
warnings.filterwarnings('ignore', category=FutureWarning, append=True)
warnings.filterwarnings('ignore',category=pd.io.pytables.PerformanceWarning)
```

Download and load example data:

The actual dowloading is only necessary once for all example notebooks.

[4]:
```
data_out='../'
```

[6]:
```
# # Download the data: This only has to be done once for all tobac examples and can take␣
↪a while
# file_path='https://zenodo.org/record/3195910/files/climate-processes/tobac_example_data-
↪v1.0.1.zip'
# #file_path='http://zenodo..'
# tempfile='temp.zip'
# print('start downloading data')
# request=urllib.request.urlretrieve(file_path,tempfile)
# print('start extracting data')
# shutil.unpack_archive(tempfile,data_out)
# # zf = zipfile.ZipFile(tempfile)
# # zf.extractall(data_out)
# os.remove(tempfile)
# print('data extracted')
```

Load Data from downloaded file:

[7]:
```
data_file_W_mid_max = os.path.join(data_out,'*','data','Example_input_midlevelUpdraft.nc
↪')
data_file_W_mid_max = glob(data_file_W_mid_max)[0]
data_file_TWC = os.path.join(data_out,'*','data','Example_input_Condensate.nc')
data_file_TWC = glob(data_file_TWC)[0]
```

[8]:
```
W_mid_max=xarray.open_dataset(data_file_W_mid_max)['w']
TWC=xarray.open_dataset(data_file_TWC)['twc']
```

[9]:
```
# Display information about the two cubes for vertical velocity and total condensate␣
↪mixing ratio:
display(W_mid_max)
display(TWC)
```

```
<xarray.DataArray 'w' (time: 47, south_north: 198, west_east: 198)>
[1842588 values with dtype=float32]
Coordinates:
  * time                (time) datetime64[ns] 2013-06-19T20:05:00 ... 2013-06-19T23:55:00
  * south_north         (south_north) int64 281 282 283 284 ... 475 476 477 478
  * west_east           (west_east) int64 281 282 283 284 ... 475 476 477 478
    bottom_top_stag     int64 ...
    latitude            (south_north, west_east) float32 ...
    longitude           (south_north, west_east) float32 ...
    model_level_number  int64 ...
    x                   (west_east) float64 ...
    y                   (south_north) float64 ...
    x_0                 (west_east) int64 ...
    y_0                 (south_north) int64 ...
Attributes:
    long_name:     w
    units:         m s-1
    cell_methods:  model_level_number: maximum
```

```
<xarray.DataArray 'twc' (time: 47, bottom_top: 94, south_north: 198, west_east: 198)>
[173203272 values with dtype=float32]
Coordinates:
  * time                (time) datetime64[ns] 2013-06-19T20:05:00 ... 2013-06-19T23:55:00
  * bottom_top          (bottom_top) int64 0 1 2 3 4 5 6 ... 88 89 90 91 92 93
  * south_north         (south_north) int64 281 282 283 284 ... 475 476 477 478
  * west_east           (west_east) int64 281 282 283 284 ... 475 476 477 478
    latitude            (south_north, west_east) float32 ...
    longitude           (south_north, west_east) float32 ...
    model_level_number  (bottom_top) int64 ...
    x                   (west_east) float64 ...
    y                   (south_north) float64 ...
    x_0                 (west_east) int64 ...
    y_0                 (south_north) int64 ...
Attributes:
    long_name:  TWC
    units:      kg kg-1
```

[10]:
```
#Set up directory to save output and plots:
savedir='Save'
if not os.path.exists(savedir):
    os.makedirs(savedir)
plot_dir="Plot"
if not os.path.exists(plot_dir):
    os.makedirs(plot_dir)
```

Feature detection:

Perform feature detection based on midlevel maximum vertical velocity and a range of threshold values.

[11]:
```
# Determine temporal and spatial sampling of the input data:
dxy,dt=tobac.utils.get_spacings(W_mid_max)
```

[12]:
```
# Keyword arguments for feature detection step:
parameters_features={}
parameters_features['position_threshold']='weighted_diff'
parameters_features['sigma_threshold']=0.5
parameters_features['min_num']=3
parameters_features['min_distance']=0
parameters_features['sigma_threshold']=1
parameters_features['threshold']=[3,5,10] #m/s
parameters_features['n_erosion_threshold']=0
parameters_features['n_min_threshold']=3
```

[13]:
```
# Perform feature detection and save results:
print('start feature detection based on midlevel column maximum vertical velocity')
dxy,dt=tobac.utils.get_spacings(W_mid_max)
Features=tobac.themes.tobac_v1.feature_detection_multithreshold(W_mid_max,dxy,
↪**parameters_features)
print('feature detection performed start saving features')
Features.to_netcdf(os.path.join(savedir,'Features.nc'))
print('features saved')
```

```
start feature detection based on midlevel column maximum vertical velocity
     frame  idx      hdim_1      hdim_2  num  threshold_value  feature  \
0        0    3   86.765659   53.350266   10                3        1
1        0    6  109.581883   65.230958    7                3        2
2        0    8  116.808699  191.185786    6                3        3
3        0   12  155.613171   33.200565    5                3        4
4        0   14   84.857363   37.092481   43                5        5
..     ...  ...         ...         ...  ...              ...      ...
624     46   36  144.848990  116.270462   51               10      625
625     46   38  158.442623  106.995282    6               10      626
626     46   39  172.189120    3.980561   53               10      627
627     46   40  186.763110  101.952079   17               10      628
628     46   41  190.393192  179.765727    4               10      629

                    time             timestr  south_north   west_east  \
0    2013-06-19 20:05:00  2013-06-19 20:05:00   367.765659  334.350266
1    2013-06-19 20:05:00  2013-06-19 20:05:00   390.581883  346.230958
2    2013-06-19 20:05:00  2013-06-19 20:05:00   397.808699  472.185786
3    2013-06-19 20:05:00  2013-06-19 20:05:00   436.613171  314.200565
4    2013-06-19 20:05:00  2013-06-19 20:05:00   365.857363  318.092481
..                   ...                  ...          ...         ...
624  2013-06-19 23:55:00  2013-06-19 23:55:00   425.848990  397.270462
625  2013-06-19 23:55:00  2013-06-19 23:55:00   439.442623  387.995282
626  2013-06-19 23:55:00  2013-06-19 23:55:00   453.189120  284.980561
627  2013-06-19 23:55:00  2013-06-19 23:55:00   467.763110  382.952079
628  2013-06-19 23:55:00  2013-06-19 23:55:00   471.393192  460.765727

     bottom_top_stag  model_level_number  projection_y_coordinate           y  \
0         334.350266          334.350266            184132.829257  367.765659
1         346.230958          346.230958            195540.941390  390.581883
2         472.185786          472.185786            199154.349360  397.808699
3         314.200565          314.200565            218556.585318  436.613171
4         318.092481          318.092481            183178.681637  365.857363
..               ...                 ...                      ...         ...
624       397.270462          397.270462            213174.494771  425.848990
625       387.995282          387.995282            219971.311750  439.442623
626       284.980561          284.980561            226844.559837  453.189120
627       382.952079          382.952079            234131.555024  467.763110
628       460.765727          460.765727            235946.595843  471.393192

                latitude             longitude  projection_x_coordinate  \
0    [30.016038859108175]  [-94.62685053589469]            167425.132894
1    [30.120045475411185]   [-94.5637249988283]            173365.479208
2     [30.14916169399921]  [-93.89838383702785]            236342.893153
3     [30.33071080683231]  [-94.73227239126004]            157350.282423
4    [30.007544572713726]  [-94.71261994128162]            159296.240490
..                    ...                  ...                      ...
624  [30.279932206676968]  [-94.29310184128425]            198885.230853
625  [30.342259128747088]  [-94.34174723855492]            194247.641224
626   [30.40663218356964]  [-94.88678519978228]            142740.280723
627  [30.471714442853838]  [-94.36760066497843]            191726.039457
628  [30.485679966689368]  [-93.95513846172912]            230632.863513
```

(continues on next page)

(continued from previous page)

```
             x
0    334.350266
1    346.230958
2    472.185786
3    314.200565
4    318.092481
..          ...
624  397.270462
625  387.995282
626  284.980561
627  382.952079
628  460.765727

[629 rows x 19 columns]
['time', 'south_north', 'west_east', 'bottom_top_stag', 'latitude', 'longitude', 'model_
↪level_number', 'x', 'y', 'x_0', 'y_0']
feature detection performed start saving features
features saved
```

Segmentation:

Perform segmentation based on 3D total condensate field to determine cloud volumes associated to identified features:

[14]:
```
parameters_segmentation_TWC={}
parameters_segmentation_TWC['method']='watershed'
parameters_segmentation_TWC['threshold']=0.1e-3  # kg/kg mixing ratio
```

[15]:
```
print('Start segmentation based on total water content')
Mask_TWC,Features_TWC=tobac.themes.tobac_v1.segmentation(Features,TWC,dxy,**parameters_
↪segmentation_TWC)
print('segmentation TWC performed, start saving results to files')
Mask_TWC.to_netcdf(os.path.join(savedir,'Mask_Segmentation_TWC.nc'))
Features_TWC.to_netcdf(os.path.join(savedir,'Features_TWC.nc'))
print('segmentation TWC performed and saved')
```

```
Start segmentation based on total water content
<xarray.DataArray 'twc' (time: 47, bottom_top: 94, south_north: 198, west_east: 198)>
[173203272 values with dtype=float32]
Coordinates:
  * time                (time) datetime64[ns] 2013-06-19T20:05:00 ... 2013-06-19T23:55:00
  * bottom_top          (bottom_top) int64 0 1 2 3 4 5 6 ... 88 89 90 91 92 93
  * south_north         (south_north) int64 281 282 283 284 ... 475 476 477 478
  * west_east           (west_east) int64 281 282 283 284 ... 475 476 477 478
    latitude            (south_north, west_east) float32 29.62032 ... 30.515106
    longitude           (south_north, west_east) float32 -94.90857 ... -93.863464
    model_level_number  (bottom_top) int64 0 1 2 3 4 5 6 ... 88 89 90 91 92 93
    x                   (west_east) float64 1.408e+05 1.412e+05 ... 2.392e+05
    y                   (south_north) float64 1.408e+05 1.412e+05 ... 2.392e+05
    x_0                 (west_east) int64 281 282 283 284 ... 475 476 477 478
    y_0                 (south_north) int64 281 282 283 284 ... 475 476 477 478
```

(continues on next page)

(continued from previous page)

```
Attributes:
    long_name:  TWC
    units:      kg kg-1
```

```
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
```

(continues on next page)

(continued from previous page)

```
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
```

(continues on next page)

(continued from previous page)

```
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
```

(continues on next page)

(continued from previous page)

```
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
/home/senf/TROPOS/tmp/tobac-dev-2020-07-09/test-env/lib/python3.7/site-packages/skimage/
↪morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated␣
↪and will be removed in version 0.19. Use ``skimage.segmentation.watershed`` instead.
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
```

```
segmentation TWC performed, start saving results to files
segmentation TWC performed and saved
```

Trajectory linking:

Detected features are linked into cloud trajectories using the trackpy library (http://soft-matter.github.io/trackpy). This takes the feature positions determined in the feature detection step into account but does not include information on the shape of the identified objects.

[16]:
```
# Keyword arguments for linking step:
parameters_linking={}
parameters_linking['method_linking']='predict'
parameters_linking['adaptive_stop']=0.2
```

(continues on next page)

(continued from previous page)

```
parameters_linking['adaptive_step']=0.95
parameters_linking['extrapolate']=0
parameters_linking['order']=1
parameters_linking['subnetwork_size']=100
parameters_linking['memory']=0
parameters_linking['time_cell_min']=5*60
parameters_linking['method_linking']='predict'
parameters_linking['v_max']=10
parameters_linking['d_min']=2000
```

[17]:
```
# Perform linking and save results:
Track=tobac.themes.tobac_v1.linking_trackpy(Features,W_mid_max,dt=dt,dxy=dxy,
↪**parameters_linking)
Track.to_netcdf(os.path.join(savedir,'Track.nc'))
```

```
Frame 46: 18 trajectories present.
```

Visualisation:

[18]:
```
# Set extent for maps plotted in the following cells ( in the form [lon_min,lon_max,lat_
↪min,lat_max])
axis_extent=[-95,-93.8,29.5,30.6]
```

[19]:
```
# Plot map with all individual tracks:
import cartopy.crs as ccrs
fig_map,ax_map=plt.subplots(figsize=(10,10),subplot_kw={'projection': ccrs.PlateCarree()}
↪)
ax_map=tobac.plot.map_tracks(Track,axis_extent=axis_extent,axes=ax_map)
```

[21]:
```
# Create animation showing tracked cells with outline of cloud volumes and the midlevel␣
↪vertical velocity as a background field:
animation_tobac=tobac.plot.animation_mask_field(Track,Features,W_mid_max,Mask_TWC,
                                        axis_extent=axis_extent,#figsize=figsize,
↪orientation_colorbar='horizontal',pad_colorbar=0.2,
                                        vmin=0,vmax=20,extend='both',cmap='Blues',
                                        interval=500,figsize=(10,7),
                                        plot_outline=True,plot_marker=True,marker_
↪track='x',plot_number=True,plot_features=True)
```

[22]:
```
# Display animation:
from IPython.display import HTML, Image, display
HTML(animation_tobac.to_html5_video())
```

[22]:
```
<IPython.core.display.HTML object>
```

[23]:
```
# # Save animation to file
# savefile_animation=os.path.join(plot_dir,'Animation.mp4')
# animation_tobac.save(savefile_animation,dpi=200)
# print(f'animation saved to {savefile_animation}')
```

[24]:
```
# Updraft lifetimes of tracked cells:
fig_lifetime,ax_lifetime=plt.subplots()
tobac.plot.plot_lifetime_histogram_bar(Track,axes=ax_lifetime,bin_edges=np.arange(0,120,
↪10),density=False,width_bar=8)
ax_lifetime.set_xlabel('lifetime (min)')
ax_lifetime.set_ylabel('counts')
```

[24]:
```
Text(0, 0.5, 'counts')
```

CHAPTER

THIRTEEN

TOBAC EXAMPLE: CYCLONE TRACKING BASED ON RELATIVE VORTICITY IN KILOMETER-SCALE SIMULATIONS

This example notebook demonstrates the use of tobac to track meso-scale vortices, based on the relative vorticity field in kilometer-scale simulations. Since such simulations are characterized by high frequencies in the vorticity field (especially in regions with complex terrain), *tobac* allows you to spectrally filter the input data by applying a bandpass filter based on user-specified wavelengths. This results in the removal of submeso-scale noise. For more details about the used filter method and the **discrete cosine transformation** that is used to transfer input data to the spectral space is given in Denis et al. 2002.

Data description

The data used in this example is relative vorticity from a 4km WRF simulation in the Tibetan Plateau-Himalaya region. This data is part of the CORDEX Flagship Pilot Study CPTP ("Convection-Permitting Third Pole"). The target weather system, which we want to track here are shallow meso-scale vortices at 500 hPa. The WRF simulation that produced the input data used Thompson microphysics, the Yonsei University (YSU) planetary boundary layer scheme, the RRTMG longwave and shortwave radiation scheme, and the Noah-MP land surface scheme.

Other applications for the spectral filtering tool in *tobac* could be to detect: - MJO - equatorial waves - atmospheric rivers - ...

You can access the data from the zenodo link, which is provided in the notebook.

[1]:
```
# Import a range of python libraries used in this notebook:
import xarray as xr
import numpy as np
import pandas as pd
from pathlib import Path
import urllib

# Import tobac itself:
import tobac
```

[2]:
```
%matplotlib inline

import matplotlib.pyplot as plt
```

[3]:
```
# Disable a few warnings:
import warnings
warnings.filterwarnings('ignore', category=UserWarning, append=True)
warnings.filterwarnings('ignore', category=RuntimeWarning, append=True)
warnings.filterwarnings('ignore', category=FutureWarning, append=True)
warnings.filterwarnings('ignore',category=pd.io.pytables.PerformanceWarning)
```

Download example data:

[4]:
```
# change this location if you want to save the downloaded data elsewhere
data_out= Path('../data')
if data_out.exists() is False:
    data_out.mkdir()
```

[5]:
```
# Download the data: This only has to be done once for all tobac examples and can take a␣
↪while
data_file = list(data_out.rglob('Example_input_vorticity_model.nc'))
if len(data_file) == 0:
    file_path='https://zenodo.org/record/6459542/files/Example_input_vorticity_model.nc'
    print('start downloading data')
    request=urllib.request.urlretrieve(file_path, data_out / ('Example_input_vorticity_
↪model.nc') )
    data_file = list(data_out.rglob('Example_input_vorticity_model.nc'))
```

```
start downloading data
```

[7]:
```
# Load Data from downloaded file:
data_file = Path(data_out/ 'Example_input_vorticity_model.nc')
ds = xr.open_dataset(data_file)
# get variables
relative_vorticity = ds.rv500
lats = ds.latitude
lons = ds.longitude

# check out what the data looks like
ds
```

[7]:
```
<xarray.Dataset>
Dimensions:      (time: 168, south_north: 469, west_east: 866)
Coordinates:
  * time         (time) datetime64[ns] 2008-07-14 ... 2008-07-20T23:00:00
  * south_north  (south_north) int64 33 34 35 36 37 38 ... 497 498 499 500 501
  * west_east    (west_east) int64 741 742 743 744 745 ... 1603 1604 1605 1606
    latitude     (south_north, west_east) float32 ...
    longitude    (south_north, west_east) float32 ...
Data variables:
    rv500        (time, south_north, west_east) float64 ...
Attributes:
    simulation:  WRF 4km
    forcing:     ECMWF-ERA5
    institute:   National Center for Atmospheric Research, Boulder
    contact:     julia.kukulies@gu.se
```

[8]:
```
#Set up directory to save output and plots:
savedir = Path('Save')
plotdir= Path('Plots')

savedir.mkdir(parents=True, exist_ok=True)
plotdir.mkdir(parents=True, exist_ok=True)
```

13.1 Check how the spectrally input field would look like

If you want to check how the filter and your filtered data looked like, you can do that by using the method *tobac.utils.general.spectral_filtering()* directly on your input data. This can be helpful in the development process, if you want to try out different ranges of wavelengths and see how this changes your data. In the example, we use use the spectral filtering method to remove wavelengths < 400 km and > 1000 km, because the focus is on meso-scale vortices.

13.1.1 Create your filter

[9]:
```
help(tobac.utils.general.spectral_filtering)
```

```
Help on function spectral_filtering in module tobac.utils.general:

spectral_filtering(dxy, field_in, lambda_min, lambda_max, return_transfer_function=False)
    This function creates and applies a 2D transfer function that can be used as a␣
↪bandpass filter to remove
    certain wavelengths of an atmospheric field (e.g. vorticity).

    Parameters:
    -----------

    dxy : float
        grid spacing in m


    field_in: numpy.array
        2D field with input data

    lambda_min: float
        minimum wavelength in km


    lambda_max: float
        maximum wavelength in km

    return_transfer_function: boolean, optional
        default: False. If set to True, then the 1D transfer function are returned.

    Returns:
    --------

    filtered_field: numpy.array
        spectrally filtered 2D field of data

    transfer_function: tuple
        Two 2D fields, where the first one corresponds to the wavelengths of the domain␣
↪and the second one
        to the 2D transfer function of the bandpass filter. Only returned, if return_
↪transfer_function is True.
```

[10]:
```
# define minimum and maximum wavelength
lambda_min, lambda_max = 400, 1000
dxy = 4000

# use spectral filtering method on input data to check how the applied filter changes␣
↪the data
transfer_function, relative_vorticity_meso = tobac.utils.general.spectral_filtering(dxy,␣
↪relative_vorticity, lambda_min, lambda_max, return_transfer_function = True)
```

13.1.2 Example for unfiltered vs. filtered input data

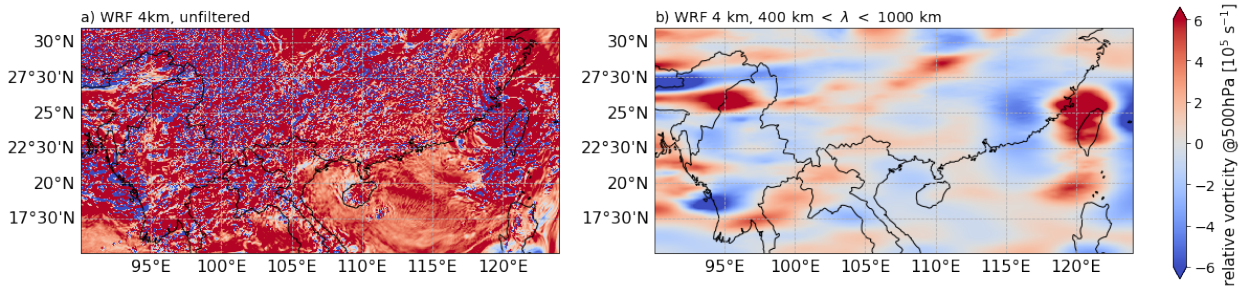
The example shows typhoon *Kalmaegi* over Taiwan on July 18^{th}, 2008 ; as can be seen the corresponding meso-scale vortex becomes only visible in the spectrally filtered field.

[18]:
```
import cartopy.crs as ccrs
import cartopy.feature as cfeature

fig = plt.figure(figsize= (18,12))

fs = 14
axes =  dict()
subplots = 1
ax1 = plt.subplot2grid(shape=(subplots, 2), loc=(0, 0), colspan=1, projection=ccrs.
↪PlateCarree())
ax2 = plt.subplot2grid(shape=(subplots, 2), loc=(0, 1), colspan=1, projection=ccrs.
↪PlateCarree())
########################################## vortex ############################

cmap = 'coolwarm'
#time step to plot
tt = 101
col = 'black'
extent = [90, 124, 15, 30]

ax1.set_extent(extent)
ax1.set_title('a) WRF 4km, unfiltered', fontsize= fs , loc = 'left')
ax1.pcolormesh(lons, lats, relative_vorticity[tt], cmap= cmap , vmin = -6, vmax = 6)
ax1.add_feature(cfeature.BORDERS, color = 'black')
ax1.add_feature(cfeature.COASTLINE, color = col)
gl= ax1.gridlines(draw_labels=True, dms=True, x_inline=False, y_inline=False, linestyle=
↪'--')
gl.top_labels = False
gl.right_labels = False
gl.xlabel_style = {'size': 16, 'color': 'black'}
gl.ylabel_style = {'size': 16, 'color': 'black'}

ax2.set_extent(extent)
ax2.set_title('b) WRF 4 km, 400 km $<$ $\lambda$ $<$ 1000 km', fontsize= fs , loc = 'left
↪')
vort  = ax2.pcolormesh(lons, lats, relative_vorticity_meso[tt] , cmap= cmap , vmin =-6,␣
↪vmax = 6)
```

(continues on next page)

(continued from previous page)

```
ax2.add_feature(cfeature.COASTLINE, color = col)
ax2.add_feature(cfeature.BORDERS, color = 'black')
gl= ax2.gridlines(draw_labels=True, dms=True, x_inline=False, y_inline=False, linestyle=
↪'--')
gl.top_labels = False
gl.right_labels = False
gl.xlabel_style = {'size': 16, 'color': 'black'}
gl.ylabel_style = {'size': 16, 'color': 'black'}

### colorbar ###
cb_ax2 = fig.add_axes([0.93, 0.35,0.01, 0.3])
cbar = fig.colorbar(vort, cax=cb_ax2, extend = 'both')
cbar.ax.tick_params(labelsize=fs)
cbar.set_label(r'relative vorticity @500hPa [10$^5$ s$^{-1}$]', size=15)
plt.rcParams.update({'font.size': fs} )
plt.show()
```

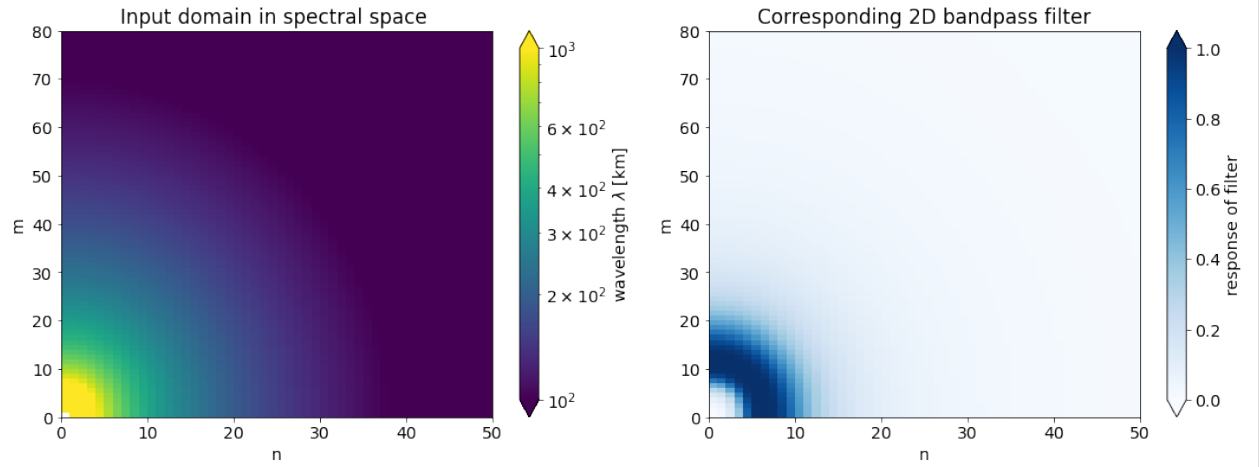


[11]:
```
# checkout how the 2D filter looks like
import matplotlib.colors as colors
plt.figure(figsize= (18,6))

ax = plt.subplot(1,2,1)
# 2D field in spectral space
k = ax.pcolormesh(transfer_function[0],norm = colors.LogNorm(1e2, 1e3 ), shading = 'auto
↪')
plt.colorbar(k, label = 'wavelength $\lambda$ [km]', extend = 'both')
ax.set_ylabel('m')
ax.set_xlabel('n')
# zoom in to see relevant wavelengths
ax.set_xlim([0,50])
ax.set_ylim([0,80])
ax.set_title('Input domain in spectral space ')

ax = plt.subplot(1,2,2)
# 2D filter
tf = ax.pcolormesh(transfer_function[1] /np.nanmax(transfer_function[1]), vmin = 0, vmax␣
↪=1, cmap = 'Blues')
plt.colorbar(tf, label = 'response of filter', extend = 'both')
ax.set_title('Corresponding 2D bandpass filter')
ax.set_ylabel('m')
ax.set_xlabel('n')
ax.set_xlim([0,50])
```

(continues on next page)

(continued from previous page)

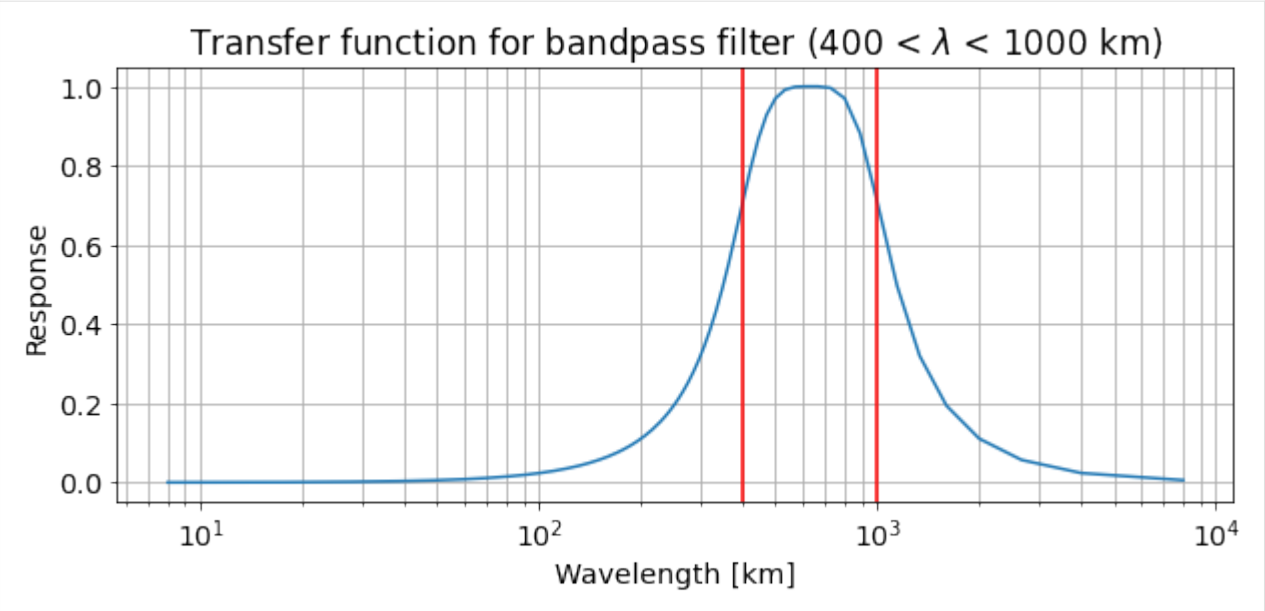
```
ax.set_ylim([0,80])

plt.show()
```



The response of the filter is 1 at locations, where wavelengths are within acceptable range and 0, when the wavelengths are outside of this range (here for: 400 km < λ < 1000 km). The transition is smoothed. To better understand this transition, one could also look at the same filter in one dimension (with the red lines indicating the wavelength truncations):

13.1.3 The same filter in 1D:

[12]:
```
from scipy import signal
# calculate filter according to lambda_min and lambda_max
dxy = 4
b, a = signal.iirfilter(2, [1/lambda_max, 1/lambda_min], btype='band', ftype='butter',␣
↪fs= 1/dxy, output ='ba')
w, h = signal.freqz(b, a, 1000, fs = 1/dxy)

fig = plt.figure(figsize=(10,4))
ax = fig.add_subplot(1, 1, 1)
plt.semilogx(1/w, abs(h))
#plt.plot(w, h)
ax.set_title('Transfer function for bandpass filter (400 < $\lambda$ < 1000 km)')
ax.set_xlabel('Wavelength [km]')
ax.set_ylabel('Response')
ax.axvline(400, c= 'r')
ax.axvline(1000, c= 'r')
ax.grid(which='both', axis='both')
plt.show()
```



If you are happy with how the filter changes your input field, continue as usual with the feature detection and segmentation:

13.2 Feature detection:

Feature detection is then performed based on the filtered relative vorticity field and your chosen set of thresholds.

[13]:
```
# you can use this function to get the grid spacings from lats and lons
dxy, dt = tobac.utils.general.get_spacings(relative_vorticity)
# but better to define exact grid spacing if known, since get_spacings() uses haversine␣
↪approximation
dxy = 4000
```

[14]:
```
# if you want your WRF data to indcate the track locations in lons and lats:
relative_vorticity
```

[14]:
```
<xarray.DataArray 'rv500' (time: 168, south_north: 469, west_east: 866)>
array([[[  3.138932,   3.150467, ...,   5.20407 ,   5.191312],
        [  3.202842,   3.218159, ...,   5.133   ,   5.111225],
        ...,
        [ 11.466752,  11.446242, ...,   2.244073,   2.288456],
        [  6.06062 ,   6.087327, ...,   2.238939,   2.280738]],

       [[  3.063716,   3.038443, ...,   4.815409,   5.123763],
        [  3.141597,   3.124234, ...,   4.726799,   5.030745],
        ...,
        [ 12.680849,  12.979313, ...,   2.141634,   2.294254],
        [ -1.421874,  -1.235242, ...,   2.125223,   2.277909]],

       ...,

       [[ 10.169939,   9.744318, ...,   3.209985,   3.176951],
```

(continues on next page)

(continued from previous page)

```
        [ 10.194508,   9.936515, ...,   3.136149,   3.103187],
        ...,
        [  3.718061,  -0.572581, ..., -28.510893,  -8.78719 ],
        [ 14.069323,  15.725659, ..., -28.109968,  -8.83858 ]],

       [[  9.703144,   8.762362, ...,   2.785694,   2.797884],
        [  9.489581,   8.667569, ...,   2.672183,   2.686641],
        ...,
        [  9.156374,   7.913566, ..., -32.878235, -10.757242],
        [ 20.767054,  15.039678, ..., -33.59285 , -11.135064]]])
Coordinates:
  * time         (time) datetime64[ns] 2008-07-14 ... 2008-07-20T23:00:00
  * south_north  (south_north) int64 33 34 35 36 37 38 ... 497 498 499 500 501
  * west_east    (west_east) int64 741 742 743 744 745 ... 1603 1604 1605 1606
    latitude     (south_north, west_east) float32 15.03 15.03 ... 31.98 31.98
    longitude    (south_north, west_east) float32 90.04 90.08 ... 124.3 124.4
Attributes:
    long name:      Relative vorticity 500 hPA
    standard name:  relative_vorticity
    units:          10^-5 s-1
```

[15]:
```
# Dictionary containing keyword arguments for feature detection step (Keywords could␣
↪also be given directly in the function call).
parameters_features={}
parameters_features['position_threshold']='weighted_diff'
parameters_features['sigma_threshold']=0.5
parameters_features['min_num']= 5
parameters_features['n_min_threshold']= 20
parameters_features['target']='maximum'
parameters_features['threshold']=[3, 5, 8 , 10 ]
```

[16]:
```
# Perform feature detection:
print('starting feature detection')
Features=tobac.themes.tobac_v1.feature_detection_multithreshold(relative_vorticity, dxy␣
↪= 4000 ,**parameters_features, wavelength_filtering = (400,1000))
Features.to_netcdf(savedir / 'Features.nc')
print('feature detection performed and saved')
```

```
starting feature detection
      frame  idx      hdim_1      hdim_2   num  threshold_value  feature  \
0         0    2   79.219141  263.069935   443                3        1
1         0    4  152.995004  399.426263  2753                3        2
2         0    5  172.575146  216.544321   474                3        3
3         0    7  251.673434  571.450241  1594                3        4
4         0    8  274.170082  410.917904   701                3        5
...     ...  ...         ...         ...   ...              ...      ...
2349    167   19  337.959750  752.236529    47                5     2350
2350    167   22  461.052734  645.960792  1127                5     2351
2351    167   25  391.695006    7.594277    52                8     2352
2352    167   28  356.405490  339.875838  1230               10     2353
2353    167   29  427.941365  399.063901   559               10     2354
```

(continues on next page)

(continued from previous page)

```
                    time              timestr  south_north    west_east  \
0     2008-07-14 00:00:00  2008-07-14 00:00:00   112.219141  1004.069935
1     2008-07-14 00:00:00  2008-07-14 00:00:00   185.995004  1140.426263
2     2008-07-14 00:00:00  2008-07-14 00:00:00   205.575146   957.544321
3     2008-07-14 00:00:00  2008-07-14 00:00:00   284.673434  1312.450241
4     2008-07-14 00:00:00  2008-07-14 00:00:00   307.170082  1151.917904
...                   ...                  ...          ...          ...
2349  2008-07-20 23:00:00  2008-07-20 23:00:00   370.959750  1493.236529
2350  2008-07-20 23:00:00  2008-07-20 23:00:00   494.052734  1386.960792
2351  2008-07-20 23:00:00  2008-07-20 23:00:00   424.695006   748.594277
2352  2008-07-20 23:00:00  2008-07-20 23:00:00   389.405490  1080.875838
2353  2008-07-20 23:00:00  2008-07-20 23:00:00   460.941365  1140.063901

                  latitude             longitude
0      [18.04406975782231]  [100.48205467762966]
1     [20.805768533908584]  [105.89511314555088]
2       [21.5306068809038]   [98.63508683939142]
3       [24.4211252451139]  [112.72410280495953]
4     [25.231653106827366]  [106.35130899162766]
...                    ...                  ...
2349  [27.500329035090285]  [119.90093733498449]
2350  [31.746546530874113]  [115.68201445287401]
2351  [29.375957848118027]    [90.3402070608934]
2352   [28.14790948911609]  [103.53108203810258]
2353  [30.622050722798424]  [105.88072618952833]

[2354 rows x 13 columns]
['time', 'south_north', 'west_east', 'latitude', 'longitude']
feature detection performed and saved
```

13.3 Segmentation

Segmentation is performed with watershedding based on the detected features and a single threshold value.

[17]:
```
# Dictionary containing keyword options for the segmentation step:
parameters_segmentation={}
parameters_segmentation['target']='maximum'
parameters_segmentation['method']='watershed'
parameters_segmentation['threshold']= 1.5
```

[18]:
```
# Perform segmentation and save results:
print('Starting segmentation based on relative vorticity.')
Mask_rv,Features_rv=tobac.themes.tobac_v1.segmentation(Features,relative_vorticity,dxy,
↪**parameters_segmentation)
print('segmentation performed, start saving results to files')
Mask_rv.to_netcdf(savedir / 'Mask_Segmentation_rv.nc')
Features_rv.to_netcdf(savedir/ 'Features_rv.nc')
print('segmentation performed and saved')
```

```
Starting segmentation based on relative vorticity.
<xarray.DataArray 'rv500' (time: 168, south_north: 469, west_east: 866)>
array([[[  3.138932,   3.150467, ...,   5.20407 ,   5.191312],
        [  3.202842,   3.218159, ...,   5.133   ,   5.111225],
        ...,
        [ 11.466752,  11.446242, ...,   2.244073,   2.288456],
        [  6.06062 ,   6.087327, ...,   2.238939,   2.280738]],

       [[  3.063716,   3.038443, ...,   4.815409,   5.123763],
        [  3.141597,   3.124234, ...,   4.726799,   5.030745],
        ...,
        [ 12.680849,  12.979313, ...,   2.141634,   2.294254],
        [ -1.421874,  -1.235242, ...,   2.125223,   2.277909]],

       ...,

       [[ 10.169939,   9.744318, ...,   3.209985,   3.176951],
        [ 10.194508,   9.936515, ...,   3.136149,   3.103187],
        ...,
        [  3.718061,  -0.572581, ..., -28.510893,  -8.78719 ],
        [ 14.069323,  15.725659, ..., -28.109968,  -8.83858 ]],

       [[  9.703144,   8.762362, ...,   2.785694,   2.797884],
        [  9.489581,   8.667569, ...,   2.672183,   2.686641],
        ...,
        [  9.156374,   7.913566, ..., -32.878235, -10.757242],
        [ 20.767054,  15.039678, ..., -33.59285 , -11.135064]]])
Coordinates:
  * time         (time) datetime64[ns] 2008-07-14 ... 2008-07-20T23:00:00
  * south_north  (south_north) int64 33 34 35 36 37 38 ... 497 498 499 500 501
  * west_east    (west_east) int64 741 742 743 744 745 ... 1603 1604 1605 1606
    latitude     (south_north, west_east) float32 15.03 15.03 ... 31.98 31.98
    longitude    (south_north, west_east) float32 90.04 90.08 ... 124.3 124.4
Attributes:
    long name:      Relative vorticity 500 hPA
    standard name:  relative_vorticity
    units:          10^-5 s-1
segmentation performed, start saving results to files
segmentation performed and saved
```

13.4 Trajectory linking

Features are linked into trajectories using the trackpy library (http://soft-matter.github.io/trackpy). This takes the feature positions determined in the feature detection step into account but does not include information on the shape of the identified objects.**

[19]:
```
# Arguments for trajectory linking:
parameters_linking={}
parameters_linking['v_max']=80
parameters_linking['stubs']=2
parameters_linking['order']=1
```

(continues on next page)

(continued from previous page)

```
parameters_linking['extrapolate']=1
parameters_linking['memory']=0
parameters_linking['adaptive_stop']=0.2
parameters_linking['adaptive_step']=0.95
parameters_linking['subnetwork_size']=1000
# require that the vortex has to persist during at least 12 hours
parameters_linking['time_cell_min']= 12*dt
parameters_linking['method_linking']= 'predict'
```

[20]:
```
# Perform linking and save results to file:
Track=tobac.themes.tobac_v1.linking_trackpy(Features, relative_vorticity ,dt=dt,dxy=dxy ,
↪**parameters_linking)
Track.to_netcdf(savedir/ 'Track.nc')
```

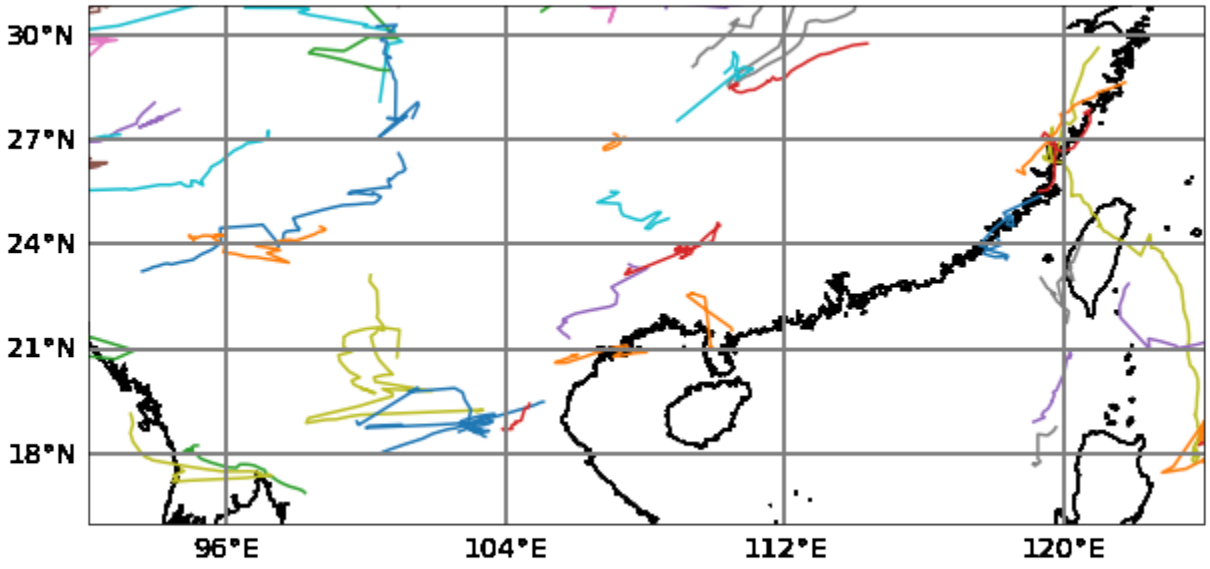
```
Frame 167: 14 trajectories present.
```

13.5 Visualisation of tracks

The long track in the east is the mesoscale vortex associated with **typhoon Kalmeagi** that passed Taiwan in July 2008.

The tracks in the west correspond to vortices that frequently form in the higher mountains over the Tibetan Plateau.

[21]:
```
# Plot map with all individual tracks:
import cartopy.crs as ccrs
fig_map,ax_map=plt.subplots(figsize=(10,10),subplot_kw={'projection': ccrs.PlateCarree()}
↪)
ax_map= tobac.plot.map_tracks(Track,axis_extent= extent,axes=ax_map)
```



CHAPTER

FOURTEEN

INTRO

This is a set of tutorial notebooks which provide example usage of the python package *tobac* - Tracking and Object-Based Analysis of Clouds.

The python package *tobac* is an open software for atmoshperic scientist aiming to support Lagrangian analysis of cloud fields. The methods can be applied either to observational fields or to simulated cloud variables.

CHAPTER

FIFTEEN

THEMES

The *tobac* package is structured in two layers: In the first layer, core functionality is provided. The second layer is called “themes” and provides a dedicated set of functions to track or analyse clouds. We keep separate developments in separate “theme” like plugins in your browser. Parts of the different themes might be exchangeable.

CHAPTER

SIXTEEN

EXAMPLES

Examples or tutorial are collected to show how *tobac* and *tobac* "themes" are working. Function have been written to ease input of certain obervational or model-simulated fields available at TROPOS. These function included data from

- **tobac_v1**
- *Tracking of Observed OLR*
- *Tracking of Observed Visible Satellite Images*
- *Tracking of Simulated OLR*
- *Tracking of Simulated Precipitation Cells*
- *Tracking of Simulated Updrafts Cells*
- *Tracking of Simulated Vorticity Anomalies*
- **your theme**
- We search for volunteers that included their cloud tracking or analysis approach into *tobac* as a new theme and provide nice tutorials on the cool new functions here.