tobac

Release v2.0-dev

May 19, 2020

Contents

tobac is a Python package to identify, track and analyse clouds in different types of gridded datasets, such as 3D model output from cloud resolving model simulations or 2D data from satellite retrievals.

The software is set up in a modular way to include different algorithms for feature identification, tracking and analyses. In the current implementation, individual features are indentified as either maxima or minima in a two dimensional time varying field. The volume/are associated with the identified object can be determined based on a time-varying 2D or 3D field and a threshold value. In the tracking step, the identified objects are linked into consistent trajectories representing the cloud over its lifecycle. Analysis and visualisation methods provide a convenient way to use and display the tracking results.

Version 1.2 of tobac and some example applications are described in a paper in the journal Geoscientific Model Development as:

Heikenfeld, M., Marinescu, P. J., Christensen, M., Watson-Parris, D., Senf, F., van den Heever, S. C., and Stier, P.: tobac v1.2: towards a flexible framework for tracking and analysis of clouds in diverse datasets, Geosci. Model Dev., https://doi.org/10.5194/gmd-12-4551-2019 , 2019.

The project is currently extended by several contributors to include additional workflows and algorithms using the same structure, synthax and data formats.

CHAPTER 1

Installation

tobac is now capable of working with both Python 2 and Python 3 (tested for Python 2.7, 3.6, 3.7 and 3.8) installations.

The easiest way is to install the most recent version of tobac via conda and the conda-forge channel:

```
` conda install -c conda-forge tobac `
```

This will take care of all necessary dependencies and should do the job for most users and also allows for an easy update of the installation by

```
` conda update -c conda-forge tobac `
```

You can also install conda via pip, which is mainly interesting for development purposed or to use specific development branches for the Github repository.

The follwoing python packages are required (including dependencies of these packages):

trackpy, *scipy*, *numpy*, *iris*, *scikit-learn*, *scikit-image*, *cartopy*, *pandas*, *pytables*

If you are using anaconda, the following command should make sure all dependencies are met and up to date:

```
conda install -c conda-forge -y trackpy scipy numpy iris scikit-learn
scikit-image cartopy pandas pytables
```

You can directly install the package directly from github with pip and either of the two following commands:

```
pip install --upgrade git+ssh://git@github.com/climate-processes/
tobac.git
```

```
pip install --upgrade git+https://github.com/climate-processes/tobac.
git
```

You can also clone the package with any of the two following commands:

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

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

and install the package from the locally cloned version (The trailing slash is actually necessary):

```
pip install --upgrade tobac/
```

CHAPTER 2

Data input and output

The output of the different analysis steps in tobac are output as either pandas DataFrames in the case of one-dimensional data, such a lists of identified features or cloud trajectoies or as Iris cubes in the case of 2D/3D/4D fields such as cloud masks.

Interoperability with xarray is provided by the convenient functions allowing for a transformation between the two data types. xarray DataArays can be easily converted into iris cubes using xarray's to__iris() method, while the Iris cubes produced as output of tobac can be turned into xarray DataArrays using the from__iris() method.

As part of the development of tobac 2.x, we're currently moving the basic data structures from Iris cubes to xarray DataArrays for improved computing performance and interoperability with other open-source sorftware packages.

CHAPTER 3

tobac themes

Starting from version 2.0 tobac includes several so called themes that group together specific workflows or approaches. One of the themes (tobac_v1) includes the routines from tobac 1.x that are also decribed in the ACP paper.

Currently included themes:

tobac_v1

CHAPTER 4

Analysis

tobac provides several analysis functions that allow for the calculation of important quantities based on the tracking results. This includes the calculation of important properties of the tracked objects such as cloud lifetimes, cloud areas/volumes, but also allows for a convenient calculation of statistics for arbitratry fields of the same shape as as the input data used for the tracking analysis.

CHAPTER 5

Plotting

tobac provides functions to conveniently visualise the tracking results and analyses.

CHAPTER 6

Example notebooks

tobac is provided with a set of Jupyter notebooks that show examples of the application of tobac for different types of datasets.

The notebooks can be found in a separate repository:

https://github.com/climate-processes/tobac-tutorials

The necessary input data for these examples is avaliable on zenodo: www.zenodo.org/... and can be downloaded automatically by the Jupyter notebooks.

The examples currently include four different applications of tobac: 1. Tracking of scattered convection based on vertical velocity and condensate mixing ratio for 3D cloud-resolving model output. 2. Tracking of scattered convection based on surface precipitation from the same cloud-resolving model output 3. Tracking of convective clouds based on outgoing longwave radiation (OLR) for convection-permitting model simulation output 4. Tracking of convective clouds based on OLR in geostationary satellite retrievals.

CHAPTER 7

API reference

7.1 Core Modules

tobac.analysis.analysis	Provide tools to analyse and visualize the tracked objects.
tobac.analysis.centerofgravity	Identify center of gravity and mass for analysis.
tobac.plot.plotting	Provide methods for plotting analyzed data.
tobac.utils	Provide essential methods.

7.1.1 tobac.analysis.analysis

Provide tools to analyse and visualize the tracked objects.

This module provides a set of routines that enables performing analyses and deriving statistics for individual clouds, such as the time series of integrated properties and vertical profiles. It also provides routines to calculate summary statistics of the entire populatin of tracked clouds in the cloud field like histograms of cloud areas/volumes or cloud mass and a detailed cell lifetime analysis. These analysis routines are all built in a modular manner. Thus, users can reuse the most basic methods for interacting with the data structure of the package in their own analysis procedures in Python. This includes functions perfomring simple tasks like looping over all identified objects or cloud trajectories and masking arrays for the analysis of individual cloud objects. Plotting routines include both visualizations for individual convective cells and their properties.

Functions

area_histogram(features, mask[, bin_edges, ...])

Parameters

- **features**

Continued on next page

Table 2 – continued from previous page

calculate_area(features, mask[, method_area])	**Parameters** • **features**
calculate_distance(feature_1, feature_2[, ...])	Computes distance between two features.
calculate_nearestneighbordistance(features)	**Parameters** • **features**
calculate_overlap(track_1, track_2[, ...])	**Parameters** • **track_1, track_2**
calculate_velocity(track[, method_distance])	**Parameters** • **track**
calculate_velocity_individual(feature_old, ...)	**Parameters** • **feature_old**
cell_statistics(input_cubes, track, mask, ...)	**Parameters** • **input_cubes** (*iris.cube.Cube*)
cell_statistics_all(input_cubes, track, ...)	**Parameters** • **input_cubes** (*iris.cube.Cube*)
haversine(lat1, lon1, lat2, lon2)	Computes the Haversine distance in kilometers.
histogram_cellwise(Track[, variable, ...])	**Parameters** • **Track**
histogram_featurewise(Track[, variable, ...])	**Parameters** • **Track**
lifetime_histogram(Track[, bin_edges, ...])	**Parameters** • **Track**
nearestneighbordistance_histogram(features)	**Parameters** • **features**
velocity_histogram(track[, bin_edges, ...])	**Parameters** • **track**

tobac.analysis.analysis.**area_histogram**(*features*, *mask*, *bin_edges=array([0, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000, 10500, 11000, 11500, 12000, 12500, 13000, 13500, 14000, 14500, 15000, 15500, 16000, 16500, 17000, 17500, 18000, 18500, 19000, 19500, 20000, 20500, 21000, 21500, 22000, 22500, 23000, 23500, 24000, 24500, 25000, 25500, 26000, 26500, 27000, 27500, 28000, 28500, 29000, 29500])*, *density=False*, *method_area=None*, *return_values=False*, *representative_area=False*)

Parameters

- **features**
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tracked volumes 0 everywhere else).
- **bin_edges** (*ndarray, optional*) – Default is np.arange(0, 30000, 500).
- **density** (*bool, optional*) – Default is False.
- **representive_area** (*bool, optional*) – Default is False.

Returns

- *hist*
- **bin_edges** (*ndarray*)
- *bin_centers*
- *areas*

Notes

short summary, types and descriptions

tobac.analysis.analysis.**calculate_area**(*features*, *mask*, *method_area=None*)

Parameters

- **features**
- **mask** (*iris.cube.Cube*) – Cube containing mask (int for tracked volumes 0 everywhere else).
- **method_area** (*{None, 'xy', 'latlon'}, optional*) – Default is None.

Returns

- *hist*
- **bin_edges** (*ndarray*)
- *bin_centers*
- *areas*

Raises `ValueError` – If neither latitude/longitude nor projection_x_coordinate/projection_y_coordinate are present in mask_coords.

If latitude/longitude coordinates are 2D.

If latitude/longitude shapes are not supported.

If method is undefined, e.i. method is neither None, 'xy' nor 'latlon'.

Notes

needs short summary, types and descriptions

tobac.analysis.analysis.**calculate_distance**(*feature_1*, *feature_2*, *method_distance=None*)

Computes distance between two features.

It is based on either lat/lon coordinates or x/y coordinates.

Parameters

- **feature_1, feature_2** (*array of latitude, longitute*) – First and second feature or points as array in degrees.
- **method_distance** (*{None, 'xy', 'latlon'}, optional*) – Default is None.

Returns **distance** – Between the two features in meters.

Return type float

Notes

check sense of types and descriptions

tobac.analysis.analysis.**calculate_nearestneighbordistance**(*features*, *method_distance=None*)

Parameters

- **features**
- **method_distance** (*{None, 'xy', 'latlon'}, optional*) – Default is None.

Returns

Return type features

Notes

short summary, types and descriptions

tobac.analysis.analysis.**calculate_overlap**(*track_1*, *track_2*, *min_sum_inv_distance=None*, *min_mean_inv_distance=None*)

Parameters

- **track_1, track_2**
- **min_sum_inv_distance** (*optional*) – Default is None.
- **min_mean_inv_distance** (*optional*) – Default is None.

Returns **overlap**

Return type pandas.DataFrame

Notes

short summary, types and descriptions

tobac.analysis.analysis.**calculate_velocity**(*track*, *method_distance=None*)

Parameters

- **track**
- **method_distance** (*{None, 'xy', 'latlon'}, optional*) – Default is None.

Returns

Return type track

Notes

needs short summary, description and type of track

tobac.analysis.analysis.**calculate_velocity_individual**(*feature_old*, *feature_new*, *method_distance=None*)

Parameters

- **feature_old**
- **feature_new**
- **method_distance** (*{None, 'xy', 'latlon'}, optional*) – Default is None.

Notes

feature_old and feature_new need types and descriptions needs a short summary

tobac.analysis.analysis.**cell_statistics**(*input_cubes*, *track*, *mask*, *aggregators*, *cell*, *output_path='./'*, *output_name='Profiles'*, *width=10000*, *z_coord='model_level_number'*, *dimensions=['x', 'y']*, ***kwargs*)

Parameters

- **input_cubes** (*iris.cube.Cube*)
- **track** (*dask.dataframe.DataFrame*)
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tracked volumes 0 everywhere else).
- **aggregators**
- **cell** (*int*) – Integer id of cell to create masked cube for output.
- **output_path** (*str, optional*) – Default is './'.
- **output_name** (*str, optional*) – Default is 'Profiles'.
- **width** (*int, optional*) – Default is 10000.
- **z_coord** (*str, optional*) – Name of the vertical coordinate in the cube. Default is 'model_level_number'.
- **dimensions** (*list of str, optional*) – Default is ['x', 'y'].
- ****kwargs**

Returns

Return type None

Notes

unsure about anything needs a short summary

tobac.analysis.analysis.**cell_statistics_all**(*input_cubes, track, mask, aggregators, output_path='./', cell_selection=None, output_name='Profiles', width=10000, z_coord='model_level_number', dimensions=['x', 'y'], **kwargs*)

Parameters

- **input_cubes** (*iris.cube.Cube*)
- **track** (*dask.dataframe.DataFrame*)
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tracked volumes 0 everywhere else).
- **aggregators**
- **output_path** (*str, optional*) – Default is './'.
- **cell_selection** (*optional*) – Default is None.
- **output_name** (*str, optional*) – Default is 'Profiles'.
- **width** (*int, optional*) – Default is 10000.
- **z_coord** (*str, optional*) – Name of the vertical coordinate in the cube. Default is 'model_level_number'.
- **dimensions** (*list of str, optional*) – Default is ['x', 'y'].
- ****kwargs**

Returns

Return type None

Notes

unsure about anything needs a short summary

tobac.analysis.analysis.**haversine**(*lat1*, *lon1*, *lat2*, *lon2*)

Computes the Haversine distance in kilometers.

Calculates the Haversine distance between two points (based on implementation CIS https://github.com/cedadev/cis).

Parameters

- **lat1, lon1** (*array of latitude, longitude*) – First point or points as array in degrees.
- **lat2, lon2** (*array of latitude, longitude*) – Second point or points as array in degrees.

Returns **arclen * RADIUS_EARTH** – Distance between the two points in kilometers.

Return type float

Notes

check types

RADIUS_EARTH = 6378.0

tobac.analysis.analysis.**histogram_cellwise**(*Track*, *variable=None*, *bin_edges=None*, *quantity='max'*, *density=False*)

Parameters

- **Track**
- **variable** (*optional*) – Default is None.
- **bin_edges** (*ndarray, optional*) – Default is None.
- **quantity** (*{'max', 'min', 'mean'}, optional*) – Default is 'max'.
- **density** (*bool, optional*) – Default is False.

Returns

- *hist*
- **bin_edges** (*ndarray*)
- *bin_centers*

Raises `ValueError` – If quantity is not 'max', 'min' or 'mean'.

Notes

short summaray, types and descriptions

tobac.analysis.analysis.**histogram_featurewise**(*Track*, *variable=None*, *bin_edges=None*, *density=False*)

Parameters

- **Track**
- **variable** (*optional*) – Default is None.
- **bin_edges** (*ndarray, optional*) – Default is None.
- **density** (*bool, optional*) – Default is False.

Returns

- *hist*
- **bin_edges** (*ndarray*)
- *bin_centers*

Notes

short summaray, types and descriptions

tobac.analysis.analysis.**lifetime_histogram**(*Track*, *bin_edges=array([0, 20, 40, 60, 80, 100, 120, 140, 160, 180])*, *density=False*, *return_values=False*)

Parameters

- **Track**
- **bin_edged** (*ndarray, optional*) – Default is np.arange(0, 200, 20).
- **density** (*bool, optional*) – Default is False.
- **return_values** (*bool, optional*) – Default is False.

Returns

- *hist*
- **bin_edges** (*ndarray*)
- *bin_centers*
- **minutes** (*float*)

Notes

unsure about anything needs short summary

tobac.analysis.analysis.**nearestneighbordistance_histogram**(*features*, *bin_edges=array([0, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000, 10500, 11000, 11500, 12000, 12500, 13000, 13500, 14000, 14500, 15000, 15500, 16000, 16500, 17000, 17500, 18000, 18500, 19000, 19500, 20000, 20500, 21000, 21500, 22000, 22500, 23000, 23500, 24000, 24500, 25000, 25500, 26000, 26500, 27000, 27500, 28000, 28500, 29000, 29500])*, *density=False*, *method_distance=None*, *return_values=False*)

Parameters

- **features**
- **bin_edges** (*ndarray, optional*) – Default is np.arange(0, 30000, 500).
- **density** (*bool, optional*) – Default is False.
- **method_distance** (*{None, 'xy', 'latlon'}, optional*) – Default is None.

Returns

- *hist*
- **bin_edges** (*ndarray*)

- *distances*

Notes

short summary, types and descriptions

tobac.analysis.analysis.**velocity_histogram**(*track*, *bin_edges=array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])*, *density=False*, *method_distance=None*, *return_values=False*)

Parameters

- **track**
- **bin_edges** (*ndarray, optional*) – Default is np.arange(0, 30, 1).
- **density** (*bool, optional*) – Default is False.
- **methods_distance** (*{None, 'xy', 'latlon'}, optional*) – Default is None.
- **return_values** (*bool, optional*) – Default is False.

Returns

- *hist*
- **bin_edges** (*ndarray*)
- *velocities*

Notes

short summary, types and descriptions

7.1.2 tobac.analysis.centerofgravity

Identify center of gravity and mass for analysis.

Functions

calculate_cog(tracks, mass, mask)	Caluclate center of gravity and mass forech tracked cell.
calculate_cog_domain(mass)	Caluclate center of gravity and mass for entire domain.
calculate_cog_untracked(mass, mask)	Caluclate center of gravity and mass for untracked domain parts.
center_of_gravity(cube_in)	Caluclate center of gravity and sum of quantity.
cog_cell(cell[, Tracks, M_total, M_liquid, ...])	**Parameters** • **cell** (*int*) – Integer id of cell to create masked cube for output.

tobac.analysis.centerofgravity.**calculate_cog**(*tracks*, *mass*, *mask*)

Caluclate center of gravity and mass forech tracked cell.

Parameters

- **tracks** (*pandas.DataFrame*) – DataFrame containing trajectories of cell centers.
- **mass** (*iris.cube.Cube*) – Cube of quantity (need coordinates 'time', 'geopotential_height','projection_x_coordinate' and 'projection_y_coordinate').
- **mask** (*iris.cube.Cube*) – Cube containing mask (int > where belonging to cloud volume, 0 everywhere else).

Returns **tracks_out** – Dataframe containing t, x, y, z positions of center of gravity and total cloud mass each tracked cells at each timestep.

Return type pandas.DataFrame

tobac.analysis.centerofgravity.**calculate_cog_domain**(*mass*)

Caluclate center of gravity and mass for entire domain.

Parameters **mass** (*iris.cube.Cube*) – Cube of quantity (need coordinates 'time', 'geopotential_height','projection_x_coordinate' and 'projection_y_coordinate').

Returns **tracks_out** – Dataframe containing t, x, y, z positions of center of gravity and total cloud mass.

Return type pandas.DataFrame

tobac.analysis.centerofgravity.**calculate_cog_untracked**(*mass*, *mask*)

Caluclate center of gravity and mass for untracked domain parts.

Parameters

- **mass** (*iris.cube.Cube*) – Cube of quantity (need coordinates 'time', 'geopotential_height','projection_x_coordinate' and 'projection_y_coordinate').
- **mask** (*iris.cube.Cube*) – Cube containing mask (int > where belonging to cloud volume, 0 everywhere else).

Returns **tracks_out** – Dataframe containing t, x, y, z positions of center of gravity and total cloud mass for untracked part of domain.

Return type pandas.DataFrame

tobac.analysis.centerofgravity.**center_of_gravity**(*cube_in*)

Caluclate center of gravity and sum of quantity.

Parameters **cube_in** (*iris.cube.Cube*) – Cube (potentially masked) of quantity (need coordinates 'geopotential_height','projection_x_coordinate' and 'projection_y_coordinate').

Returns

- **x** (*float*) – X position of center of gravity.
- **y** (*float*) – Y position of center of gravity.
- **z** (*float*) – Z position of center of gravity.
- **variable_sum** (*float*) – Sum of quantity of over unmasked part of the cube.

tobac.analysis.centerofgravity.**cog_cell**(*cell*, *Tracks=None*, *M_total=None*, *M_liquid=None*, *M_frozen=None*, *Mask=None*, *savedir=None*)

Parameters

- **cell** (*int*) – Integer id of cell to create masked cube for output.
- **Tracks** (*optional*) – Default is None.

- **M_total** (*subset of cube, optional*) – Default is None.
- **M_liquid** (*subset of cube, optional*) – Default is None.
- **M_frozen** (*subset of cube, optional*) – Default is None.
- **savedir** (*str*) – Default is None.

Returns

Return type None

Notes

unsure about anything needs a short summary

7.1.3 tobac.plot.plotting

Provide methods for plotting analyzed data.

Plotting routines include both visualizations of the entire clod field and detailed visualizations for individual convective cells and their properties.

Notes

many short summaries are the same

References

Functions

animation_mask_field(track, features, field, ...)	**Parameters** • **track**
make_map(axes)	**Parameters** **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot*)
map_tracks(track[, axis_extent, figsize, axes])	**Parameters** • **track**
plot_histogram_cellwise(track, bin_edges, ...)	**Parameters** • **track**
plot_histogram_featurewise(Track, bin_edges, ...)	**Parameters** • **Track**

Continued on next page

Table 4 – continued from previous page

plot_lifetime_histogram(track[, axes, ...])	**Parameters** • **track**
plot_lifetime_histogram_bar(track[, axes, ...])	**Parameters** • **track**
plot_mask_cell_individual_3Dstatic(cell_i, ...)	Make plots for all cells with fixed frame.
plot_mask_cell_individual_follow(cell_i, ...)	Make individual plot for cell centered around cell.
plot_mask_cell_individual_static(cell_i, ...)	Make plots for cell in fixed frame
plot_mask_cell_track_2D3Dstatic(cell, track, ...)	Make plots for all cells with fixed frame.
plot_mask_cell_track_3Dstatic(cell, track, ...)	Make plots for all cells with fixed frame.
plot_mask_cell_track_follow(cell, track, ...)	Make plots for all cells centered around cell.
plot_mask_cell_track_static(cell, track, ...)	Make plots for all cells with fixed frame.
plot_mask_cell_track_static_timeseries(cell, ...)	Make plots for all cells with fixed frame.
plot_tracks_mask_field(track, field, mask, ...)	**Parameters** • **track**
plot_tracks_mask_field_loop(track, field, ...)	**Parameters** • **track**

tobac.plot.plotting.**animation_mask_field**(*track*, *features*, *field*, *mask*, *interval=500*, *figsize=(10, 10)*, ***kwargs*)

Parameters

- **track**
- **field** (*iris.cube.Cube*)
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **interval** (*int, optional*) – Default is 500.
- **figsize** (*tupel, optional*) – Default is (10, 10).
- ****kwargs**

Returns

Return type none

Notes

needs more descriptions and a short summary line

tobac.plot.plotting.**make_map**(*axes*)

Parameters **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot*)

Returns **axes**

Return type cartopy.mpl.geoaxes.GeoAxesSubplot

Notes

needs more descriptions and short summary

tobac.plot.plotting.**map_tracks**(*track*, *axis_extent=None*, *figsize=(10, 10)*, *axes=None*)

Parameters

- **track**
- **axis_extent** (*matplotlib.axes, optional*) – Default is None.
- **figsize** (*tupel, optional*) – Default is (10, 10).
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.

Returns **axes**

Return type cartopy.mpl.geoaxes.GeoAxesSubplot

Notes

needs short summary needs more descriptions unsure about anything

tobac.plot.plotting.**plot_histogram_cellwise**(*track*, *bin_edges*, *variable*, *quantity*, *axes=None*, *density=False*, ***kwargs*)

Parameters

- **track**
- **bin_edges** (*numpy.arange, optional*) – Default is np.arange(0, 200, 20).
- **variable**
- **quantity**
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.
- **density** (*bool, optional*) – Defualt is False.
- ****kwargs**

Returns

Return type plot_hist

Notes

needs short summary and descriptions

tobac.plot.plotting.**plot_histogram_featurewise**(*Track*, *bin_edges*, *variable*, *axes=None*, *density=False*, ***kwargs*)

Parameters

- **Track**
- **bin_edges** (*numpy.arange, optional*) – Default is np.arange(0, 200, 20).
- **variable**
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.
- **density** (*bool, optional*) – Defualt is False.
- ****kwargs**

Returns

Return type plot_hist

Notes

needs short summary and descriptions

tobac.plot.plotting.**plot_lifetime_histogram**(*track*, *axes=None*, *bin_edges=array([0, 20, 40, 60, 80, 100, 120, 140, 160, 180])*, *density=False*, ***kwargs*)

Parameters

- **track**
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.
- **bin_edges** (*numpy.arange, optional*) – Default is np.arange(0, 200, 20).
- **density** (*bool, optional*) – Defualt is False.
- ****kwargs**

Returns

Return type plot_hist

Notes

needs short summary and descriptions

tobac.plot.plotting.**plot_lifetime_histogram_bar**(*track*, *axes=None*, *bin_edges=array([0, 20, 40, 60, 80, 100, 120, 140, 160, 180])*, *density=False*, *width_bar=1*, *shift=0.5*, ***kwargs*)

Parameters

- **track**
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.
- **bin_edges** (*numpy.arange, optional*) – Default is np.arange(0, 200, 20).

- **density** (*bool, optional*) – Defualt is False.
- **width_bar** (*int, optional*) – Default is 1.
- **shift** (*float, optional*) – Default is 0.5.
- ****kwargs**

Returns

Return type plot_hist

Notes

needs short summary and descriptions

tobac.plot.plotting.**plot_mask_cell_individual_3Dstatic**(*cell_i*, *track*, *cog*, *features*, *mask_total*, *field_contour*, *field_filled*, *axes=None*, *xlim=None*, *ylim=None*, *label_field_contour=None*, *cmap_field_contour='Blues'*, *norm_field_contour=None*, *linewidths_contour=0.8*, *contour_labels=False*, *vmin_field_contour=0*, *vmax_field_contour=50*, *levels_field_contour=None*, *nlevels_field_contour=10*, *label_field_filled=None*, *cmap_field_filled='summer'*, *norm_field_filled=None*, *vmin_field_filled=0*, *vmax_field_filled=100*, *levels_field_filled=None*, *nlevels_field_filled=10*, *title=None*, *feature_number=False*, *ele=10.0*, *azim=210.0*)

Make plots for all cells with fixed frame.

With one background field as filling and one background field as contrours.

Parameters

- **cell_i**
- **track**
- **cog**
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **mask_total** (*iris.cube.Cube*)
- **field_contour**
- **field_filled**
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.
- **xlim** (*list of float, optional*) – [x_min/1000, x_max/1000]. Default is None.

- **ylim** (*list of float, optional*) – [y_min/1000, y_max/1000]. Default is None.
- **label_field_contour** (*optional*) – Default is None.
- **cmap_field_contour** (*{'Blues', ... }, optional*) – matplotlib.colors. Default is 'Blues'.
- **norm_field_contour** (*optional*) – Default is None.
- **linewidths_contour** (*float, optional*) – Default is 0.8.
- **contour_labels** (*bool, optional*) – Default is False.
- **vmin_field_contour** (*int, optional*) – Default is 0.
- **vmax_field_contour** (*int, optional*) – Default is 50.
- **levels_field_contour** (*optional*) – Default is None.
- **nlevels_field_contour** (*int, optional*) – Default is 10.
- **label_field_filled** (*optional*) – Default is None.
- **cmap_field_filled** (*{'summer', ... }, optional*) – matplotlib.pyplot colormap. Default is 'summer'.
- **norm_field_filled** (*optional*) – Default is None.
- **vmin_field_filled** (*int, optional*) – Default is 0.
- **vmax_field_filled** (*int, optional*) – Default is 100.
- **levels_field_filled** (*optional*) – Default is None.
- **nlevels_field_filled** (*int, optional*) – Default is 10.
- **title** (*optional*) – Default is None.
- **feature_number** (*bool, optional*) – Defaults is False.
- **ele** (*float, optional*) – Default is 10.0.
- **azim** (*float, optional*) – Default is 210.0.

Returns **axes**

Return type cartopy.mpl.geoaxes.GeoAxesSubplot

Notes

needs more descriptions

tobac.plot.plotting.**plot_mask_cell_individual_follow**(*cell_i*, *track*, *cog*, *features*, *mask_total*, *field_contour*, *field_filled*, *axes=None*, *width=10000*, *label_field_contour=None*, *cmap_field_contour='Blues'*, *norm_field_contour=None*, *linewidths_contour=0.8*, *contour_labels=False*, *vmin_field_contour=0*, *vmax_field_contour=50*, *levels_field_contour=None*, *nlevels_field_contour=10*, *label_field_filled=None*, *cmap_field_filled='summer'*, *norm_field_filled=None*, *vmin_field_filled=0*, *vmax_field_filled=100*, *levels_field_filled=None*, *nlevels_field_filled=10*, *title=None*)

Make individual plot for cell centered around cell.

With one background field as filling and one background field as contrours.

Parameters

- **cell_i**
- **track**
- **cog**
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **mask_total** (*iris.cube.Cube*)
- **field_contour**
- **field_filled**
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.
- **width** (*int, optional*) – Default is 10000.
- **label_field_contour** (*optional*) – Default is None.
- **cmap_field_contour** (*{'Blues', ... }, optional*) – matplotlib.colors. Default is 'Blues'.
- **norm_field_contour** (*optional*) – Default is None.
- **linewidths_contour** (*float, optional*) – Default is 0.8.
- **contour_labels** (*bool, optional*) – Default is False.
- **vmin_field_contour** (*int, optional*) – Default is 0.
- **vmax_field_contour** (*int, optional*) – Default is 50.
- **levels_field_contour** (*optional*) – Default is None.
- **label_field_filled** (*optional*) – Default is None.
- **cmap_field_filled** (*{'summer', ... }, optional*) – matplotlib.pyplot colormap, Default is 'summer'.

- **norm_field_filled** (*optional*) – Default is None.
- **vmin_field_filled** (*int, optional*) – Default is 0.
- **vmax_field_filled** (*int, optional*) – Default is 100.
- **levels_field_filled** (*optional*) – Default is None.
- **nlevels_field_filled** (*int, optional*) – Default is 10.
- **title** (*optional*) – Default is None.

Returns axes

Return type cartopy.mpl.geoaxes.GeoAxesSubplot

tobac.plot.plotting.**plot_mask_cell_individual_static**(*cell_i*, *track*, *cog*, *features*, *mask_total*, *field_contour*, *field_filled*, *axes=None*, *xlim=None*, *ylim=None*, *label_field_contour=None*, *cmap_field_contour='Blues'*, *norm_field_contour=None*, *linewidths_contour=0.8*, *contour_labels=False*, *vmin_field_contour=0*, *vmax_field_contour=50*, *levels_field_contour=None*, *nlevels_field_contour=10*, *label_field_filled=None*, *cmap_field_filled='summer'*, *norm_field_filled=None*, *vmin_field_filled=0*, *vmax_field_filled=100*, *levels_field_filled=None*, *nlevels_field_filled=10*, *title=None*, *feature_number=False*)

Make plots for cell in fixed frame

With one background field as filling and one background field as contrours.

Parameters

- **cell_i**
- **track**
- **cog**
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **mask_total** (*iris.cube.Cube*)
- **field_contour**
- **field_filled**
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.
- **xlim** (*list of float, optional*) – [x_min/1000, x_max/1000]. Default is None.
- **ylim** (*list of float, optional*) – [y_min/1000, y_max/1000]. Default is None.
- **label_field_contour** (*optional*) – Default is None.

- **cmap_field_contour** (*{'Blues', ... }, optional*) – matplotlib.colors. Default is 'Blues'.
- **norm_field_contour** (*optional*) – Default is None.
- **linewidths_contour** (*float, optional*) – Default is 0.8.
- **contour_labels** (*bool, optional*) – Default is False.
- **vmin_field_contour** (*int, optional*) – Default is 0.
- **vmax_field_contour** (*int, optional*) – Default is 50.
- **levels_field_contour** (*optional*) – Default is None.
- **nlevels_field_contour** (*int, optional*) – Default is 10.
- **label_field_filled** (*optional*) – Default is None.
- **cmap_field_filled** (*{'summer', ... }, optional*) – matplotlib.pyplot colormap. Default is 'summer'.
- **norm_field_filled** (*optional*) – Default is None.
- **vmin_field_filled** (*int, optional*) – Default is 0.
- **vmax_field_filled** (*int, optional*) – Default is 100.
- **levels_field_filled** (*optional*) – Default is None.
- **nlevels_field_filled** (*int, optional*) – Default is 10.
- **title** (*optional*) – Default is None.
- **feature_number** (*bool, optional*) – Defaults is False.

Returns axes

Return type cartopy.mpl.geoaxes.GeoAxesSubplot

Notes

needs more descriptions

tobac.plot.plotting.**plot_mask_cell_track_2D3Dstatic**(*cell, track, cog, features, mask_total, field_contour, field_filled, width=10000, n_extend=1, name='test', plotdir='./', file_format=['png'], figsize=(3.937007874015748, 3.937007874015748), dpi=300, ele=10, azim=30, **kwargs*)

Make plots for all cells with fixed frame.

Including entire development of the cell and with one background field as filling and one background field as contrours.

Parameters

- **cell**
- **track**
- **cog**
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.

- **mask_total** (*iris.cube.Cube*)
- **field_contour**
- **field_filled**
- **width** (*int, optional*) – Default is 10000.
- **n_extend** (*int, optional*) – Default is 1.
- **name** (*str, optional*) – Default is 'test'.
- **plotdir** (*str, optional*) – Path where the plot will be saved. Default is './'.
- **file_format** (*{['png'], ['pdf']}, optional*) – Default is ['png'].
- **figsize** (*tupel of float, optional*) – Default is (10/2.54, 10/2.54).
- **dpi** (*int, optional*) – Plot resolution. Default is 300.
- ****kwargs**

Returns

Return type none

Notes

needs more descriptions

tobac.plot.plotting.**plot_mask_cell_track_3Dstatic**(*cell, track, cog, features, mask_total, field_contour, field_filled, width=10000, n_extend=1, name='test', plotdir='./', file_format=['png'], figsize=(3.937007874015748, 3.937007874015748), dpi=300, **kwargs*)

Make plots for all cells with fixed frame.

Including entire development of the cell and with one background field as filling and one background field as contrours.

Parameters

- **cell**
- **track**
- **cog**
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **mask_total** (*iris.cube.Cube*)
- **field_contour**
- **field_filled**
- **width** (*int, optional*) – Default is 10000.
- **n_extend** (*int, optional*) – Default is 1.
- **name** (*str, optional*) – Default is 'test'.
- **plotdir** (*str, optional*) – Path where the plot will be saved. Default is './'.

- **file_format** (*{['png'], ['pdf']}, optional*) – Default is ['png'].
- **figsize** (*tupel of float, optional*) – Default is (10/2.54, 10/2.54).
- **dpi** (*int, optional*) – Plot resolution. Default is 300.
- ****kwargs**

Returns

Return type none

Notes

needs more descriptions

`tobac.plot.plotting.`**`plot_mask_cell_track_follow`**(*cell, track, cog, features, mask_total, field_contour, field_filled, width=10000, name='test', plotdir='./', file_format=['png'], figsize=(3.937007874015748, 3.937007874015748), dpi=300, **kwargs*)

Make plots for all cells centered around cell.

With one background field as filling and one background field as contrours.

Parameters

- **cell** (*int*) – Integer id of cell to create masked cube for.
- **track**
- **cog**
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **mask_total** (*iris.cube.Cube*)
- **field_contour**
- **field_filled**
- **width** (*int, optional*) – Default is 10000.
- **name** (*str, optional*) – Default is 'test'.
- **plotdir** (*str, optional*) – Path where the plot will be saved. Default is './'.
- **file_format** (*{['png'], ['pdf']}, optional*) – Default is ['png'].
- **dpi** (*int, optional*) – Plot resolution. Default is 300.
- ****kwargs**

Returns

Return type none

Notes

unsure about features, mask_total needs more descriptions

tobac.plot.plotting.**plot_mask_cell_track_static**(*cell, track, cog, features, mask_total, field_contour, field_filled, width=10000, n_extend=1, name='test', plotdir='./', file_format=['png'], figsize=(3.937007874015748, 3.937007874015748), dpi=300, **kwargs*)

Make plots for all cells with fixed frame.

Inculding entire development of the cell and with one background field as filling and one background field as contrours.

Parameters

- **cell**
- **track**
- **cog**
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **mask_total** (*iris.cube.Cube*)
- **field_contour**
- **field_filled**
- **width** (*int, optional*) – Default is 10000.
- **n_extend** (*int, optional*) – Default is 1.
- **name** (*str, optional*) – Default is 'test'.
- **plotdir** (*str, optional*) – Path where the plot will be saved. Default is './'.
- **file_format** (*{['png'], ['pdf']}, optional*) – Default is ['png'].
- **figsize** (*tupel of float, optional*) – Default is (10/2.54, 10/2.54).
- **dpi** (*int, optional*) – Plot resolution. Default is 300.
- ****kwargs**

Returns

Return type none

Notes

needs more descriptions

tobac.plot.plotting.**plot_mask_cell_track_static_timeseries**(*cell, track, cog, features, mask_total, field_contour, field_filled, track_variable=None, variable=None, variable_ylabel=None, variable_label=[None], variable_legend=False, variable_color=None, width=10000, n_extend=1, name='test', plotdir='./', file_format=['png'], figsize=(7.874015748031496, 3.937007874015748), dpi=300, **kwargs*)

Make plots for all cells with fixed frame.

Including entire development of the cell and with one background field as filling and one background field as contrours.

Parameters

- **cell**
- **track**
- **cog**
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **mask_total** (*iris.cube.Cube*)
- **field_contour**
- **field_filled**
- **track_variable** (*optional*) – Default is None.
- **variable** (*optional*) – Default is None.
- **variable_ylabel** (*optional*) – Default is None.
- **variable_label** (*list, optional*) – Default is [None].
- **variable_legend** (*bool, optional*) – Default is False.
- **variable_color** (*optional*) – Default is None.
- **width** (*int, optional*) – Default is 10000.
- **n_extend** (*int, optional*) – Default is 1.
- **name** (*str, optional*) – Default is 'test'.
- **plotdir** (*str, optional*) – Path where the plot will be saved. Default is './'.
- **file_format** (*{['png'], ['pdf']}, optional*) – Default is ['png'].
- **figsize** (*tupel of float, optional*) – Default is (10/2.54, 10/2.54).

- **dpi** (*int, optional*) – Plot resolution. Default is 300.
- ****kwargs**

Returns axes

Return type cartopy.mpl.geoaxes.GeoAxesSubplot

Notes

needs more descriptions

tobac.plot.plotting.**plot_tracks_mask_field**(*track*, *field*, *mask*, *features*, *axes=None*, *axis_extent=None*, *plot_outline=True*, *plot_marker=True*, *marker_track='x'*, *markersize_track=4*, *plot_number=True*, *plot_features=False*, *marker_feature=None*, *markersize_feature=None*, *title=None*, *title_str=None*, *vmin=None*, *vmax=None*, *n_levels=50*, *cmap='viridis'*, *extend='neither'*, *orientation_colorbar='horizontal'*, *pad_colorbar=0.05*, *label_colorbar=None*, *fraction_colorbar=0.046*, *rasterized=True*, *linewidth_contour=1*)

Parameters

- **track**
- **field** (*iris.cube.Cube*)
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.
- **axis_extent** (*matplotlib.axes, optional*) – Default is None.
- **plot_outline** (*bool, optional*) – Default is True.
- **plot_marker** (*bool, optional*) – Default is True.
- **marker_track** (*str, optional*) – Default is 'x'.
- **markersize_track** (*int, optional*) – Default is 4.
- **plot_number** (*bool, optional*) – Default is True.
- **plot_features** (*bool, optional*) – Default is False.
- **marker_feature** (*optional*) – Default is None.
- **markersize_feature** (*optional*) – Default is None.
- **title** (*optional*) – Default is None.
- **title_str** (*str, optional*) – Default is None.
- **vmin** (*optional*) – Default is None.
- **vmax** (*optional*) – Default is None.

- **n_levels** (*int, optional*) – Default is 50.
- **cmap** (*{'viridis',... }, optional*) – matplotlib.colors. Default is 'viridis'.
- **extend** (*str, optional*) – Default is 'neither'.
- **orientation_colorbar** (*str, optional*) – Default is 'horizontal'.
- **pad_colorbar** (*float, optional*) – Default is 0.05.
- **label_colorbar** (*optional*) – Default is None.
- **fraction_colorbar** (*float, optional*) – Default is 0.046.
- **rasterized** (*bool, optional*) – Default is True.
- **linewidth_contour** (*int, optional*) – Default is 1.

Returns axes

Return type cartopy.mpl.geoaxes.GeoAxesSubplot

Raises `ValueError` – If axes are not cartopy.mpl.geoaxes.GeoAxesSubplot.

If mask.ndim is neither 2 nor 3.

Notes

needs a short summary line and descriptions

`tobac.plot.plotting.`**`plot_tracks_mask_field_loop`**(*track*, *field*, *mask*, *features*, *axes=None*, *name=None*, *plot_dir='./'*, *figsize=(3.937007874015748, 3.937007874015748)*, *dpi=300*, *margin_left=0.05*, *margin_right=0.05*, *margin_bottom=0.05*, *margin_top=0.05*, ***kwargs*)

Parameters

- **track**
- **field** (*iris.cube.Cube*)
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **axes** (*cartopy.mpl.geoaxes.GeoAxesSubplot, optional*) – Default is None.
- **name** (*str*)
- **plot_dir** (*str, optional*) – Path where the plot will be saved. Default is './'.
- **figsize** (*tupel of float, optional*) – Default is (10/2.54, 10/2.54).
- **dpi** (*int, optional*) – Plot resolution. Default is 300.
- **margin_left** (*float, optional*) – Default is 0.05.
- **margin_right** (*float, optional*) – Default is 0.05.
- **margin_bottom** (*float, optional*) – Default is 0.05.
- **margin_top** (*float, optional*) – Default is 0.05.

- ****kwargs**

Returns

Return type none

Notes

needs more descriptions and a short summary line

7.1.4 tobac.utils

Provide essential methods.

Functions

add_coordinates(t, variable_cube)	
column_mask_from2D(mask_2D, cube[, z_coord])	Turn 2D watershedding mask into a 3D mask of selected columns.
get_bounding_box(x[, buffer])	
get_spacings(field_in[, grid_spacing, ...])	**Parameters** • **field_in** (*iris.cube.Cube*) – Input field where to get spacings.
mask_all_surface(mask[, masked, z_coord])	Create surface mask for individual features.
mask_cell(mask, cell, track[, masked])	Create mask for specific cell.
mask_cell_columns(mask, cell, track[, ...])	Create mask with entire columns for individual cell.
mask_cell_surface(mask, cell, track[, ...])	Create surface projection of mask for individual cell.
mask_cube(cube_in, mask)	Mask cube where mask (array) is larger than zero.
mask_cube_all(variable_cube, mask)	Mask cube (iris.cube) for tracked volume.
mask_cube_cell(variable_cube, mask, cell, track)	Mask cube for tracked volume of an individual cell.
mask_cube_features(variable_cube, mask, ...)	Mask cube for tracked volume of an individual cell.
mask_cube_untracked(variable_cube, mask)	Mask cube (iris.cube) for untracked volume.
mask_features(mask, feature_ids[, masked])	Create mask for specific features.
mask_features_surface(mask, feature_ids[, ...])	Create surface mask for individual features.

tobac.utils.**column_mask_from2D**(*mask_2D*, *cube*, *z_coord='model_level_number'*)

Turn 2D watershedding mask into a 3D mask of selected columns.

Parameters

- **cube** (*iris.cube.Cube*) – Data cube.
- **mask_2D** (*iris.cube.Cube*) – 2D cube containing mask (int id for tacked volumes 0 everywhere else).
- **z_coord** (*str*) – Name of the vertical coordinate in the cube.

Returns **mask_2D** – 3D cube containing columns of 2D mask (int id for tacked volumes 0 everywhere else).

Return type iris.cube.Cube

tobac.utils.**get_spacings**(*field_in*, *grid_spacing=None*, *time_spacing=None*)

Parameters

- **field_in** (*iris.cube.Cube*) – Input field where to get spacings.
- **grid_spacing** (*float, optional*) – Grid spacing in input data. Default is None.
- **time_spacing** (*float, optional*) – Time resolution of input data. Default is None.

Returns

- **dxy** (*float*) – Grid spacing.
- **dt** (*float*) – Time resolution.

Raises `ValueError` – If input_cube does not contail projection_x_coord and projection_y_coord or keyword argument grid_spacing.

Notes

need short summary

tobac.utils.**mask_all_surface**(*mask*, *masked=False*, *z_coord='model_level_number'*)

Create surface mask for individual features.

Parameters

- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **masked** (*bool, optional*) – Default is False.
- **z_coord** (*str, optional*) – Name of the vertical coordinate in the cube. Default is 'model_level_number'.

Returns **mask_i_surface** – Mask with 1 below features and 0 everywhere else.

Return type iris.cube.Cube (2D)

tobac.utils.**mask_cell**(*mask*, *cell*, *track*, *masked=False*)

Create mask for specific cell.

Parameters

- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **cell** (*int*) – Interger id of cell to create masked cube for.
- **track**
- **masked** (*bool, optional*) – Default is False.

Returns **mask_i** – Masked cube for untracked volume.

Return type numpy.ndarray

Notes

unsure about mask_i, track and masked

tobac.utils.**mask_cell_columns**(*mask*, *cell*, *track*, *masked=False*, *z_coord='model_level_number'*)

Create mask with entire columns for individual cell.

Parameters

- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **cell** (*int*) – Interger id of cell to create masked cube for.
- **track**
- **masked** (*bool, optional*) – Default is False.
- **z_coord** (*str, optional*) – Default is 'model_level_number'.

Returns **mask_i** – Masked cube for untracked volume.

Return type iris.cube.Cube

tobac.utils.**mask_cell_surface**(*mask*, *cell*, *track*, *masked=False*, *z_coord='model_level_number'*)

Create surface projection of mask for individual cell.

Parameters

- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **cell** (*int*) – Interger id of cell to create masked cube for.
- **track**
- **masked** (*bool, optional*) – Default is False.
- **z_coord** (*str, optional*) – Default is 'model_level_number'.

Returns **mask_i_surface** – Masked cube for untracked volume.

Return type iris.cube.Cube

Notes

unsure about Returns

tobac.utils.**mask_cube**(*cube_in*, *mask*)

Mask cube where mask (array) is larger than zero.

Parameters

- **cube_in** (*iris.cube.Cube*) – Unmasked data cube.
- **mask** (*numpy.ndarray or dask.array*) – Mask to use for masking, >0 where cube is supposed to be masked.

Returns **variable_cube_out** – Masked cube.

Return type iris.cube.Cube

tobac.utils.**mask_cube_all**(*variable_cube*, *mask*)

Mask cube (iris.cube) for tracked volume.

Parameters

- **variable_cube** (*iris.cube.Cube*) – Unmasked data cube.
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).

Returns **variable_cube_out** – Masked cube for untracked volume.

Return type iris.cube.Cube

Notes

unsure about short summary

tobac.utils.**mask_cube_cell**(*variable_cube*, *mask*, *cell*, *track*)
Mask cube for tracked volume of an individual cell.

Parameters

- **variable_cube** (*iris.cube.Cube*) – Unmasked data cube.
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **cell** (*int*) – Interger id of cell to create masked cube for.

Returns **variable_cube_out** – Masked cube with data for respective cell.

Return type iris.cube.Cube

tobac.utils.**mask_cube_features**(*variable_cube*, *mask*, *feature_ids*)
Mask cube for tracked volume of an individual cell.

Parameters

- **variable_cube** (*iris.cube.Cube*) – Unmasked data cube.
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **cell** (*int*) – Interger id of cell to create masked cube for.
- **feature_ids** (*int*)

Returns **variable_cube_out** – Masked cube with data for respective cell.

Return type iris.cube.Cube

tobac.utils.**mask_cube_untracked**(*variable_cube*, *mask*)
Mask cube (iris.cube) for untracked volume.

Parameters

- **variable_cube** (*iris.cube.Cube*) – Unmasked data cube.
- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).

Returns **variable_cube_out** – Masked cube for untracked volume.

Return type iris.cube.Cube

tobac.utils.**mask_features**(*mask*, *feature_ids*, *masked=False*)
Create mask for specific features.

Parameters

- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **feature_ids** (*int*)
- **masked** (*bool, optional*) – Default is False.

Returns **mask_i** – Masked cube for untracked volume.

Return type numpy.ndarray

Notes

unsure about Returns, Parameters need more descriptions

tobac.utils.**mask_features_surface**(*mask*, *feature_ids*, *masked=False*, *z_coord='model_level_number'*)

Create surface mask for individual features.

Parameters

- **mask** (*iris.cube.Cube*) – Cube containing mask (int id for tacked volumes 0 everywhere else).
- **feature_ids** (*int*)
- **masked** (*bool, optional*) – Default is False.
- **z_coord** (*str, optional*) – Name of the vertical coordinate in the cube. Default is 'model_level_number'.

Returns **mask_i_surface** – Masked cube for untracked volume.

Return type iris.cube.Cube

Notes

unsure about Returns

7.2 Theme Modules: Tobac v1

tobac.themes.tobac_v1.feature_detection	Provide feature detection.
tobac.themes.tobac_v1.segmentation	Provide segmentation techniques.
tobac.themes.tobac_v1.tracking	Provide tracking methods.
tobac.themes.tobac_v1.wrapper	Wraps up methods in feature_dection, segmentation and tracking.

7.2.1 tobac.themes.tobac_v1.feature_detection

Provide feature detection.

This module can work with any two-dimensional field either present or derived from the input data. To identify the features, contiguous regions above or below a threshold are determined and labelled individually. To describe the specific location of the feature at a specific point in time, different spatial properties are used to describe the identified region.[2]

References

[2] Heikenfeld, M., Marinescu, P. J., Christensen, M., Watson-Parris, D., Senf, F., van den Heever, S. C., and Stier, P.: tobac v1.0: towards a flexible framework for tracking and analysis of clouds in diverse datasets, Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2019-105 , in review, 2019, 6f.

Functions

feature_detection_multithreshold(field_in, dxy)	Perform feature detection based on contiguous regions.
feature_detection_multithreshold_timestep(i, ...)	Find features in each timestep.
feature_detection_threshold(data_i, i_time)	Find features based on individual threshold value.
feature_position(hdim1_indices, ...)	Determine feature position.
filter_min_distance(features, dxy, min_distance)	Perform feature detection based on contiguous regions.
remove_parents(features_thresholds, ...)	Remove parents of newly detected feature regions.
test_overlap(region_inner, region_outer)	Test for overlap between two regions

tobac.themes.tobac_v1.feature_detection.**feature_detection_multithreshold**(*field_in*, *dxy*, *threshold=None*, *min_num=0*, *target='maximum'*, *position_threshold='center'*, *sigma_threshold=0.5*, *n_erosion_threshold=0*, *n_min_threshold=0*, *min_distance=0*, *feature_number_start=1*)

Perform feature detection based on contiguous regions.

The regions are above/below a threshold.

Parameters

- **field_in** (*iris.cube.Cube*) – 2D field to perform the tracking on (needs to have coordinate 'time' along one of its dimensions),
- **dxy** (*float*) – Grid spacing of the input data.
- **thresholds** (*list of floats, optional*) – Threshold values used to select target regions to track. Default is None.
- **target** (*{'maximum', 'minimum'}, optional*) – Flag to determine if tracking is targetting minima or maxima in the data. Default is 'maximum'.
- **position_threshold** (*{'center', 'extreme', 'weighted_diff',}*) – 'weighted_abs'}, optional Flag choosing method used for the position of the tracked feature. Default is 'center'.
- **sigma_threshold** (*float, optional*) – Standard deviation for intial filtering step. Default is 0.5.
- **n_erosion_threshold** (*int, optional*) – Number of pixel by which to erode the identified features. Default is 0.
- **n_min_threshold** (*int, optional*) – Minimum number of identified features. Default is 0.
- **min_distance** (*float, optional*) – Minimum distance between detected features. Default is 0.

- **feature_number_start** (*int, optional*) – Feature id to start with. Default is 1.

Returns **features** – Detected features.

Return type pandas.DataFrame

tobac.themes.tobac_v1.feature_detection.**feature_detection_multithreshold_timestep**(*data_i*, *i_time*, *threshold=None*, *min_num=0*, *target='maximum'*, *position_threshold*, *sigma_threshold*, *n_erosion_threshold*, *n_min_threshold*, *min_distance*, *feature_number*

Find features in each timestep.

Based on iteratively finding regions above/below a set of thresholds. Smoothing the input data with the Gaussian filter makes output more reliable.[2]

Parameters

- **data_i** (*iris.cube.Cube*) – 2D field to perform the feature detection (single timestep) on.
- **threshold** (*float, optional*) – Threshold value used to select target regions to track. Default is None.
- **min_num** (*int, optional*) – Default is 0.
- **target** (*{'maximum', 'minimum'}, optinal*) – Flag to determine if tracking is targetting minima or maxima in the data. Default is 'maximum'.
- **position_threshold** (*{'center', 'extreme', 'weighted_diff',}* – 'weighted_abs'}, optional Flag choosing method used for the position of the tracked feature. Default is 'center'.
- **sigma_threshold** (*float, optional*) – Standard deviation for intial filtering step. Default is 0.5.
- **n_erosion_threshold** (*int, optional*) – Number of pixel by which to erode the identified features. Default is 0.
- **n_min_threshold** (*int, optional*) – Minimum number of identified features. Default is 0.
- **min_distance** (*float, optional*) – Minimum distance between detected features. Default is 0.
- **feature_number_start** (*int, optional*) – Feature id to start with. Default is 1.

Returns **features_threshold** – Detected features for individual timestep.

Return type pandas.DataFrame

Notes

unsure about feature_number_start

tobac.themes.tobac_v1.feature_detection.**feature_detection_threshold**(*data_i*, *i_time*, *threshold=None*, *min_num=0*, *target='maximum'*, *position_threshold='center'*, *sigma_threshold=0.5*, *n_erosion_threshold=0*, *n_min_threshold=0*, *min_distance=0*, *idx_start=0*)

Find features based on individual threshold value.

Parameters

- **data_i** (*iris.cube.Cube*) – 2D field to perform the feature detection (single timestep) on.
- **i_time** (*int*) – Number of the current timestep.
- **threshold** (*float, optional*) –

  Threshold value used to select target regions to track. Default is None.
- **target** (*{'maximum', 'minimum'}, optional*) – Flag to determine if tracking is targetting minima or maxima in the data. Default is 'maximum'.
- **position_threshold** (*{'center', 'extreme', 'weighted_diff',}* – 'weighted_abs'}, optional Flag choosing method used for the position of the tracked feature. Default is 'center'.
- **sigma_threshold** (*float, optional*) – Standard deviation for intial filtering step. Default is 0.5.
- **n_erosion_threshold** (*int, optional*) – Number of pixel by which to erode the identified features. Default is 0.
- **n_min_threshold** (*int, optional*) – Minimum number of identified features. Default is 0.
- **min_distance** (*float, optional*) – Minimum distance between detected features. Default is 0.
- **idx_start** (*int, optional*) – Feature id to start with. Default is 0.

Returns

- **features_threshold** (*pandas DataFrame*) – Detected features for individual threshold.
- **regions** (*dict*) – Dictionary containing the regions above/below threshold used for each feature (feature ids as keys).

tobac.themes.tobac_v1.feature_detection.**feature_position**(*hdim1_indices*, *hdim2_indeces*, *region*, *track_data*, *threshold_i*, *position_threshold*, *target*)

Determine feature position.

Parameters

- **hdim1_indices, hdim2_indices** (*list*)
- **region** (*list*) – 2-element tuples.

- **track_data** (*numpy.ndarray*) – 2D numpy array containing the data.
- **threshold_i** (*float*)
- **position_threshold** (*str*)
- **target** (*{'maximum', 'minimum'}*) – Flag to determine if tracking is targetting minima or maxima in the data.

Returns **hdim1_index, hdim2_index** – Feature position along 1st and 2nd horizontal dimension.

Return type float

Notes

need more descriptions

tobac.themes.tobac_v1.feature_detection.**filter_min_distance**(*features*, *dxy*, *min_distance*)

Perform feature detection based on contiguous regions.

Regions are above/below a threshold.

Parameters

- **features** (*pandas.DataFrame*)
- **dxy** (*float*) – Grid spacing of the input data.
- **min_distance** (*float, optional*) – Minimum distance between detected features.

Returns **features** – Detected features.

Return type pandas.DataFrame

tobac.themes.tobac_v1.feature_detection.**remove_parents**(*features_thresholds*, *regions_i*, *regions_old*)

Remove parents of newly detected feature regions.

Remove features where its regions surround newly detected feature regions.

Parameters

- **features_thresholds** (*pandas.DataFrame*) – Dataframe containing detected features.
- **regions_i** (*dict*) – Dictionary containing the regions above/below threshold for the newly detected feature (feature ids as keys).
- **regions_old** (*dict*) – Dictionary containing the regions above/below threshold from previous threshold (feature ids as keys).

Returns **features_thresholds** – Dataframe containing detected features excluding those that are superseded by newly detected ones.

Return type pandas.DataFrame

tobac.themes.tobac_v1.feature_detection.**test_overlap**(*region_inner*, *region_outer*)

Test for overlap between two regions

(probably scope for further speedup here)

Parameters **region_inner region_outer** (*list*) – List of 2-element tuples defining the indeces of all cells in the region.

Returns **overlap** – True if there are any shared points between the two regions.

Return type bool

Notes

rework extended summary unsure about description of region_inner, region_outer

7.2.2 tobac.themes.tobac_v1.segmentation

Provide segmentation techniques.

Segmentation techniques are used to associate areas or volumes to each identified feature. The segmentation is implemented using watershedding techniques from the field of image processing with a fixed threshold value. This value has to be set specifically for every type of input data and application. The segmentation can be performed for both two-dimensional and three-dimensional data. At each timestep, a marker is set at the position (weighted mean center) of each feature identified in the detection step in an array otherwise filled with zeros. In case of the three-dimentional watershedding, all cells in the column above the weighted mean center position of the identified features fulfilling the threshold condition are set to the respective marker. The algorithm then fills the area (2D) or volume (3D) based on the input field starting from these markers until reaching the threshold. If two or more cloud objects are directly connected, the border runs along the watershed line between the two regions. This procedure creates a mask of the same shape as the input data, with zeros at all grid points where there is no cloud or updraft and the integer number of the associated feature at all grid points belonging to that specific cloud/updraft. this mask can be conveniently and efficiently used to select the volume of each cloud object at a specific time step for further analysis or visialization.[4]

References

Functions

segmentation(features, field, dxy[, ...])	Use watershedding or random walker.
segmentation_2D(features, field, dxy[, ...])	Prepare the output for the 2D watershedding segmentation.
segmentation_3D(features, field, dxy[, ...])	Prepare the output for the 3D watershedding segmentation.
segmentation_timestep(field_in, features_in, dxy)	Perform watershedding for an individual time step of the data.
watershedding_2D(track, field_in, **kwargs)	**Parameters** • **track**
watershedding_3D(track, field_in, **kwargs)	**Parameters** • **track**

tobac.themes.tobac_v1.segmentation.**segmentation**(*features*, *field*, *dxy*, *threshold=0.003*, *target='maximum'*, *level=None*, *method='watershed'*, *max_distance=None*, *vertical_coord='auto'*)

Use watershedding or random walker.

Determine cloud volumes associated with tracked updrafts.

[4] Heikenfeld, M., Marinescu, P. J., Christensen, M., Watson-Parris, D., Senf, F., van den Heever, S. C., and Stier, P.: tobac v1.0: towards a flexible framework for tracking and analysis of clouds in diverse datasets, Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2019-105 , in review, 2019, 7ff.

Parameters

- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **field** (*iris.cube.Cube*) – Containing the field to perform the watershedding on.
- **dxy** (*float*) – Grid spacing of the input data.
- **threshold** (*float, optional*) – Threshold for the watershedding field to be used for the mask. Default is 3e-3.
- **target** (*{'maximum', 'minimum'}, optional*) – Flag to determine if tracking is targetting minima or maxima in the data. Default is 'maximum'.
- **level** (*slice of iris.cube.Cube, optional*) – Levels at which to seed the cells for the watershedding algorithm. Default is None.
- **method** (*{'watershed'}, optional*) – Flag determining the algorithm to use (currently watershedding implemented). 'random_walk' could be uncommented.
- **max_distance** (*float, optional*) – Maximum distance from a marker allowed to be classified as belonging to that cell. Default is None.
- **vertical_coord** (*{'auto', 'z', 'model_level_number', 'altitude',}*) – 'geopotential_height'}, optional

Returns

- **segmentation_out** (*iris.cube.Cube*) – Cloud mask, 0 outside and integer numbers according to track inside the clouds.
- **features_out** (*pandas.DataFrame*) – Feature dataframe including the number of cells (2D or 3D) in the segmented area/volume of the feature at the timestep.

Raises `ValueError` – If field_in.ndim is neither 3 nor 4 and 'time' is not included in coords.

Notes

vertical_coord needs a description

`tobac.themes.tobac_v1.segmentation.`**`segmentation_2D`**(*features*, *field*, *dxy*, *threshold=0.003*, *target='maximum'*, *level=None*, *method='watershed'*, *max_distance=None*)

Prepare the output for the 2D watershedding segmentation.

Parameters

- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **field** (*iris.cube.Cube*) – Containing the field to perform the watershedding on.
- **dxy** (*float*) – Grid spacing of the input data.
- **threshold** (*float, optional*) – Threshold for the watershedding field to be used for the mask. Default is 3e-3.
- **target** (*{'maximum', 'minimum'}, optional*) – Flag to determine if tracking is targetting minima or maxima in the data. Default is 'maximum'.
- **level** (*slice of iris.cube.Cube, optional*) – Levels at which to seed the cells for the watershedding algorithm. Default is None.

- **method** (*{'watershed'}, optional*) – Flag determining the algorithm to use (currently watershedding implemented). 'random_walk' could be uncommented.
- **max_distance** (*float, optional*) – Maximum distance from a marker allowed to be classified as belonging to that cell. Default is None.

Returns

- **segmentation_out** (*iris.cube.Cube*) – Cloud mask, 0 outside and integer numbers according to track inside the clouds.
- **features_out** (*pandas.DataFrame*) – Feature dataframe including the number of cells (2D or 3D) in the segmented area/volume of the feature at the timestep.

tobac.themes.tobac_v1.segmentation.**segmentation_3D**(*features*, *field*, *dxy*, *threshold=0.003*, *target='maximum'*, *level=None*, *method='watershed'*, *max_distance=None*)

Prepare the output for the 3D watershedding segmentation.

Parameters

- **features** (*pandas.DataFrame*) – Output from trackpy/maketrack.
- **field** (*iris.cube.Cube*) – Containing the field to perform the watershedding on.
- **dxy** (*float*) – Grid spacing of the input data.
- **threshold** (*float, optional*) – Threshold for the watershedding field to be used for the mask. Default is 3e-3.
- **target** (*{'maximum', 'minimum'}, optional*) – Flag to determine if tracking is targetting minima or maxima in the data. Default is 'maximum'.
- **level** (*slice of iris.cube.Cube, optional*) – Levels at which to seed the cells for the watershedding algorithm. Default is None.
- **method** (*{'watershed'}, optional*) – Flag determining the algorithm to use (currently watershedding implemented). 'random_walk' could be uncommented.
- **max_distance** (*float, optional*) – Maximum distance from a marker allowed to be classified as belonging to that cell. Default is None.

Returns

- **segmentation_out** (*iris.cube.Cube*) – Cloud mask, 0 outside and integer numbers according to track inside the clouds.
- **features_out** (*pandas.DataFrame*) – Feature dataframe including the number of cells (2D or 3D) in the segmented area/volume of the feature at the timestep.

tobac.themes.tobac_v1.segmentation.**segmentation_timestep**(*field_in*, *features_in*, *dxy*, *threshold=0.003*, *target='maximum'*, *level=None*, *method='watershed'*, *max_distance=None*, *vertical_coord='auto'*)

Perform watershedding for an individual time step of the data.

Parameters

- **field_in** (*iris.cube.Cube*) – Input field to perform the watershedding on (2D or 3D for one specific point in time).

- **features_in** (*pandas.DataFrame*) – Features for one specific point in time.
- **dxy** (*float*) – Grid spacing of the input data.
- **threshold** (*float, optional*) – Threshold for the watershedding field to be used for the mask. Default is 3e-3.
- **target** (*{'maximum', 'minimum'}, optional*) – Flag to determine if tracking is targetting minima or maxima in the data. Default is 'maximum'.
- **level** (*slice of iris.cube.Cube, optional*) – Levels at which to seed the cells for the watershedding algorithm. Default is None.
- **method** (*{'watershed'}, optional*) – Flag determining the algorithm to use (currently watershedding implemented). 'random_walk' could be uncommented.
- **max_distance** (*float, optional*) – Maximum distance from a marker allowed to be classified as belonging to that cell. Default is None.
- **vertical_coord** (*{'auto', 'z', 'model_level_number', 'altitude',}* – 'geopotential_height'}, optional

Returns

- **segmentation_out** (*iris.cube.Cube*) – Cloud mask, 0 outside and integer numbers according to track inside the clouds.
- **features_out** (*pandas.DataFrame*) – Feature dataframe including the number of cells (2D or 3D) in the segmented area/volume of the feature at the timestep.

Raises `ValueError` – If target is neither 'maximum' nor 'minimum'.

If vertical_coord is not in {'auto', 'z', 'model_level_number', 'altitude', geopotential_height'}.

If there is more than one coordinate name.

If the spatial dimension is neither 2 nor 3.

If method is not 'watershed'.

Notes

unsure about target, dxy and vertical_coord

`tobac.themes.tobac_v1.segmentation.`**`watershedding_2D`**(*track*, *field_in*, ***kwargs*)

Parameters

- **track**
- **field_in** (*iris.cube.Cube*) – Input field to perform the watershedding on (2D or 3D for one specific point in time).
- ****kwargs**

Returns

- **segmentation_out** (*iris.cube.Cube*) – Cloud mask, 0 outside and integer numbers according to track inside the clouds.
- **features_out** (*pandas.DataFrame*) – Feature dataframe including the number of cells (2D or 3D) in the segmented area/volume of the feature at the timestep.

Notes

needs short summary and track needs type and description

tobac.themes.tobac_v1.segmentation.**watershedding_3D**(*track*, *field_in*, ***kwargs*)

Parameters

- **track**
- **field_in** (*iris.cube.Cube*) – Input field to perform the watershedding on (2D or 3D for one specific point in time).
- ****kwargs**

Returns

- **segmentation_out** (*iris.cube.Cube*) – Cloud mask, 0 outside and integer numbers according to track inside the clouds.
- **features_out** (*pandas.DataFrame*) – Feature dataframe including the number of cells (2D or 3D) in the segmented area/volume of the feature at the timestep.

Notes

needs short summary and track needs type and description

7.2.3 tobac.themes.tobac_v1.tracking

Provide tracking methods.

The individual features and associated area/volumes identified in each timestep have to be linked into cloud trajectories to analyse the time evolution of cloud properties for a better understanding of the underlying pyhsical processes.

The implementations are structured in a way that allows for the future addition of more complex tracking methods recording a more complex network of relationships between cloud objects at different points in time.

References

Functions

add_cell_time(t)	Add cell time as time since the initiation of each cell.
fill_gaps(t[, order, extrapolate, ...])	Add cell time as time since the initiation of each cell.
linking_trackpy(features, field_in, dt, dxy)	Perform Linking of features in trajectories.

tobac.themes.tobac_v1.tracking.**add_cell_time**(*t*)

Add cell time as time since the initiation of each cell.

Parameters **t** (*pandas.DataFrame*) – Trajectories from trackpy.

Returns **t** – Trajectories with added cell time.

Return type pandas.DataFrame

tobac.themes.tobac_v1.tracking.**fill_gaps**(*t*, *order=1*, *extrapolate=0*, *frame_max=None*, *hdim_1_max=None*, *hdim_2_max=None*)

Add cell time as time since the initiation of each cell.

Parameters

- **t** (*pandas.DataFrame*) – Trajectories from trackpy.
- **order** (*int, optional*) – Order of polynomial used to extrapolate trajectory into gaps and beyond start and end point. Default is 1.
- **extrapolate** (*int, optional*) – Number or timesteps to extrapolate trajectories. Default is 0.
- **frame_max** (*int, optional*) – Size of input data along time axis. Default is None.
- **hdim_1_max, hdim2_max** (*int, optional*) – Size of input data along first and second horizontal axis. Default is None.

Returns **t** – Trajectories from trackpy with with filled gaps and potentially extrapolated.

Return type pandas.DataFrame

tobac.themes.tobac_v1.tracking.**linking_trackpy**(*features*, *field_in*, *dt*, *dxy*, *v_max=None*, *d_max=None*, *d_min=None*, *subnetwork_size=None*, *memory=0*, *stubs=1*, *time_cell_min=None*, *order=1*, *extrapolate=0*, *method_linking='random'*, *adaptive_step=None*, *adaptive_stop=None*, *cell_number_start=1*)

Perform Linking of features in trajectories.

The linking determines which of the features detected in a specific timestep is identical to an existing feature in the previous timestep. For each existing feature, the movement within a time step is extrapolated based on the velocities in a number previous time steps. The algorithm then breaks the search process down to a few candidate features by restricting the search to a circular search region centered around the predicted position of the feature in the next time step. For newly initialized trajectories, where no velocity from previous timesteps is available, the algorithm resort to the average velocity of the nearest tracked objects. v_max and d_min are given as physical quantities and then converted into pixel-based values used in trackpy. This allows for cloud tracking that is controlled by physically-based parameters that are independent of the temporal and spatial resolution of the input data. The algorithm creates a continuous track for the cloud that most directly follows the direction of travel of the preceding or following cell path.

Parameters

- **features** (*pandas.DataFrame*) – Detected features to be linked.
- **field_in** (*iris.cube.Cube*) – Input field to perform the watershedding on (2D or 3D for one specific point in time).
- **dt** (*float*) – Time resolution of tracked features.
- **dxy** (*float*) – Grid spacing of the input data.
- **d_max** (*optional*) – Default is None.
- **d_min** (*optional*) – Variations in the shape of the regions used to determine the positions of the features can lead to quasi-instantaneous shifts of the position of the feature by one or two grid cells even for a very high temporal resolution of the input data, potentially jeopardising the tracking procedure. To prevent this, tobac uses an additional minimum radius of the search range.

  Default is None.
- **subnetwork_size** (*int, optional*) – Maximim size of subnetwork for linking. Default is None.
- **v_max** (*float, optional*) – Speed at which features are allowed to move. Default is None.

- **memory** (*int, optional*) – Number of output timesteps features allowed to vanish for to be still considered tracked. Default is 0.

  ..warning :: This parameter should be used with caution, as it can lead to erroneous trajectory linking, espacially for data with low time resolution.

- **stubs** (*int, optional*) – Default is 1.
- **time_cell_min** (*optional*) – Default is None.
- **order** (*int, optional*) – Order of polynomial used to extrapolate trajectory into gaps and beyond start and end point. Default is 1.
- **extrapolate** (*int, optional*) – Number or timesteps to extrapolate trajectories. Default is 0.
- **method_linking** (*{'random', 'predict'}, optional*) – Flag choosing method used for trajectory linking. Default is 'random'.
- **adaptive_step** (*optional*) – Default is None.
- **adaptive_stop** (*optional*) – Default is None.
- **cell_number_start** (*int, optional*) – Default is 1.

Returns **trajectories_final** – This enables filtering the resulting trajectories, e.g. to reject trajectories that are only partially captured at the boundaries of the input field both in space and time.

Return type pandas.DataFrame

Raises `ValueError` – If method_linking is neither 'random' nor 'predict'.

Notes

missing type and description: stubs, time_cell_min, extrapolate, adaptive_step, adaptive_stop, cell_number_start, d_max, d_min

7.2.4 tobac.themes.tobac_v1.wrapper

Wraps up methods in feature_dection, segmentation and tracking.

Functions

maketrack(field_in[, grid_spacing, ...])	Identify features and link them into trajectories.
tracking_wrapper(field_in_features, ... [, ...])	**Parameters** • **field_in_features** (*iris.cube.Cube*)

tobac.themes.tobac_v1.wrapper.**maketrack**(*field_in*, *grid_spacing=None*, *time_spacing=None*, *target='maximum'*, *v_max=None*, *d_max=None*, *memory=0*, *stubs=5*, *order=1*, *extrapolate=0*, *method_detection='threshold'*, *position_threshold='center'*, *sigma_threshold=0.5*, *n_erosion_threshold=0*, *threshold=1*, *min_num=0*, *min_distance=0*, *method_linking='random'*, *cell_number_start=1*, *subnetwork_size=None*, *adaptive_stop=None*, *adaptive_step=None*, *return_intermediate=False*)

Identify features and link them into trajectories.

field_in [iris.cube.Cube] 2D input field tracking is performed on.

grid_spacing [float, optional] Grid spacing in input data. Default is None.

time_spacing [float, optional] Time resolution of input data. Default is None.

target [{'maximum', 'minimum'}] Flag to determine if tracking is targetting minima or maxima in the data. Default is 'maximum'.

v_max [float, optional] Speed at which features are allowed to move. Default is None.

d_max [optional] Default is None.

memory [int, optional] Number of timesteps for which objects can be missed by the algorithm to still give a constistent track. Default is 0.

..warning :: This parameter should be used with caution, as it can lead to erroneous trajectory linking, espacially for data with low time resolution.

stubs [float, optional] Default is 5.

order [int, optional] Order if interpolation spline to fill gaps in tracking (from allowing memory to be larger than 0).

method_detection: {'threshold', 'threshold_multi'} Flag choosing method used for feature detection. Default is 'threshold'.

position_threshold [{'center', 'extreme', 'weighted_diff',]

'weighted_abs'}, optional

Flag choosing method used for the position of the tracked feature. Default is 'center'.

sigma_threshold: float, optional Standard deviation for intial filtering step. Default is 0.5.

n_erosion_threshold: int, optional Number of pixel by which to erode the identified features. Default is 0.

min_num [int, optional] Minimum number of cells above threshold in the feature to be tracked. Default is 0.

min_distance [float, optional] Minimum distance between detected features. Default is 0.

method_linking [{'random', 'predict'}, optional] Flag choosing method used for trajectory linking. Default is 'random'.

cell_number_start [int, optional]

Default is 1.

adaptive_step [optional] Default is None.

adaptive_stop [optional] Default is None.

subnetwork_size [int, optional] Maximim size of subnetwork for linking. Default is None.

return_intermediate: bool, optional Flag to determine if only final tracjectories are output (False, default) or if detected features, filtered features and unfilled tracks are returned additionally (True).

trajectories_final: pandas.DataFrame Tracked updrafts, one row per timestep and updraft, includes dimensions 'time', 'latitude', 'longitude', 'projection_x_variable', 'projection_y_variable' based on w cube. 'hdim_1' and 'hdim_2' are used for segementation step.

features : pandas.DataFrame

ValueError If input_cube does not contail projection_x_coord and projection_y_coord or keyword argument grid_spacing.

If method_detection is neither 'threshold' nor 'threshold_multi'.

features needs more information

Optional output: features_filtered: pandas.DataFrame

features_unfiltered: pandas.DataFrame

trajectories_filtered_unfilled: pandas.DataFrame

`tobac.themes.tobac_v1.wrapper.`**`tracking_wrapper`**(*field_in_features*, *field_in_segmentation*, *time_spacing=None*, *grid_spacing=None*, *parameters_features=None*, *parameters_tracking=None*, *parameters_segmentation=None*)

Parameters

- **field_in_features** (*iris.cube.Cube*)
- **field_in_segmentation** (*iris.cube.Cube*)
- **grid_spacing** (*float, optional*) – Grid spacing in input data. Default is None.
- **time_spacing** (*float, optional*) – Time resolution of input data. Default is None.
- **parameters_features** (*optional*) – Default is None.
- **parameters_tracking** (*optional*) – Default is None.
- **parameters_segmentation** (*optional*) – Default is None.

Raises `ValueError` – If method_detection is neither 'threshold' nor 'threshold_multi'.

If method_linking is not 'trackpy'.

Notes

needs short summary unsure about field_in_features, field_in_segmentation and parameters_*

Python Module Index

t

Index

N

P

R

S

T

V

W