IMTreatment Documentation

Release 1.0

Gaby Launay

Feb 12, 2019

Contents

This module has been written to carry out analysis and more specifically structure detection on PIV velocity fields. It is now more general and can handle different kind of data (point cloud, scalar and vector field, ...) and perform classical and more advanced analysis on them (spectra, pod, post-processing, visualization, ...).

Hosted on FramaGit.

Full documentation available on ReadTheDocs.

CHAPTER 1

General data analysis

1. Class representing 2D fields of 1 component (ScalarField)
2. Class representing 2D fields of 2 components (VectorField)
3. Classes representing sets of scalar fields vector fields (SpatialScalarFields, TemporalScalarFields, SpatialVectorFields and TemporalVectorFields)
4. Class representing profiles (Profile)
5. Class representing scatter points (Points)
6. Module for modal decomposition (POD, DMD) and reconstruction (pod)
7. Module to import/export data from/to Davis, matlab, ascii, pivmat and images files (file_operation)
8. Functionalities to visualize those data (plotlib)

CHAPTER 2

Flow analysis

1. Module to create artificial vortices: Burger, Jill, Rankine, ... and to simulate their motion in potential flows (vortex_creation)
2. Module providing several vortex criterions computation (vortex_criterions)
3. Module to automatically detect and track critical points (vortex_detection)
4. Module to compute the evolution of some vortex properties (vortex_properties)
5. Module to generate potential flows with arbitrary geometries (potential_flow)

CHAPTER 3

Dependencies

Mandatory:

- numpy
- matplotlib
- scipy
- unum
- modred

Optional:

- sklearn (to work with point clustering)
- networkx (to use force-directed algorithms to compare trajectories)
- colorama (to have a nice interface when manipulating files)
- h5py (allow to import data from pivmat files)

CHAPTER 4

Installation

You can run `pip install 'git+https://framagit.org/gabylaunay/IMTreatment.git#egg=IMTreatment'` from the shell.

You can try to run the tests with `run_tests.sh`, but the test suite is not plateform-independent yet, and should fail miserably.

If you intend to modify this package, store it some place safe and install it as a developement package with `python setup.py develop`.

CHAPTER 5

Documentation

IMTreatment is documented inline and in ReadTheDocs. you can also use `build_doc.sh` to locally build the html doc.

5.1 IMTreatment - A fields study package

This module has been written to carry out analysis and more specifically structure detection on PIV velocity fields. It is now more general and can handle different kind of data (point cloud, scalar and vector field, ...) and perform classical and more advanced analysis on them (spectra, pod, post-processing, visualization, ...).

Hosted on FramaGit.

Full documentation available on ReadTheDocs.

5.1.1 General data analysis

1. Class representing 2D fields of 1 component (ScalarField)
2. Class representing 2D fields of 2 components (VectorField)
3. Classes representing sets of scalar fields vector fields (SpatialScalarFields, TemporalScalarFields, SpatialVectorFields and TemporalVectorFields)
4. Class representing profiles (Profile)
5. Class representing scatter points (Points)
6. Module for modal decomposition (POD, DMD) and reconstruction (pod)
7. Module to import/export data from/to Davis, matlab, ascii, pivmat and images files (file_operation)
8. Functionalities to visualize those data (plotlib)

5.1.2 Flow analysis

1. Module to create artificial vortices: Burger, Jill, Rankine, ... and to simulate their motion in potential flows (vortex_creation)
2. Module providing several vortex criterions computation (vortex_criterions)
3. Module to automatically detect and track critical points (vortex_detection)
4. Module to compute the evolution of some vortex properties (vortex_properties)
5. Module to generate potential flows with arbitrary geometries (potential_flow)

5.1.3 Dependencies

Mandatory:

- numpy
- matplotlib
- scipy
- unum
- modred

Optional:

- sklearn (to work with point clustering)
- networkx (to use force-directed algorithms to compare trajectories)
- colorama (to have a nice interface when manipulating files)
- h5py (allow to import data from pivmat files)

5.1.4 Installation

You can run `pip install 'git+https://framagit.org/gabylaunay/IMTreatment.git#egg=IMTreatment'` from the shell.

You can try to run the tests with `run_tests.sh`, but the test suite is not plateform-independent yet, and should fail miserably.

If you intend to modify this package, store it some place safe and install it as a developement package with `python setup.py develop`.

5.1.5 Documentation

IMTreatment is documented inline and in ReadTheDocs. you can also use `build_doc.sh` to locally build the html doc.

5.2 API

5.2.1 Classes

5.2.1.1 Points class

class IMTreatment.core.points.**Points**(*xy=array([], shape=(0, 2), dtype=float64), v=[], unit_x='', unit_y='', unit_v='', name=''*)

Bases: object

add(*pt*, *v=None*)
Add a new point.

Parameters

- **pt** (*2x1 array of numbers*) – Point to add.
- **v** (*number, optional*) – Value of the point (needed if other points have values).

augment_resolution(*fact=2*, *interp='linear'*, *inplace=False*)
Augment the temporal resolution of the points. Only have sense if points are sorted to set some kind of trajectory.

Parameters

- **fact** (*integer*) – Resolution augmentation needed (default is '2', for a result profile with twice more points)
- **interp** (*string in ['linear', 'nearest', slinear', 'quadratic, 'cubic']*) – Specifies the kind of interpolation as a string (Default is 'linear'). slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order.
- **bool** (*inplace*) – .

Note: If masked values are present, they are interpolated as well, using the surrounding values.

change_unit(*axe*, *new_unit*)
Change the unit of an axe.

Parameters

- **axe** (*string*) – 'y' for changing the profile y axis unit 'x' for changing the profile x axis unit 'v' for changing the profile values unit
- **new_unit** (*Unum.unit object or string*) – The new unit.

copy()
Return a copy of the Points object.

crop(*intervx=None*, *intervy=None*, *intervv=None*, *inplace=True*, *ind=False*)
Crop the points cloud.

Parameters

- **intervx** (*2x1 tuple*) – Interval on x axis
- **intervy** (*2x1 tuple*) – Interval on y axis
- **intervv** (*2x1 tuple*) – Interval on v values

Returns **tmp_pts** – croped version of the point cloud.

Return type Points object

cut(*intervx=None*, *intervy=None*)
Return a point cloud where the given area has been removed.

Parameters

- **intervx** (*2x1 tuple*) – Interval on x axis
- **intervy** (*2x1 tuple*) – Interval on y axis

Returns **tmp_pts** – Cutted version of the point cloud.

Return type Points object

decompose()
return a tuple of Points object, with only one point per object.

display(*kind=None*, *axe_x=None*, *axe_y=None*, *axe_color=None*, ***plotargs*)
Display the set of points.

Parameters

- **kind** (*string, optional*) – Can be 'plot' (default if points have not values). or 'scatter' (default if points have values). or 'colored_plot'.
- **axe_y, axe_color** (*axe_x,*) – To determine wich value has to be plotted along which axis, and whith value is used to color the scattered points. Default plot 'y' to 'x' with colors from 'v'.
- ****plotargs** (*dic*) – Additionnal arguments sent to 'plot' or 'scatter'

display3D(*kind='plot'*, *xlabel=''*, *ylabel=''*, *zlabel=''*, *title=''*, ***plotargs*)
Display the points on a 3D graph.

Parameters

- **kind** (*string, optional*) – Kind of graph to use, can be 'plot' or 'surf'.
- **ylabel, zlabel** (*xlabel,*) – Label fo each axis (respectively 'x', 'y', and 'v')
- **title** (*strin, optional*) – Title
- ****plotargs** – Additional parameters feeded to matplotlib

export_to_profile(*axe_x='x'*, *axe_y='y'*)
Export the unsorted point object to a sorted Profile object.

Parameters **axe_y** (*axe_x,*) – Which value used to construct the profile

fit(*kind='polynomial'*, *order=2*, *simplify=False*)
Return the parametric coefficients of the fitting curve on the points.

Parameters

- **kind** (*string, optional*) – The kind of fitting used. Can be 'polynomial' or 'ellipse'.
- **order** (*integer*) – Approximation order for the fitting.
- **Simplify** (*boolean or string, optional*) – Can be False (default), 'x' or 'y'. Perform a simplification (see Points.Siplify()) before the fitting.

Returns

- **p** (*array, only for polynomial fitting*) – Polynomial coefficients, highest power first

- **radii** (*array, only for ellipse fitting*) – Ellipse demi-axes radii.
- **center** (*array, only for ellipse fitting*) – Ellipse center coordinates.
- **alpha** (*number*) – Angle between the x axis and the major axis.

get_clusters(*eps*, *min_samples=5*)
Perform DBSCAN clustering from vector array or distance matrix. (see sklearn.cluster.DBSCAN)

5.2.1.1.1 Notes

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

get_envelope(*alpha=None*)
Return the convex or concave hull (if alpha specified) for the set of points.

Parameters **alpha** (*number*) – maximum distance between two points of the hull.

5.2.1.1.2 Notes

Credit to mlaloux (https://github.com/mlaloux/Python–alpha-shape_concave_hull)

get_evolution_on_sf(*SF*, *axe_x=None*)
Return the evolution of the value represented by a scalar field, on the path of the trajectory.

Parameters

- **SF** (*ScalarField object*) –
- **axe_x** (*string, optional*) – What put in the x axis (can be 'x', 'y', 'v'). default is 'v' when available and 'x' else.

Returns **evol**

Return type Profile object

get_evolution_on_tsf(*TSF*, *axe_x=None*)
Return the evolution of the value represented by scalar fields, on the path of the trajectory. Timse of the TSF must be consistent with the times of the Points.

Parameters

- **TSF** (*tsf.TemporalScalarField object*) –
- **axe_x** (*string, optional*) – What put in the x axis (can be 'x', 'y', 'v'). default is 'v' (associated with time)

Returns **evol**

Return type Profile object

get_points_density(*bw_method=None*, *resolution=100*, *output_format='normalized'*, *raw=False*)
Return a ScalarField with points density.

Parameters

- **bw_method** (*str, scalar or callable, optional*) – The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If a scalar, this will be used as percent of the data std. If a callable, it should take

a gaussian_kde instance as only parameter and return a scalar. If None (default), 'scott' is used.

- **resolution** (*integer, optional*) – Resolution for the resulting field.
- **output_format** (*string, optional*) –

  'normalized' (default) [give position probability] (integral egal 1).

  'ponderated' [give position probability ponderated by the number] or points (integral egal number of points).

  'concentration' : give local concentration (in point per surface).
- **raw** (*boolean, optional*) – If 'False' (default), return a ScalarField object, if 'True', return numpy array.

Returns density – Return 'None' if there is not enough points in the cloud.

Return type array, ScalarField object or None

get_points_density2(*res*, *subres=None*, *raw=False*, *ponderated=False*)

Return a ScalarField with points density.

Parameters

- **res** (*number or 2x1 array of numbers*) – fdensity field number of subdivision. Can be the same number for both axis, or one number per axis (need to give a tuple).
- **raw** (*boolean, optional*) – If 'False' (default), return a ScalarField object, if 'True', return numpy array.
- **ponderated** (*boolean, optiona*) – If 'True', values associated to points are used to ponderate the density field. Default is 'False'.
- **subres** (*odd integer, optional*) – If specified, a subgrid of resolution res*subres is used to make result more accurate.

get_velocity(*incr=1*, *smooth=0*, *xaxis='time'*)

Assuming that associated 'v' values are times for each points, compute the velocity of the trajectory.

Parameters

- **incr** (*integer, optional*) – Increment use to get used points (default is 1).
- **smooth** (*number, optional*) – Cut off frequency for the lowpass filter.
- **xaxis** (*string, optional*) – Value to put in the profile x axis, can be 'time' (default), 'x' or 'y'.

Returns

- **Vx** (*Profile object*) – Profile of x velocity versus time.
- **Vy** (*Profile object*) – Profile of y velocity versus time.

name

order_on_line(*inplace=False*)

Re-order the set of points to try to create a line.

remove(*ind*)

Remove the point number 'ind' of the points cloud. In place.

Parameters ind (*integer or array of integer*) –

remove_doublons(*method='average'*, *inplace=False*, *eps_rel=1e-06*)
Replace values associated to the same 'v' by their average.

Parameters **method** (*string in {'average', 'max', 'min'}*) – Method used to remove the doublons.

remove_nans(*inplace=False*)
Remove the points containing nans values.

reverse()
Return a Points object where x and y axis are swaped.

rotate(*angle*, *inplace=False*)
Rotate the point set.

Parameters **angle** (*number*) – Rotation angle in radian.

scale(*scalex=1.0*, *scaley=1.0*, *scalev=1.0*, *inplace=False*)
Change the scale of the axis.

Parameters

- **scaley, scalev** (*scalex,*) – scales along x, y and v
- **inplace** (*boolean, optional*) – If 'True', scaling is done in place, else, a new instance is returned.

set_origin(*x=None*, *y=None*)
Set the given point (x, y) as the new referential.

Parameters

- **x** (*number*) – .
- **y** (*number*) – .

smooth(*tos='gaussian'*, *size=None*, *inplace=False*, ***kw*)
Return a smoothed points field.

Parameters

- **tos** (*string, optional*) – Type of smoothing, can be 'uniform' (default), 'gaussian' or 'lowpass'.
- **size** (*number, optional*) – radius of the smoothing for 'uniform', radius of the smoothing for 'gaussian', cut off frequency for 'lowpass' Default are 3 for 'uniform', 1 for 'gaussian' and 0.1 for ' lowpass'.
- **inplace** (*boolean*) – If 'False', return a smoothed points field else, smooth in place.
- **kw** (*dic*) – Additional parameters for ndimage methods (See ndimage documentation)

sort(*ref='x'*, *inplace=False*)
Sort the points according to the reference.

Parameters

- **ref** (*string or array of indice*) – can be 'x', 'y' or 'v' to sort according those values or an array of indice
- **; boolean** (*inplace*) – If 'True', sort in place, else, return an new sorted instance.

unit_v

unit_x

unit_y

v

xy

5.2.1.2 Profile class

class IMTreatment.core.profile.**Profile**(*x=[]*, *y=[]*, *mask=False*, *unit_x=''*, *unit_y=''*, *name=''*)

Bases: object

Class representing a profile. You can use 'make_unit' to provide unities.

Parameters

- **y** (*x,*) – Profile values.
- **unit_y** (*unit_x,*) – Values unities.
- **name** (*string, optionnal*) – A name for the profile.

add_point(*x*, *y*)
Add the given point to the profile.

add_points(*prof*)
Add points from another profile.

augment_resolution(*fact=2*, *interp='linear'*, *inplace=True*)
Augment the temporal resolution of the profile.

Parameters

- **fact** (*integer*) – Resolution augmentation needed (default is '2', for a result profile with twice more points)
- **interp** (*string in ['linear', 'nearest', slinear', 'quadratic, 'cubic']*) – Specifies the kind of interpolation as a string (Default is 'linear'). slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order.
- **bool** (*inplace*) – .

Note: If masked values are present, they are interpolated as well, using the surrounding values.

change_unit(*axe*, *new_unit*)
Change the unit of an axe.

Parameters

- **axe** (*string*) – 'y' for changing the profile values unit 'x' for changing the profile axe unit
- **new_unit** (*Unum.unit object or string*) – The new unit.

copy()
Return a copy of the Profile object.

crop(*intervx=None*, *intervy=None*, *ind=False*, *inplace=False*)
Crop the profile along 'x'.

Parameters

- **intervx** (*array of two numbers*) – Bound values of x.

- **intervy** (*array of two numbers*) – Bound values of y.
- **ind** (*Boolean, optionnal*) – If 'False' (Default), 'intervx' and 'intervy' are values along x axis, if 'True', 'intervx' and 'intervy' are indices of values along x.
- **inplace** (*boolean, optional*) – .

crop_masked_border (*inplace=False*)
Remove the masked values at the border of the profile in place or not.

display (*kind='plot'*, *reverse=False*, ***plotargs*)
Display the profile.

Parameters

- **reverse** (*Boolean, optionnal*) – If 'False', x is put in the abscissa and y in the ordinate. If 'True', the inverse.
- **kind** (*string*) – Kind of display to plot ('plot', 'semilogx', 'semilogy', 'loglog')
- ****plotargs** (*dict, optionnale*) – Additional argument for the 'plot' command.

Returns **fig** – Reference to the displayed plot.

Return type Plot reference

evenly_space (*kind_interpolation='linear'*, *dx=None*, *inplace=False*)
Return a profile with evenly spaced x values. Use interpolation to get missing values.

Parameters **kind_interpolation** (*string or int, optional*) – Specifies the kind of interpolation as a string ('value', 'linear', 'nearest', 'zero', 'slinear', 'quadratic, 'cubic' where 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order) or as an integer specifying the order of the spline interpolator to use. Default is 'linear'.

fill (*kind='slinear'*, *fill_value=0.0*, *inplace=False*, *crop=False*)
Return a filled profile (no more masked values).

Warning : If 'crop' is False, border masked values can't be interpolated and are filled with 'fill_value' or the nearest value.

Parameters

- **kind** (*string or int, optional*) – Specifies the kind of interpolation as a string ('value', 'linear', 'nearest', 'zero', 'slinear', 'quadratic, 'cubic' where 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order) or as an integer specifying the order of the spline interpolator to use. Default is 'linear'.
- **fill_value** (*number, optional*) – For kind = 'value', filling value.
- **inplace** (*boolean, optional*) – .
- **crop** (*boolean, optional*) – .

Returns **prof** – Filled profile

Return type Profile object

get_auto_correlation (*window_len*, *raw=False*)
Return the auto-correlation profile.

This algorithm make auto-correlation for all the possible values, and an average of the resulting profile. Profile are normalized, so the central value of the returned profile should be 1.

Parameters

- **window_len** (*integer*) – Window length for sweep correlation.
- **raw** (*bool, optional*) – If 'True', return an array If 'False' (default), return a profile

get_convolution(*other_prof*, *mode='full'*)
Return the convolution with the give profile.

Parameters

- **other_prof** (*Profile object*) – .
- **mode** (*{'full', 'valid', 'same'}, optional*) –

  'full': By default, mode is 'full'. This returns the convolution at each point of overlap, with an output shape of (N+M-1,). At the end-points of the convolution, the signals do not overlap completely, and boundary effects may be seen.

  'same': Mode same returns output of length max(M, N). Boundary effects are still visible.

  'valid': Mode valid returns output of length max(M, N) - min(M, N) + 1. The convolution product is only given for points where the signals overlap completely. Values outside the signal boundary have no effect.

Returns **conv_prof** – Result of the convolution

Return type Profile object

5.2.1.2.1 Notes

Use the numpy function 'convolve'.

get_convolution_of_difference(*other_profile*, *normalized=True*)
Return a convolution that use difference instead of multiplication.

Note: Difference is not normaized, but averaged on the available points.

---

get_dephasage(*other_profile*, *conv='difference'*)
Return the dephasage between the two profiles using convolution

Parameters **conv** (*string in ['classic', 'difference']*) – The convection type to use

Returns **dep** – Dephasage, in profiles unit

Return type number

get_distribution(*output_format='normalized'*, *resolution=100*, *bw_method='scott'*)
Return he distribution of y values by using gaussian kernel estimator.

Parameters

- **output_format** (*string, optional*) – 'normalized' (default) : give position probability (integral egal 1). 'ponderated' : give position probability ponderated by the number

  or points (integral egal number of points).

  'concentration' : give local concentration (in point per length).
- **resolution** (*integer*) – Resolution of the resulting profile (number of values in it).

- **bw_method** (*str or scalar, optional*) – The method used to calculate the estimator bandwidth. Can be 'scott', 'silverman' or a number to set manually the gaussians std (it should aproximately be the size of the density node you want to see). (see 'scipy.stats.gaussian_kde' documentation for more details)

Returns **distrib** – The y values distribution.

Return type Profile object

get_extrema_position(*smoothing=None*, *ind=False*, *debug=False*)
Return the local extrema of the profile.

Parameters

- **smoothing** (*number, optional*) – Size of the gaussian smoothing to apply before extrema detection.
- **ind** (*bool, optional*) – If 'True', return indice position, else, return position along x axis (default is 'False').

Returns **min_pos, max_pos** – .

Return type arrays of numbers

get_fitting(*func*, *p0=None*, *output_param=False*)
Use non-linear least squares to fit a function, f, to the profile.

Parameters

- **func** (*callable*) – The model function, f(x, ...). It must take the independent variable as the first argument and the parameters to fit as separate remaining arguments.
- **p0** (*None, scalar, or M-length sequence*) – Initial guess for the parameters. If None, then the initial values will all be 1 (if the number of parameters for the function can be determined using introspection, otherwise a ValueError is raised).
- **output_param** (*boolean, optional*) – If 'False' (default), return only a Profile with fitted values If 'True', return also the parameters values.

Returns

- **fit_prof** (*Profile obect*) – The Fitted profile.
- **params** (*tuple, optional*) – Fitting parameters.

get_gradient(*position=None*, *wanted_dx=None*)
Return the profile gradient. If 'position' is renseigned, interpolations or finite differences are used to get the gradient at x = position. Else, a profile with gradient at profile points is returned. Warning : only work with evenly spaced x

Parameters

- **position** (*number, optional*) – Wanted point position
- **wanted_dx** (*number, optional*) – interval on which compute gradient when position is renseigned (default is dx similar to axis).

get_integral()
Return the profile integral, and is unit. Use the trapezoidal aproximation.

get_interpolated_values(*x=None*, *y=None*, *ind=False*)
Get the interpolated (or not) value for given 'x' or 'y' values.

If several possibilities are possible, an array with all the results is returned.

Parameters

- **x** (*number or array of number*) – Value(s) of x, for which we want the y value.
- **y** (*number or array of number*) – Value(s) of y, for which we want the x value.
- **ind** (*boolean*) – If 'True', 'x' and 'y' are treated as indices, else, they are treated as position alog axis.

Returns **i_values** – Interpolated value(s).

Return type number or array

get_interpolator(*kind='linear'*, *bounds_error=True*, *fill_value=nan*)
Return an interpolator of the profile

Parameters

- **kind** (*str or int, optional*) – Specifies the kind of interpolation as a string ('linear', 'nearest', 'zero', 'slinear', 'quadratic, 'cubic' where 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order) or as an integer specifying the order of the spline interpolator to use. Default is 'linear'.
- **bounds_error** (*bool, optional*) – If True, a ValueError is raised any time interpolation is attempted on a value outside of the range of x (where extrapolation is necessary). If False, out of bounds values are assigned *fill_value*. By default, an error is raised unless *fill_value="extrapolate"*.
- **fill_value** (*array-like or (array-like, array_like) or "extrapolate"*) –

  – if a ndarray (or float), this value will be used to fill in for

  requested points outside of the data range. If not provided, then the default is NaN. The array-like must broadcast properly to the dimensions of the non-interpolation axes. - If a two-element tuple, then the first element is used as a fill value for `x_new < x[0]` and the second element is used for `x_new > x[-1]`. Anything that is not a 2-element tuple (e.g., list or ndarray, regardless of shape) is taken to be a single array-like argument meant to be used for both bounds as `below, above = fill_value, fill_value`. - If "extrapolate", then points outside the data range will be extrapolated.

Returns **interpolator** – Take a single value 'x' and return the interpolated value of 'y'.

Return type function

Note: Use scipy.interpolate module

---

get_pdf(*bw_method='scott'*, *resolution=1000*, *raw=False*)
Return the probability density function.

Parameters

- **bw_method** (*str, scalar or callable, optional*) – The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If a scalar, this will be used directly as kde.factor. If a callable, it should take a gaussian_kde instance as only parameter and return a scalar. If None (default), 'scott' is used. See 'scipy.stats.kde' for more details.
- **resolution** (*integer, optional*) – Resolution of the returned pdf.
- **raw** (*boolean, optional*) – If 'True', return an array, else, return a Profile object.

get_props()
Print the Profile main properties

get_spectrum(*wanted_x=None*, *welch_seglen=None*, *scaling='base'*, *fill='linear'*, *mask_error=True*, *detrend='constant'*)

Return a Profile object, with the frequential spectrum of 'component', on the point 'pt'.

Parameters

- **wanted_x** (*2x1 array, optional*) – Time interval in which compute spectrum (default is all).
- **welch_seglen** (*integer, optional*) – If specified, welch's method is used (dividing signal into overlapping segments, and averaging periodogram) with the given segments length (in number of points).
- **scaling** (*string, optional*) – If 'base' (default), result are in component unit. If 'spectrum', the power spectrum is returned (in unit^2). If 'density', the power spectral density is returned (in unit^2/(1/unit_x))
- **fill** (*string or float*) – Specifies the way to treat missing values. A value for value filling. A string ('linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' where 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order) for interpolation.
- **mask_error** (*boolean*) – If 'False', instead of raising an error when masked value appear on time profile, '(None, None)' is returned.
- **detrend** (*string, optional*) – Method used to detrend the profile. Can be 'none', 'constant' (default) or 'linear'.

Returns **magn_prof** – Magnitude spectrum.

Return type Profile object

get_value_position(*value*, *ind=False*)

Return the interpolated position(s) of the wanted value.

Parameters

- **value** (*number*) – .
- **ind** (*boolean*) – If 'True', return the value indices, else, return the 'y' position. (Default is 'False')

get_wavelet_transform(*widths=None*, *fill='linear'*, *raw=False*, *verbose=False*)

Return the wavelet transformation of the profile.

Parameters

- **widths** (*array of number, optional*) – Widths of the wavelet to use (by default use 100 homogeneously distributed wavelets)
- **fill** (*string or float*) – Specifies the way to treat missing values. A value for value filling. A string ('linear', 'nearest', 'zero', 'slinear', 'quadratic, 'cubic' where 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order) for interpolation.
- **raw** (*bool*) – if 'True', return an array, else (default), return a ScalarField object.
- **verbose** (*boolean*) – If 'True', display message on the computing advancement.

Warning: Only work with uniformely spaced data.

mask

max
Return the maxima along an axe.

Parameters **axe** (*integer, optionnal*) – Axe along which we want the maxima.

Returns **max** – Maxima along 'axe'.

Return type number

mean
Return the minima along an axe.

Parameters **axe** (*integer, optionnal*) – Axe along which we want the minima.

Returns **max** – Minima along 'axe'.

Return type number

min
Return the minima along an axe.

Parameters **axe** (*integer, optionnal*) – Axe along which we want the minima.

Returns **max** – Minima along 'axe'.

Return type number

remove_doublons (*method='average'*, *inplace=False*, *eps_rel=1e-06*)
Replace values associated to the same 'x' by their average.

Parameters **method** (*string in {'average', 'max', 'min'}*) – Method used to remove the doublons.

remove_local_marginal_values (*fact=5*, *neighb=20*, *inplace=False*)
Remove (mask) the local marginal values.

Parameters

- **fact** (*positive number*) – Number of standard deviation in the 'acceptable' value range. (default to 5)
- **neighb** (*positive number*) – Size of the neighbourhood to consider to check if a value if marginal or not.

remove_marginal_values (*fact=5*, *inplace=False*)
Remove (mask) the marginal values.

Parameters **fact** (*positive number*) – Number of standard deviation in the 'acceptable' value range. (default to 5)

remove_nans (*inplace=False*)
Remove the NaNs points from the profile.

remove_point (*ind*)
Remove a point from the profile

Parameters **ind** (*integer*) – Idice of the point to remove

rotate (*angle*, *inplace=False*)
Rotate the profile.

Parameters **angle** (*number*) – Rotation angle in radian.

scale (*scalex=1.0*, *scaley=1.0*, *inplace=False*)
Change the scale of the axis.

Parameters

- **scaley** (*scalex,*) – scales along x and y
- **inplace** (*boolean, optional*) – If 'True', scaling is done in place, else, a new instance is returned.

smooth(*tos='uniform'*, *size=None*, *direction='y'*, *inplace=False*, ***kw*)
Return a smoothed profile. Warning : fill up the field

Parameters

- **tos** (*string, optional*) – Type of smoothing, can be 'uniform' (default) or 'gaussian' (See ndimage module documentation for more details)
- **size** (*number, optional*) – Size of the smoothing (is radius for 'uniform' and sigma for 'gaussian'). Default is 3 for 'uniform' and 1 for 'gaussian'.
- **dir** (*string, optional*) – In which direction smoothing (can be 'x', 'y' or 'xy').
- **inplace** (*boolean*) – If 'False', return a smoothed profile else, smooth in place.
- **kw** (*dic*) – Additional parameters for ndimage methods (See ndimage documentation)

spectral_filtering(*fmin=None*, *fmax=None*, *order=2*)
Perform a spectral filtering (highpass, lowpass, bandpass).

Parameters

- **fmax** (*fmin,*) – Minimal and maximal frequencies
- **order** (*integer, optional*) – Butterworth filter order

Returns **filt_prof** – Filtered profile

Return type Profile object

unit_x

unit_y

x

y

5.2.1.3 ScalarField class

class IMTreatment.core.scalarfield.**ScalarField**
Bases: *IMTreatment.core.field.Field*

Class representing a scalar field (2D field, with one component on each point).

change_dtype(*new_type*)
Change the values dtype.

change_unit(*axe*, *new_unit*)
Change the unit of an axe.

Parameters

- **axe** (*string*) – 'y' for changing the profile y axis unit 'x' for changing the profile x axis unit 'values' or changing values unit
- **new_unit** (*Unum.unit object or string*) – The new unit.

copy()
Return a copy of the scalarfield.

crop(*intervx=None*, *intervy=None*, *ind=False*, *inplace=False*)
Crop the area in respect with given intervals.

Parameters

- **intervx** (*array, optional*) – interval wanted along x
- **intervy** (*array, optional*) – interval wanted along y
- **ind** (*boolean, optional*) – If 'True', intervals are understood as indices along axis. If 'False' (default), intervals are understood in axis units.
- **inplace** (*boolean, optional*) – If 'True', the field is croped in place.

crop_masked_border(*hard=False*, *inplace=False*)
Crop the masked border of the field in place or not.

Parameters **hard** (*boolean, optional*) – If 'True', partially masked border are croped as well.

display(*component=None*, *kind=None*, ***plotargs*)
Display the scalar field.

Parameters

- **component** (*string, optional*) – Component to display, can be 'values' or 'mask'
- **kind** (*string, optinnal*) – If 'imshow': (default) each datas are plotted (imshow), if 'contour': contours are ploted (contour), if 'contourf': filled contours are ploted (contourf).
- ****plotargs** (*dict*) – Arguments passed to the 'contourf' function used to display the scalar field.

Returns **fig** – Reference to the displayed figure.

Return type figure reference

export_to_scatter(*mask=None*)
Return the scalar field under the form of a pts.Points object.

Parameters **mask** (*array of boolean, optional*) – Mask to choose values to extract (values are taken where mask is False).

Returns **Pts** – Contening the ScalarField points.

Return type pts.Points object

extend(*nmb_left=0*, *nmb_right=0*, *nmb_up=0*, *nmb_down=0*, *value=None*, *inplace=False*, *ind=True*)
Add columns or lines of masked values at the scalarfield.

Parameters

- **nmb_****** (*integers*) – Number of lines/columns to add in each direction.
- **value** (*None or number*) – Value used to fill the new columns and lines. If 'value' is not given, new columns and lines are masked.
- **inplace** (*bool*) – If 'False', return a new extended field, if 'True', modify the field inplace.

Returns **Extended_field** – Extended field.

Return type Field object, optional

fill(*kind='linear'*, *value=0.0*, *inplace=False*, *reduce_tri=True*, *crop=False*)
Fill the masked part of the array.

Parameters

- **kind** (*string, optional*) – Type of algorithm used to fill. 'value' : fill with the given value 'nearest' : fill with the nearest value 'linear' (default): fill using linear interpolation (Delaunay triangulation) 'cubic' : fill using cubic interpolation (Delaunay triangulation)
- **value** (*number*) – Value used to fill (for kind='value').
- **inplace** (*boolean, optional*) – If 'True', fill the ScalarField in place. If 'False' (default), return a filled version of the field.
- **reduce_tri** (*boolean, optional*) – If 'True', treatment is used to reduce the triangulation effort (faster when a lot of masked values) If 'False', no treatment (faster when few masked values)
- **crop** (*boolean, optional*) – If 'True', SF borders are croped before filling.

get_histogram(*cum=False*, *normalized=False*, *bins=None*, *range=None*)
Return the image histogram.

Parameters **cum** (*boolean*) – If True, get a cumulative histogram.

Returns **hist** – Histogram.

Return type array of numbers

get_interpolator(*interp='linear'*)
Return the field interpolator.

Parameters **kind** (*{'linear', 'cubic', 'quintic'}, optional*) – The kind of spline interpolation to use. Default is 'linear'.

get_nearest_extrema(*pts*, *extrema='max'*, *ind=False*)
For a given set of points, return the positions of the nearest local extrema (minimum or maximum).

Parameters **pts** (*Nx2 array*) – Set of pts.Points position.

Returns **extremum_pos**

Return type Nx2 array

get_norm(*norm=2*, *normalized='perpoint'*)
Return the field norm

get_profile(*direction*, *position*, *ind=False*, *interp='linear'*)
Return a profile of the scalar field, at the given position (or at least at the nearest possible position). If position is an interval, the fonction return an average profile in this interval.

Parameters

- **direction** (*string in ['x', 'y']*) – Direction along which we choose a position.
- **position** (*float, interval of float or string*) – Position, interval in which we want a profile or 'all'
- **ind** (*boolean*) – If 'True', position has to be given in indices If 'False' (default), position has to be given in axis unit.
- **interp** (*string in ['nearest', 'linear']*) – if 'nearest', get the profile at the nearest position on the grid, if 'linear', use linear interpolation to get the profile at the exact position

Returns **profile** – Wanted profile

Return type prof.Profile object

get_props()
Print the ScalarField main properties

get_spatial_autocorrelation(*direction*, *window_len=None*)
Return the spatial auto-correlation along the wanted direction.

Take the middle point for reference for correlation computation.

Parameters

- **direction** (*string*) – 'x' or 'y'
- **window_len** (*integer, optional*) – Window length for sweep correlation. if 'None' (default), all the signal is used, and boundary effect can be seen.

Returns **profile** – Spatial correlation

Return type prof.Profile object

get_spatial_spectrum(*direction*, *intervx=None*, *intervy=None*, *welch_seglen=None*, *scaling='base'*, *fill='linear'*)
Return a spatial spectrum.

Parameters

- **direction** (*string*) – 'x' or 'y'.
- **and intervy** (*intervx*) – To chose the zone where to calculate the spectrum. If not specified, the biggest possible interval is choosen.
- **welch_seglen** (*integer, optional*) – If specified, welch's method is used (dividing signal into overlapping segments, and averaging periodogram) with the given segments length (in number of points).
- **scaling** (*string, optional*) – If 'base' (default), result are in component unit. If 'spectrum', the power spectrum is returned (in unit^2). If 'density', the power spectral density is returned (in unit^2/(1/unit_axe))
- **fill** (*string or float*) – Specifies the way to treat missing values. A value for value filling. A string ('linear', 'nearest' or 'cubic') for interpolation.

Returns **spec** – Magnitude spectrum.

Return type prof.Profile object

5.2.1.3.1 Notes

If there is missing values on the field, 'fill' is used to linearly interpolate the missing values (can impact the spectrum).

get_value(*x*, *y*, *ind=False*, *unit=False*)
Return the scalar field value on the point (x, y). If ind is true, x and y are indices, else, x and y are value on axes (interpolated if necessary).

get_zones_centers(*bornes=[0.75, 1], rel=True, kind='ponderated'*)
Return a pts.Points object contening centers of the zones lying in the given bornes.

Parameters

- **bornes** (*2x1 array, optionnal*) – Trigger values determining the zones. '[inferior borne, superior borne]'

- **rel** (*Boolean*) – If 'rel' is 'True' (default), values of 'bornes' are relative to the extremum values of the field. If 'rel' is 'False', values of bornes are treated like absolute values.
- **kind** (*string, optional*) – if 'kind' is 'center', given points are geometrical centers, if 'kind' is 'extremum', given points are extrema (min or max) on zones if 'kind' is 'ponderated'(default, given points are centers of mass, ponderated by the scaler field.

Returns **pts** – Contening the centers coordinates

Return type pts.Points object

import_from_arrays(*axe_x*, *axe_y*, *values*, *mask=None*, *unit_x=''*, *unit_y=''*, *unit_values=''*, *dtype=<class 'float'>*, *dontchecknans=False*, *dontcheckunits=False*)

Set the field from a set of arrays.

Parameters

- **axe_x** (*array*) – Discretized axis value along x
- **axe_y** (*array*) – Discretized axis value along y
- **values** (*array or masked array*) – Values of the field at the discritized points
- **unit_x** (*String unit, optionnal*) – Unit for the values of axe_x
- **unit_y** (*String unit, optionnal*) – Unit for the values of axe_y
- **unit_values** (*String unit, optionnal*) – Unit for the scalar field
- **dontchecknans** (*boolean*) – Don't check for nans values (faster)
- **dontcheckunits** (*boolean*) – Don't check for unit consistency (faster)
- **dtype** (*Numerical type*) – Numerical type to store the data to. Should be a type supported by numpy arrays. Default to 'float'.

integrate_over_line(*direction*, *interval*)

Return the integral on an interval and along a direction ('x' or 'y'). Discretized integral is computed with a trapezoidal algorithm.

Parameters

- **direction** (*string in ['x', 'y']*) – Direction along which we choose a position.
- **interval** (*interval of numbers*) – Interval on which we want to calculate the integral.

Returns

- **integral** (*float*) – Result of the integrale calcul
- **unit** (*Unit object*) – Unit of the result

integrate_over_surface(*intervx=None*, *intervy=None*)

Return the integral on a surface. Discretized integral is computed with a very rustic algorithm which just sum the value on the surface. if 'intervx' and 'intervy' are given, return the integral over the delimited surface. WARNING : Only works (and badly) with regular axes.

Parameters

- **intervx** (*interval of numbers, optional*) – Interval along x on which we want to compute the integrale.
- **intervy** (*interval of numbers, optional*) – Interval along y on which we want to compute the integrale.

Returns

- **integral** (*float*) – Result of the integrale computation.
- **unit** (*Unit object*) – The unit of the integrale result.

make_evenly_spaced(*interp='linear'*, *res=1*)
Use interpolation to make the field evenly spaced

Parameters

- **interp** (*{'linear', 'cubic', 'quintic'}, optional*) – The kind of spline interpolation to use. Default is 'linear'.
- **res** (*number*) – Resolution of the resulting field. A value of 1 meaning a spatial resolution equal to the smallest space along the two axis for the initial field.

mask

mask_as_sf

max

mean

median

min

mirroring(*direction*, *position*, *inds_to_mirror='all'*, *mir_coef=1*, *interp=None*, *value=0*, *inplace=False*)
Return a field with additional mirrored values.

Parameters

- **direction** (*string in ['x', 'y']*) – Axe on which place the symetry plane
- **position** (*number*) – Position of the symetry plane alogn the given axe
- **inds_to_mirror** (*integer*) – Number of vector rows to symetrize (default is all)
- **mir_coef** (*number, optional*) – Optional coefficient applied only to the mirrored values.
- **inplace** (*boolean, optional*) – .
- **interp** (*string, optional*) – If specified, method used to fill the gap near the symetry plane by interpoaltion. 'value' : fill with the given value, 'nearest' : fill with the nearest value, 'linear' (default): fill using linear interpolation (Delaunay triangulation), 'cubic' : fill using cubic interpolation (Delaunay triangulation)
- **value** (*array, optional*) – Value at the symetry plane, in case of interpolation

reduce_spatial_resolution(*fact*, *inplace=False*)
Reduce the spatial resolution of the field by a factor 'fact'

Parameters

- **fact** (*int*) – Reducing factor.
- **inplace** (*boolean, optional*) – .

rotate(*angle*, *inplace=False*)
Rotate the scalar field.

Parameters

- **angle** (*integer*) – Angle in degrees (positive for trigonometric direction). In order to preserve the orthogonal grid, only multiples of 90° are accepted (can be negative multiples).
- **inplace** (*boolean, optional*) – If 'True', scalar field is rotated in place, else, the function return a rotated field.

Returns **rotated_field** – Rotated scalar field.

Return type ScalarField object, optional

scale(*scalex=None*, *scaley=None*, *scalev=None*, *inplace=False*)
Scale the ScalarField.

Parameters

- **scaley, scalev** (*scalex,*) – Scale for the axis and the values
- **inplace** (*boolean*) – .

smooth(*tos='uniform'*, *size=None*, *inplace=False*, ***kw*)
Smooth the scalarfield in place. Warning : fill up the field (should be used carefully with masked field borders)

Parameters

- **tos** (*string, optional*) – Type of smoothing, can be 'uniform' (default) or 'gaussian' (See ndimage module documentation for more details)
- **size** (*number, optional*) – Size of the smoothing (is radius for 'uniform' and sigma for 'gaussian') in indice number. Default is 3 for 'uniform' and 1 for 'gaussian'.
- **inplace** (*boolean, optional*) – If True, Field is smoothed in place, else, the smoothed field is returned.
- **kw** (*dic*) – Additional parameters for ndimage methods (See ndimage documentation)

unit_values

values

5.2.1.4 VectorField class

class IMTreatment.core.vectorfield.**VectorField**
Bases: *IMTreatment.core.field.Field*

Class representing a vector field (2D field, with two components on each point).

change_unit(*axe*, *new_unit*)
Change the unit of an axe.

Parameters

- **axe** (*string*) – 'y' for changing the profile y axis unit 'x' for changing the profile x axis unit 'values' or changing values unit
- **new_unit** (*Unum.unit object or string*) – The new unit.

check_consistency()
Raise errors if the vectorfield show some incoherences.

comp_x

comp_x_as_sf

comp_y

comp_y_as_sf

copy()
Return a copy of the vectorfield.

crop(*intervx=None*, *intervy=None*, *ind=False*, *inplace=False*)
Crop the area in respect with given intervals.

Parameters

- **intervx** (*array, optional*) – interval wanted along x
- **intervy** (*array, optional*) – interval wanted along y
- **ind** (*boolean, optional*) – If 'True', intervals are understood as indices along axis. If 'False' (default), intervals are understood in axis units.
- **inplace** (*boolean, optional*) – If 'True', the field is croped in place.

crop_masked_border(*hard=False*, *inplace=False*)
Crop the masked border of the field in place.

Parameters **hard** (*boolean, optional*) – If 'True', partially masked border are croped as well.

display(*component=None*, *kind=None*, ***plotargs*)
Display something from the vector field. If component is not given, a quiver is displayed. If component is an integer, the coresponding component of the field is displayed.

Parameters

- **component** (*string, optional*) – Component to display, can be 'V', 'x', 'y', 'mask'
- **kind** (*string, optinnal*) – Scalar plots : if 'None': each datas are plotted (imshow), if 'contour': contours are ploted (contour), if 'contourf': filled contours are ploted (contourf). Vector plots : if 'quiver': quiver plot, if 'stream': streamlines.
- **plotargs** (*dict*) – Arguments passed to the function used to display the vector field.

Returns **fig** – Reference to the displayed figure

Return type figure reference

extend(*nmb_left=0*, *nmb_right=0*, *nmb_up=0*, *nmb_down=0*, *inplace=False*)
Add columns or lines of masked values at the vectorfield.

Parameters

- **nmb_****** (*integers*) – Number of lines/columns to add in each direction.
- **inplace** (*bool*) – If 'False', return a new extended field, if 'True', modify the field inplace.

Returns **Extended_field** – Extended field.

Return type Field object, optional

fill(*kind='linear', value=[0.0, 0.0], inplace=False, reduce_tri=True, crop=False*)
Fill the masked part of the array.

Parameters

- **kind** (*string, optional*) – Type of algorithm used to fill. 'value' : fill with the given value 'nearest' : fill with the nearest value 'linear' (default): fill using linear interpolation (Delaunay triangulation) 'cubic' : fill using cubic interpolation (Delaunay triangulation)
- **value** (*2x1 array of numbers*) – Values used to fill (for kind='value').
- **inplace** (*boolean, optional*) – If 'True', fill the sf.ScalarField in place. If 'False' (default), return a filled version of the field.
- **reduce_tri** (*boolean, optional*) – If 'True', treatment is used to reduce the triangulation effort (faster when a lot of masked values) If 'False', no treatment (faster when few masked values)
- **crop** (*boolean, optional*) – If 'True', TVF borders are croped before filling.

get_norm(*norm=2*, *normalized='perpoint'*)
Return the field norm

get_profile(*component*, *direction*, *position*, *ind=False*, *interp='linear'*)
Return a profile of the vector field component, at the given position (or at least at the nearest possible position). If position is an interval, the fonction return an average profile in this interval.

Parameters

- **component** (*string in ['vx', 'vy']*) – component to treat.
- **direction** (*string in ['x', 'y']*) – Direction along which we choose a position.
- **position** (*float or interval of float*) – Position or interval in which we want a profile.
- **ind** (*boolean, optional*) – If 'True', position is taken as an indice Else (default), position is in the field units.
- **interp** (*string in ['nearest', 'linear']*) – if 'nearest', get the profile at the nearest position on the grid, if 'linear', use linear interpolation to get the profile at the exact position

Returns

- **profile** (*Profile object*) – Asked profile.
- **cutposition** (*array or number*) – Final position or interval in which the profile has been taken.

get_props()
Print the VectorField main properties

get_value(*x*, *y*, *ind=False*, *unit=False*)
Return the vector field components on the point (x, y). If ind is true, x and y are indices, else, x and y are value on axes (interpolated if necessary).

import_from_arrays(*axe_x*, *axe_y*, *comp_x*, *comp_y*, *mask=False*, *unit_x=''*, *unit_y=''*, *unit_values=''*)
Set the vector field from a set of arrays.

Parameters

- **axe_x** (*array*) – Discretized axis value along x
- **axe_y** (*array*) – Discretized axis value along y

- **comp_x** (*array or masked array*) – Values of the x component at the discritized points
- **comp_y** (*array or masked array*) – Values of the y component at the discritized points
- **mask** (*array of boolean, optional*) – Mask on comp_x and comp_y
- **unit_x** (*string, optionnal*) – Unit for the values of axe_x
- **unit_y** (*string, optionnal*) – Unit for the values of axe_y
- **unit_values** (*string, optionnal*) – Unit for the field components.

magnitude
Return a scalar field with the velocity field magnitude.

magnitude_as_sf
Return a scalarfield with the velocity field magnitude.

make_evenly_spaced(*interp='linear'*, *res=1*)
Use interpolation to make the field evenly spaced

Parameters

- **interp** (*{'linear', 'cubic', 'quintic'}, optional*) – The kind of spline interpolation to use. Default is 'linear'.
- **res** (*number*) – Resolution of the resulting field. A value of 1 meaning a spatial resolution equal to the smallest space along the two axis for the initial field.

mask

mask_as_sf

max

min

mirroring(*direction, position, inds_to_mirror='all', mir_coef=1.0, inplace=False, interp=None, value=[0, 0]*)
Return a field with additional mirrored values.

Parameters

- **direction** (*string in ['x', 'y']*) – Axe on which place the symetry plane.
- **position** (*number*) – Position of the symetry plane along the given axe
- **inds_to_mirror** (*integer*) – Number of vector rows to symetrize (default is all)
- **mir_coef** (*number or 2x1 array, optional*) – Optional coefficient(s) applied only to the mirrored values. If ana array first value is for 'comp_x' and second one to 'comp_y'
- **inplace** (*boolean, optional*) – .
- **interp** (*string, optional*) – If specified, method used to fill the gap near the symetry plane by interpoaltion. 'value' : fill with the given value, 'nearest' : fill with the nearest value, 'linear' (default): fill using linear interpolation (Delaunay triangulation), 'cubic' : fill using cubic interpolation (Delaunay triangulation)
- **value** (*array, optional*) – Value at the symetry plane, in case of interpolation

reduce_spatial_resolution(*fact*, *inplace=False*)
Reduce the spatial resolution of the field by a factor 'fact'

Parameters

- **fact** (*int*) – Reducing factor.
- **inplace** (*boolean, optional*) – .

rotate (*angle*, *inplace=False*)
Rotate the vector field.

Parameters

- **angle** (*integer*) – Angle in degrees (positive for trigonometric direction). In order to preserve the orthogonal grid, only multiples of 90° are accepted (can be negative multiples).
- **inplace** (*boolean, optional*) – If 'True', vector field is rotated in place, else, the function return a rotated field.

Returns **rotated_field** – Rotated vector field.

Return type VectorField object, optional

scale (*scalex=None*, *scaley=None*, *scalev=None*, *inplace=False*)
Scale the VectorField.

Parameters

- **scaley, scalev** (*scalex,*) – Scale for the axis and the values.
- **inplace** (*boolean*) – .

smooth (*tos='uniform'*, *size=None*, *inplace=False*, ***kw*)
Smooth the vectorfield in place. Warning : fill up the field (should be used carefully with masked field borders)

Parameters

- **tos** (*string, optional*) – Type of smoothing, can be 'uniform' (default) or 'gaussian' (See ndimage module documentation for more details)
- **size** (*number, optional*) – Size of the smoothing (is radius for 'uniform' and sigma for 'gaussian'). Default is 3 for 'uniform' and 1 for 'gaussian'.
- **inplace** (*boolean, optional*) – .
- **kw** (*dic*) – Additional parameters for ndimage methods (See ndimage documentation)

theta
Return a scalar field with the vector angle (in reference of the unit_y vector [1, 0]).

Parameters **low_velocity_filter** (*number*) – If not zero, points where V < Vmax*low_velocity_filter are masked.

Returns **theta_sf** – Contening theta field.

Return type sf.ScalarField object

theta_as_sf
Return a scalarfield with the velocity field angles.

unit_values

5.2.1.5 TemporalScalarFields class

class IMTreatment.core.temporalscalarfields.**TemporalScalarFields**
Bases: *IMTreatment.core.temporalfields.TemporalFields*

Class representing a set of time-evolving scalar fields.

fill(*tof='spatial'*, *kind='linear'*, *value=0.0*, *inplace=False*, *crop=False*)
Fill the masked part of the array in place.

Parameters

- **tof** (*string*) – Can be 'temporal' for temporal interpolation, or 'spatial' for spatial interpolation.
- **kind** (*string, optional*) – Type of algorithm used to fill. 'value' : fill with a given value 'nearest' : fill with nearest available data 'linear' : fill using linear interpolation 'cubic' : fill using cubic interpolation
- **value** (*2x1 array*) – Value for filling, '[Vx, Vy]' (only usefull with tof='value')
- **inplace** (*boolean, optional*) – .
- **crop** (*boolean, optional*) – If 'True', TVF borders are croped before filling.

get_max_field(*nmb_min=1*)
Calculate the maximum scalar field, from all the fields.

Parameters **nmb_min** (*integer, optional*) – Minimum number of values used to take a maximum value. Else, the value is masked.

get_min_field(*nmb_min=1*)
Calculate the minimum scalar field, from all the fields.

Parameters **nmb_min** (*integer, optional*) – Minimum number of values used to take a minimum value. Else, the value is masked.

get_phase_map(*freq*, *tf=None*, *check_spec=None*, *verbose=True*)
Return the phase map of the temporal scalar field for the given frequency.

Parameters

- **freq** (*number*) – Wanted frequency
- **tf** (*Integer*) – Last time indice to use
- **check_spec** (*Integer*) – If not None, specify the number of spectrum to display (useful to check if choosen frequencies are relevant).
- **verbose** (*Boolean*) – .

Returns **phase_map** – .

Return type ScalarField object

get_spectral_filtering(*fmin*, *fmax*, *order=2*, *inplace=False*)
Perform a temporal spectral filtering

Parameters

- **fmax** (*fmin,*) – Minimal and maximal frequencies
- **order** (*integer, optional*) – Butterworth filter order

Returns **filt_tsf** – Filtered temporal field

Return type *TemporalScalarFields*

values

values_as_sf

5.2.1.6 TemporalVectorFields class

class IMTreatment.core.temporalvectorfields.**TemporalVectorFields**
Bases: *IMTreatment.core.temporalfields.TemporalFields*

Class representing a set of time-evolving velocity fields.

Vx

Vx_as_sf

Vy

Vy_as_sf

fill(*tof='spatial', kind='linear', value=[0.0, 0.0], inplace=False, crop=False*)
Fill the masked part of the array in place.

Parameters

- **tof** (*string*) – Can be 'temporal' for temporal interpolation, or 'spatial' for spatial interpolation.
- **kind** (*string, optional*) – Type of algorithm used to fill. 'value' : fill with a given value 'nearest' : fill with nearest available data 'linear' : fill using linear interpolation 'cubic' : fill using cubic interpolation
- **value** (*2x1 array*) – Value for filling, '[Vx, Vy]' (only usefull with tof='value')
- **inplace** (*boolean, optional*) – .
- **crop** (*boolean, optional*) – If 'True', TVF borders are croped before filling.

get_mean_kinetic_energy()
Calculate the mean kinetic energy.

get_mean_tke()

get_reynolds_stress(*nmb_val_min=1*)
Calculate the reynolds stress.

Returns **Re_xx, Re_yy, Re_xy** – Reynolds shear stress

Return type ScalarField objects

get_spectral_filtering(*fmin*, *fmax*, *order=2*, *inplace=False*)
Perform a temporal spectral filtering

Parameters

- **fmax** (*fmin,*) – Minimal and maximal frequencies
- **order** (*integer, optional*) – Butterworth filter order

Returns **filt_tvf** – Filtered temporal field

Return type *TemporalVectorFields*

get_time_auto_correlation()
Return auto correlation based on Vx and Vy.

get_tke()
Calculate the turbulent kinetic energy.

get_turbulent_intensity()
Calculate the turbulent intensity.

TI = sqrt(2/3*k)/sqrt(Vx**2 + Vy**2)

magnitude

magnitude_as_sf

theta

theta_as_sf

5.2.1.7 SpatialScalarFields class

class IMTreatment.core.spatialscalarfields.**SpatialScalarFields**
Bases: *IMTreatment.core.spatialfields.SpatialFields*

values_as_sf

5.2.1.8 SpatialVectorFields class

class IMTreatment.core.spatialvectorfields.**SpatialVectorFields**
Bases: *IMTreatment.core.spatialfields.SpatialFields*

Class representing a set of spatial-evolving velocity fields.

Vx_as_sf

Vy_as_sf

magnitude_as_sf

theta_as_sf

5.2.2 General modules

5.2.2.1 file_operation module

IMTreatment.file_operation.file_operation.**IM7_to_imt**(*im7_path*, *imt_path*, *kind='SF'*, *compressed=True*, ***kwargs*)

Transfome an IM7 (davis) file into a, imt exploitable file.

Parameters

- **im7_path** (*path to file or directory*) – Path to the IM7 file(s) , can be path to a single file or path to a directory contening multiples files.
- **imt_path** (*path to file or directory*) – Path where to save imt files, has to be the same type of path than 'im7_path' (path to file or path to directory)
- **kind** (*string*) – Kind of object to store (can be 'TSF' for TemporalScalarFields, 'SSF' for SpatialScalarFields or 'SF' for multiple ScalarField)
- **compressed** (*boolean, optional*) – If 'True' (default), the file is compressed using gzip.
- **kwargs** (*dict, optional*) – Additional arguments for 'import_from_***()'.

IMTreatment.file_operation.file_operation.**VC7_to_imt**(*vc7_path*, *imt_path*, *kind='VF'*, *compressed=True*, ***kwargs*)

Transfome an VC7 (davis) file into a, imt exploitable file.

Parameters

- **vc7_path** (*path to file or directory*) – Path to the VC7 file(s) , can be path to a single file or path to a directory contening multiples files.
- **imt_path** (*path to file*) – Path where to save imt file.
- **kind** (*string*) – Kind of object to store (can be 'TVF' for TemporalVectorFields, 'SVF' for SpatialVectorFields or 'VF' for multiple VectorField)
- **compressed** (*boolean, optional*) – If 'True' (default), the file is compressed using gzip.
- **kwargs** (*dict, optional*) – Additional arguments for 'import_from_***()'.

IMTreatment.file_operation.file_operation.**check_path**(*filepath*, *newfile=False*)

Normalize and check the validity of the given path to feed importation functions.

IMTreatment.file_operation.file_operation.**export_to_ascii**(*obj*, *filepath*)

IMTreatment.file_operation.file_operation.**export_to_file**(*obj*, *filepath*, *compressed=True*, ***kw*)

Write the object in the specified file. Additionnals arguments for the JSON encoder may be set with the **kw argument. If existing, specified file will be truncated. If not, it will be created.

Parameters

- **obj** – Object to store (common and IMT objects are supported).
- **filepath** (*string*) – Path specifiing where to save the object.
- **compressed** (*boolean, optional*) – If 'True' (default), the file is compressed using gzip.

IMTreatment.file_operation.file_operation.**export_to_matlab**(*obj*, *filepath*, ***kw*)

IMTreatment.file_operation.file_operation.**export_to_picture**(*SF*, *filepath*)

Export a scalar field to a picture file.

Parameters

- **SF** – .
- **filepath** (*string*) – Path to the picture file.

IMTreatment.file_operation.file_operation.**export_to_pictures**(*SFs*, *filepath*)

Export scalar fields to a picture file.

Parameters

- **SF** – .
- **filename** (*string*) – Path to the picture file. Should include a name for the image (without the extension).

IMTreatment.file_operation.file_operation.**export_to_video**(*SFs*, *filepath*, *fps=24*, *colormap=None*)

Export scalar fields to a video file.

Parameters

- **SF** – .

- **filename** (*string*) – Path to the video file.

IMTreatment.file_operation.file_operation.**export_to_vtk**(*obj*, *filepath*, *axis=None*, ***kw*)

Export the field to a .vtk file, for Mayavi use.

Parameters

- **filepath** (*string*) – Path where to write the vtk file.
- **axis** (*tuple of strings*) – By default, field axe are set to (x,y), if you want different axis, you have to specified them here. For example, "('z', 'y')", put the x field axis values in vtk z axis, and y field axis in y vtk axis.
- **line** (*boolean (only for Points object)*) – If 'True', lines between points are writen instead of points.

IMTreatment.file_operation.file_operation.**find_file_in_path**(*regs*, *dirpath*, *ask=False*)

Search recursively for a folder containing files matching a regular expression, in the given root folder.

Parameters

- **exts** (*list of string*) – List of regular expressions
- **dirpath** (*string*) – Root path to search from
- **ask** (*bool*) – If 'True', ask for the wanted folder and return only this one.

Returns **folders** – List of folder containings wanted files.

Return type list of string

IMTreatment.file_operation.file_operation.**import_from_IM7**(*filename*, *infos=False*)

Import a scalar field from a .IM7 file.

Parameters

- **filename** (*string*) – Path to the IM7 file.
- **infos** (*boolean, optional*) – If 'True', also return a dictionary with informations on the im7

IMTreatment.file_operation.file_operation.**import_from_IM7s**(*fieldspath*, *kind='TSF'*, *fieldnumbers=None*, *incr=1*)

Import scalar fields from .IM7 files. 'fieldspath' should be a tuple of path to im7 files. All im7 file present in the folder are imported.

Parameters

- **fieldspath** (*string or tuple of string*) –
- **kind** (*string, optional*) – Kind of object to create with IM7 files. (can be 'TSF' for TemporalScalarFields or 'SSF' for SpatialScalarFields).
- **fieldnumbers** (*2x1 tuple of int*) – Interval of fields to import, default is all.
- **incr** (*integer*) – Incrementation between fields to take. Default is 1, meaning all fields are taken.

IMTreatment.file_operation.file_operation.**import_from_VC7**(*filename*, *infos=False*, *add_fields=False*)

Import a vector field or a velocity field from a .VC7 file

Parameters

- **filename** (*string*) – Path to the file to import.
- **infos** (*boolean, optional*) – If 'True', also return a dictionary with informations on the im7
- **add_fields** (*boolean, optional*) – If 'True', also return a tuple containing additional fields contained in the vc7 field (peak ratio, correlation value, ...)

IMTreatment.file_operation.file_operation.**import_from_VC7s**(*fieldspath*, *kind='TVF'*, *fieldnumbers=None*, *incr=1*, *add_fields=False*, *verbose=False*)

Import velocity fields from .VC7 files. 'fieldspath' should be a tuple of path to vc7 files. All vc7 file present in the folder are imported.

Parameters

- **fieldspath** (*string or tuple of string*) – If no '.vc7' are found directly under 'fieldspath', present folders are recursively serached for '.vc7' files.
- **kind** (*string, optional*) – Kind of object to create with VC7 files. (can be 'TVF' or 'SVF').
- **fieldnumbers** (*2x1 tuple of int*) – Interval of fields to import, default is all.
- **incr** (*integer*) – Incrementation between fields to take. Default is 1, meaning all fields are taken.
- **add_fields** (*boolean, optional*) – If 'True', also return a tuple containing additional fields contained in the vc7 field (peak ratio, correlation value, ...).
- **Verbose** (*bool, optional*) – .

IMTreatment.file_operation.file_operation.**import_from_file**(*filepath*, ***kw*)

Load and return an object from the specified file using the JSON format. Additionnals arguments for the JSON decoder may be set with the **kw argument. Such as'encoding' (to change the file encoding, default='utf-8').

Parameters

- **filepath** (*string*) – Path specifiing the file to load.
- **full_import** (*boolean*) – If 'True', everything is charged in memory, else, data are loaded in memory when they are needed.

IMTreatment.file_operation.file_operation.**import_from_matlab**(*filepath*, *obj*, *show_struct=False*, ***kwargs*)

Import data from a matlab (.m) file.

Data should be a dictionnary.

Parameters

- **filepath** (*string*) – Path of the matlab file
- **obj** (*string in ['ScalarField', 'VectorField', 'Profile',)* – 'Points'] Kind of object to import to.
- **show_struct** (*boolean*) – If True, just show the structure of the file (and return it, so you can play with it)
- **kwargs** – Rest of the keyword arguments should indicate where to find the necessary information in the matlab dictionnary.

5.2.2.1.1 Example

With a matlab file containing a dictionnary with the following entries: ‘x’, ‘x_unit’, ‘y’, ‘y_unit’, ‘u’, ‘v’ >>> vf = import_from_matlab(‘data.m’, ‘VectorField’, ... axe_x=’x’, axe_y=’y’, ... unit_x=’x_unit’, unit_y=’y_unit’, ... comp_x=’u’, comp_y=’v’)

IMTreatment.file_operation.file_operation.**import_from_picture**(*filepath*, *axe_x=None*, *axe_y=None*, *unit_x=”*, *unit_y=”*, *unit_values=”*, *dtype=<class 'float'>*)

Import a scalar field from a picture file.

Parameters

- **filepath** (*string*) – Path to the picture file.
- **axe_x** – .
- **axe_y** – .
- **unit_x** – .
- **unit_y** – .
- **unit_values** – .
- **dtype** – Type of the stored values (default to float)

Returns

.

Return type tmp_sf

IMTreatment.file_operation.file_operation.**import_from_pictures**(*filepath*, *axe_x=None*, *axe_y=None*, *unit_x=”*, *unit_y=”*, *unit_values=”*, *times=None*, *unit_times=”*, *dtype=<class 'float'>*, *field-numbers=None*, *incr=1*, *ver-bose=False*)

Import scalar fields from a bunch of picture files.

Parameters

- **filepath** (*string*) – regex matching the files.
- **axe_x** – .
- **axe_y** – .
- **unit_x** – .

- **unit_y** – .
- **unit_values** – .
- **dtype** – Type of the stored values (default to float)
- **fieldnumbers** (*2x1 array*) – Interval of fields to import, default is all.
- **incr** (*integer*) – Increment (incr=2 will load only 1 picture over 2).

Returns

.

Return type tmp_sf

IMTreatment.file_operation.file_operation.**import_from_video**(*filepath*, *dx=1*, *dy=1*, *unit_x=''*, *unit_y=''*, *unit_values=''*, *dt=1*, *unit_times=''*, *dtype=<class 'float'>*, *frame_inds=None*, *incr=1*, *verbose=False*)

Import scalar fields from a video.

Parameters

- **filepath** (*string*) – regex matching the files.
- **dx** – .
- **dy** – .
- **unit_x** – .
- **unit_y** – .
- **unit_values** – .
- **dtype** – Type of the stored values (default to float)
- **frame_inds** (*2x1 array*) – Interval of fields to import, default is all.
- **incr** (*integer*) – Increment (incr=2 will load only 1 picture over 2).

Returns

.

Return type tmp_sf

IMTreatment.file_operation.file_operation.**import_profile_from_ascii**(*filepath*, *x_col=1*, *y_col=2*, *unit_x=1[]*, *unit_y=1[]*, ***kwargs*)

Import a Profile object from an ascii file.

Parameters

- **y_col** (*x_col,*) – Colonne numbers for the given variables (begining at 1).

- **unit_y** (*unit_x,*) – Unities for the given variables.
- ****kwargs** – Possibles additional parameters are the same as those used in the numpy function 'genfromtext()' : 'delimiter' to specify the delimiter between colonnes. 'skip_header' to specify the number of colonne to skip at file

  begining

IMTreatment.file_operation.file_operation.**import_pts_from_ascii**(*filepath*, *x_col=1*, *y_col=2*, *v_col=None*, *unit_x=1 []*, *unit_y=1 []*, *unit_v=1 []*, ***kwargs*)

Import a Points object from an ascii file.

Parameters

- **y_col, v_col** (*x_col,*) – Colonne numbers for the given variables (begining at 1).
- **unit_y, unit_v** (*unit_x,*) – Unities for the given variables.
- ****kwargs** – Possibles additional parameters are the same as those used in the numpy function 'genfromtext()' : 'delimiter' to specify the delimiter between colonnes. 'skip_header' to specify the number of colonne to skip at file

  begining

  ...

IMTreatment.file_operation.file_operation.**import_sf_from_ascii**(*filepath*, *x_col=1*, *y_col=2*, *vx_col=3*, *unit_x=1 []*, *unit_y=1 []*, *unit_values=1 []*, ***kwargs*)

Import a scalarfield from an ascii file.

Parameters

- **y_col, vx_col** (*x_col,*) – Colonne numbers for the given variables (begining at 1).
- **unit_y, unit_v** (*unit_x,*) – Unities for the given variables.
- ****kwargs** – Possibles additional parameters are the same as those used in the numpy function 'genfromtext()' : 'delimiter' to specify the delimiter between colonnes. 'skip_header' to specify the number of colonne to skip at file

  begining

  ...

IMTreatment.file_operation.file_operation.**import_vf_from_ascii**(*filepath*, *x_col=1*, *y_col=2*, *vx_col=3*, *vy_col=4*, *unit_x=1 []*, *unit_y=1 []*, *unit_values=1 []*, ***kwargs*)

Import a vectorfield from an ascii file.

Parameters

- **y_col, vx_col, vy_col** (*x_col,*) – Colonne numbers for the given variables (begining at 1).
- **unit_y, unit_v** (*unit_x,*) – Unities for the given variables.
- ****kwargs** – Possibles additional parameters are the same as those used in the numpy function 'genfromtext()' : 'delimiter' to specify the delimiter between colonnes. 'skip_header' to specify the number of colonne to skip at file

  begining

  ...

IMTreatment.file_operation.file_operation.**import_vfs_from_ascii**(*filepath*, *kind='TVF'*, *incr=1*, *interval=None*, *x_col=1*, *y_col=2*, *vx_col=3*, *vy_col=4*, *unit_x=1 []*, *unit_y=1 []*, *unit_values=1 []*, *times=[]*, *unit_time=1 []*, ***kwargs*)

Import velocityfields from an ascii file.

Parameters

- **filepath** (*string*) – Pathname pattern to the ascii files.
- **incr** (*integer, optional*) – Increment value between two fields taken.
- **interval** (*2x1 array, optional*) – Interval in which take fields.
- **y_col, vx_col, vy_col** (*x_col,*) – Colonne numbers for the given variables (begining at 1).
- **unit_y, unit_v** (*unit_x,*) – Unities for the given variables.
- **times** (*array of number, optional*) – Times of the instantaneous fields.
- **unit_time** (*Unit object, optional*) – Time unit, 'second' by default.
- ****kwargs** – Possibles additional parameters are the same as those used in the numpy function 'genfromtext()' : 'delimiter' to specify the delimiter between colonnes. 'skip_header' to specify the number of colonne to skip at file begining

Note: txt files are taken in alpha-numerical order ('file2.txt' is taken before 'file20.txt'). So you should name your files properly.

IMTreatment.file_operation.file_operation.**imts_to_imt**(*imts_path*, *imt_path*, *kind*)
Concatenate some .imt files to one .imt file.

Parameters

- **imts_path** (*string*) – Path to the .imt files
- **imt_path** (*string*) – Path to store the new imt file.
- **kind** (*string*) – Kind of object for the new imt file (can be 'TSF' for TemporalScalarFields, 'SSF' for SpatialScalarFields, 'TVF' for TemporalVectorFields, 'SVF' for SpatialVectorFields)

IMTreatment.file_operation.file_operation.**matlab_parser**(*obj*, *name*)

5.2.2.2 field_treatment module

IMTreatment.field_treatment.field_treatment.**extrapolate_until_wall**(*field*, *direction='x'*, *position=0.0*, *kind_interp='linear'*)
Interpolate the given fiels until it reach wall (0 velocity) at the given position.

Parameters

- **field** (*ScalarField or VectorField object.*) – Field to extend
- **direction** (*string*) – Axe on which the wall is placed (default to 'x')
- **position** (*number*) – position of the wall (in the same unit as the field)
- **kind_interp** (*string*) – Type of algorithm used to fill. 'linear' (default): fill using linear interpolation 'cubic' : fill using cubic interpolation

Returns **extended_field** – Extended field.

Return type *ScalarField* or *VectorField*

IMTreatment.field_treatment.field_treatment.**get_Kenwright_field**(*field*, *raw=False*)
Return a field with the vector product between the velocity field and the eigen vectors of the velocity jacobian. Values of 0 on these field should represent separation lines or re-attachment lines (See Kenwright et al (1998)).

Parameters

- **field** (*VectorField*) – .
- **raw** (*bool, optional*) – If 'False' (default), ScalarFields are returned If 'True', arrays are returned

Returns

- **K1_field** (*ScalarField*) – Vector product with the principal eigen vector
- **K2_field** (*ScalarField*) – Vector product with the oher eigen vector

IMTreatment.field_treatment.field_treatment.**get_divergence**(*vf*)
Return the divergence.

IMTreatment.field_treatment.field_treatment.**get_fieldlines**(*VF*, *xys*, *reverse=False*, *rel_err=1e-08*, *max_steps=1000*, *resolution=0.25*, *resolution_kind='length'*, *boundary_tr='stop'*)
Return the field lines associated to a set of particules initialy at the positions xys. Use Fortran integration of the VODE algorithm. (Source: http://www.netlib.org/ode/vode.f)

Parameters

- **VF** (*VectorField or velocityField object*) – Field on which compute the fieldlines
- **xys** (*2xN tuple of number*) – Initial points positions
- **rel_err** (*number*) – relative maximum error for rk4 algorithm
- **max_steps** (*integer*) – Maximum number of steps (default = 1000).
- **resolution** (*number,*) – resolution of the resulting fieldline (do not impact accuracy).
- **resolution_kind** (*string in ['time', 'length']*) – if 'time', returned points time space are homogeneous, if 'length', returned points space interval are homogeneous.
- **boundary_tr** (*string*) – Method to treat the field boundaries. If 'stop', fieldlines are stopped when encountering a boundary, If 'hide', fieldlines that encounter a boundary are not returned.

IMTreatment.field_treatment.field_treatment.**get_grad_field**(*field*, *direction='x'*)
Return a field based on original field gradients. (Vx = dV/dx, Vy = DV/Vy)

Parameters

- **field** (*VectorField object*) – .
- **direction** (*string in ['x', 'y']*) – if 'x', return Vx gradients if 'y', return Vy gradients.

Returns **gfield** – .

Return type VectorField object

IMTreatment.field_treatment.field_treatment.**get_gradients**(*field*, *raw=False*)
Return gradients along x and y.

(Obtained arrays corespond to components of the Jacobian matrix)

Parameters

- **vf** (*VelocityField, ScalarField or Profile object*) – Field/profile to compute gradient from.
- **raw** (*boolean*) – If 'False' (default), ScalarFields objects are returned. If 'True', arrays are returned.

Returns **grad** – For VectorField input : (dVx/dx, dVx/dy, dVy/dx, dVy/dy), for ScalarField input : (dV/dx, dV/dy). for Profile input : dy/dx.

Return type tuple of ScalarField or arrays or *Profile*

IMTreatment.field_treatment.field_treatment.**get_jacobian_eigenproperties**(*field*, *raw=False*, *eig_val=True*, *eig_vect=True*)

Return eigenvalues and eigenvectors of the jacobian matrix on all the field.

Parameters

- **field** (*VectorField object*) – .
- **raw** (*boolean*) – If 'False' (default), ScalarFields objects are returned. If 'True', arrays are returned.
- **eig_val** (*boolean, optional*) – If 'True', eigenvalues are returned.
- **eig_vect** (*boolean, optional*) – If 'True', eigenvectors are returned.

Returns

- **eig_val1_re** (*ScalarField object, or array*) – Real part of the first eigenvalue.
- **eig_val1_im** (*ScalarField object, or array*) – Imaginary part of the first eigenvalue.
- **eig_val2_re** (*ScalarField object, or array*) – Real part of the second eigenvalue.
- **eig_val2_im** (*ScalarField object, or array*) – Imaginary part of the second eigenvalue.
- **eig1_vf** (*VectorField object, or tuple of arrays*) – Eigenvector associated with first eigenvalue.
- **eig2_vf** (*VectorField object, or tuple of arrays*) – Eigenvector associated with second eigenvalue.

IMTreatment.field_treatment.field_treatment.**get_shear_stress**(*vf*, *raw=False*)

Return a vector field with the shear stress.

Parameters

- **vf** (*VectorField or Velocityfield*) – Field on which compute shear stress
- **raw** (*boolean, optional*) – If 'True', return two arrays, if 'False' (default), return a VectorField object.

IMTreatment.field_treatment.field_treatment.**get_streamlines**(*VF*, *xys*, *reverse=False*, *rel_err=1e-08*, *max_steps=1000*, *resolution=0.25*, *resolution_kind='length'*, *boundary_tr='stop'*)

Return the lagrangien displacement of a set of particules initialy at the positions xys. Use Fortran integration of the VODE algorithm. (Source: http://www.netlib.org/ode/vode.f)

Parameters

- **VF** (*VectorField or velocityField object*) – Field on which compute the streamlines
- **xys** (*2xN tuple of number*) – Initial points positions
- **rel_err** (*number*) – relative maximum error for rk4 algorithm
- **max_steps** (*integer*) – Maximum number of steps (default = 1000).

- **resolution** (*number,*) – resolution of the resulting streamline (do not impact accuracy).
- **resolution_kind** (*string in ['time', 'length']*) – if 'time', returned points time space are homogeneous, if 'length', returned points space interval are homogeneous.
- **boundary_tr** (*string*) – Method to treat the field boundaries. If 'stop', streamlines are stopped when encountering a boundary, If 'hide', streamlines that encounter a boundary are not returned.

IMTreatment.field_treatment.field_treatment.**get_streamlines_fast**(*vf*, *xy*, *delta=0.25*, *interp='linear'*, *reverse=False*)

Return a tuples of Points object representing the streamline begining at the points specified in xy. Is called fast because use simpler integration method than the 'get_streamlines()' method and can be tuned with the 'delta' and 'interp' parameters.

Parameters

- **vf** (*VectorField or velocityField object*) – Field on which compute the streamlines
- **xy** (*tuple*) – Tuple containing each starting point for streamline.
- **delta** (*number, optional*) – Spatial discretization of the stream lines, relative to a the spatial discretization of the field.
- **interp** (*string, optional*) – Used interpolation for streamline computation. Can be 'linear'(default) or 'cubic'
- **reverse** (*boolean, optional*) – If True, the streamline goes upstream.

Returns **streams** – Each Points object represent a streamline

Return type tuple of Points object

IMTreatment.field_treatment.field_treatment.**get_swirling_vector**(*vf*, *raw=False*)

Return a vector field with the swirling vectors (eigenvectors of the velocity laplacian matrix ponderated by eigenvalues) (Have to be adjusted : which part of eigenvalues and eigen vectors take ?)

Parameters

- **vf** (*VectorField or Velocityfield*) – Field on which compute shear stress
- **raw** (*boolean, optional*) – If 'True', return an arrays, if 'False' (default), return a ScalarField object.

IMTreatment.field_treatment.field_treatment.**get_track_field**(*vf*)

Return the track field. (Vx, Vy) => (-Vy, Vx)

IMTreatment.field_treatment.field_treatment.**get_tracklines**(*VF*, *xys*, *reverse=False*, *rel_err=1e-08*, *max_steps=1000*, *resolution=0.25*, *resolution_kind='length'*, *boundary_tr='stop'*)

Return a tuples of Points object representing the tracklines begining at the points specified in xy. Warning : fill

the field before computing streamlines, can give bad results if the field have a lot of masked values.

Parameters

- **VF** (*VectorField or velocityField object*) – Field on which compute the streamlines
- **xys** (*2xN tuple of number*) – Initial points positions
- **rel_err** (*number*) – relative maximum error for rk4 algorithm
- **max_steps** (*integer*) – Maximum number of steps (default = 1000).
- **resolution** (*number,*) – resolution of the resulting streamline (do not impact accuracy).
- **resolution_kind** (*string in ['time', 'length']*) – if 'time', returned points time space are homogeneous, if 'length', returned points space interval are homogeneous.
- **boundary_tr** (*string*) – Method to treat the field boundaries. If 'stop', streamlines are stopped when encountering a boundary, If 'hide', streamlines that encounter a boundary are not returned.

Returns **tracks** – Each Points object represent a trackline

Return type tuple of Points object

IMTreatment.field_treatment.field_treatment.**reconstruct_from_gradients**(*field_dx*, *field_dy*, *field2_dx=None*, *field2_dy=None*, *ols=False*, *maxiter=10*)

Reconstruct a field with the gradients of this field.

Parameters

- **field_dy** (*field2_dx,*) – Gradients along x and y.
- **field_dy** – Gradients for the second component, if the reconstructed field is a vector field.
- **ols** (*boolean, optional*) – If 'True', ordinary least square is used to get a more precise result (can be quite long). If 'False' (default), a simple reconstruction based on taylor developement is used.
- **maxiter** (*integer, optional*) – Maximum number of iteration for the ols solver (default: 10) (more mean accurate results but slower computation).

Returns **rec_field** – Reconstructed field.

Return type *ScalarField* or VectorField object

5.2.2.2.1 Notes

Given result can only be relative values (because of the information lost while derivating). The returned result are normalized so that the mean of the field is egal to zero.

5.2.2.3 plotlib module

class IMTreatment.plotlib.plotlib.**ButtonManager**(*displayers*, *xlabel=''*, *ylabel=''*, *sharecb=True*, *normcb=None*, *play_interval=2*)

Bases: object

activate_buttons()

add_displayers(*displayers*)

close()

deactivate_buttons()

goto()

goto_beg(*event*)

goto_end(*event*)

goto_lims()

keyf(*event*)

link_to_other_graph(*graph*)

nextf(*event*)

playf(*event*)

prevf(*event*)

save()

save_animation(*animpath*, *fields='all'*, *writer='ffmpeg'*, *fps=24*, *title=''*, *artist='IMTreatment'*, *comment=''*, *bitrate=-1*, *codec='ffv1'*, *dpi=150*)

Save the button manager displays as an animation.

Parameters

- **animpath** (*string*) – Path where to save animation
- **fields** (*string or 2x1 list of numbers*) – Fields interval to save. Default is 'all' for all the fields.
- **writer** (*string*) – Name of the writer to use (available writers are listed in 'matplotlib.animation.writers.list()'
- **codec** (*string*) – One of the codec of the choosen writer (default to 'ffv1')
- **fps** (*integer*) – Number of frame per second (default to 24)
- **bitrate** (*integer*) – Video bitrate in kb/s (default to -1) Set this to -1 for letting the writter choose.
- **dpi** (*integer*) – dpi of the video images before compression (default to 150)
- **artist, comment** (*title,*) – Information added to the file metadata

set_lims(*lim1=None*, *lim2=None*)

slid(*event*)

timer_play()

update()

class IMTreatment.plotlib.plotlib.**DataCursorPoints**(*ax*, *tolerance=5*, *offsets=(-20, 20)*, *formatter=None*, *display_all=False*, *color=(0.76, 0.86, 0.92)*)

Bases: object

annotate(*ax*)

Draws and hides the annotation box for the given axis "ax".

get_color_from_event(*event*)

snap(*x*, *y*)

Return the value in self._points closest to (x, y).

class IMTreatment.plotlib.plotlib.**DataCursorTextDisplayer**(*displayer*, *i=None*, *precision=3*)

Bases: object

class IMTreatment.plotlib.plotlib.**Displayer**(*x*, *y*, *values=None*, *data_type=None*, *sharebds=True*, *buffer_size=100*, ***kwargs*)

Bases: object

draw(*i=None*, *ax=None*, *cb=False*, *remove_current=False*, *rescale=True*)

draw_multiple(*inds*, *sharecb=False*, *sharex=False*, *sharey=False*, *ncol=None*, *nrow=None*)

draw_new(*i=None*, *ax=None*, *cb=False*, *rescale=True*)

field_1D_default_args = {'aspect': 'equal', 'interpolation': 'nearest', 'kind': 'ma

field_2D_default_args = {'aspect': 'equal', 'kind': 'quiver'}

get_data(*i=None*)

get_data_at_point(*x*, *y*, *i=None*)

get_global_norm()

get_norm(*i*)

points_default_args = {'kind': 'scatter'}

profile_default_args = {'kind': 'plot'}

class IMTreatment.plotlib.plotlib.**Formatter**(*order=0*, *fformat='%1.1f'*, *offset=True*, *mathtext=True*)

Bases: matplotlib.ticker.ScalarFormatter

IMTreatment.plotlib.plotlib.**annotate_multiple**(*s*, *xy*, *xytext*, **args*, ***kwargs*)

IMTreatment.plotlib.plotlib.**colored_plot**(*x*, *y*, *z=None*, *log='plot'*, *min_colors=1000*, *colorbar=False*, *color_label=''*, ***kwargs*)

Plot a colored line with coordinates x and y

Parameters

- **y** (*x,*) – coordinates of each points
- **z** (*nx1 array of number, optional*) – values for the color
- **log** (*string, optional*) – Type of axis, can be 'plot' (default), 'semilogx', 'semilogy', 'loglog'
- **min_colors** (*integer, optional*) – Minimal number of different colors in the plot (default to 1000).
- **colorbar** (*bool*) – .

- **color_label** (*string, optional*) – Colorbar label if color is an array.
- **kwargs** (*dict, optional*) – list of arguments to pass to the common plot (see matplotlib documentation).

IMTreatment.plotlib.plotlib.**get_color_cycles**()

IMTreatment.plotlib.plotlib.**get_color_gradient**(*cmap='jet'*, *number=10*)
Return a gradient of color for plot uses.

IMTreatment.plotlib.plotlib.**is_sorted**(*data*)

IMTreatment.plotlib.plotlib.**make_cmap**(*colors*, *position=None*, *name='my_cmap'*)
Return a color map cnstructed with the geiven colors and positions.

Parameters

- **colors** (*Nx1 list of 3x1 tuple*) – Each color wanted on the colormap. each value must be between 0 and 1.
- **positions** (*Nx1 list of number, optional*) – Relative position of each color on the colorbar. default is an uniform repartition of the given colors.
- **name** (*string, optional*) – Name for the color map

IMTreatment.plotlib.plotlib.**make_discrete_cmap**(*interv_centers*, *cmap=None*)
Create a discrete map, with intervals centered on the given values.

IMTreatment.plotlib.plotlib.**make_segments**(*x*, *y*)
Create list of line segments from x and y coordinates, in the correct format for LineCollection: an array of the form numlines x (points per line) x 2 (x and y) array

IMTreatment.plotlib.plotlib.**mark_axe**(*txt*, *ax=None*, *loc=2*, *pad=0.3*, *borderpad=0.0*, *font_props=None*, *frameon=True*)

IMTreatment.plotlib.plotlib.**save_animation**(*animpath*, *fig=None*, *fields='all'*, *writer='ffmpeg'*, *fps=24*, *title=''*, *artist='IMTreatment'*, *comment=''*, *bitrate=-1*, *codec='ffv1'*, *dpi=150*)
Save the current button manager displays as an animation.

Parameters

- **animpath** (*string*) – Path where to save animation
- **fig** (*Figure instance*) – Figure to save the animation from (if None, get the current one)
- **fields** (*string or 2x1 list of numbers*) – Fields interval to save. Default is 'all' for all the fields.
- **writer** (*string*) – Name of the writer to use (available writers are listed in 'matplotlib.animation.writers.list()'
- **codec** (*string*) – One of the codec of the choosen writer (default to 'ffv1')
- **fps** (*integer*) – Number of frame per second (default to 24)
- **bitrate** (*integer*) – Video bitrate in kb/s (default to -1) Set this to -1 for letting the writter choose.
- **dpi** (*integer*) – dpi of the video images before compression (default to 150)
- **artist, comment** (*title,*) – Information added to the file metadata

IMTreatment.plotlib.plotlib.**use_perso_style**()
Change matplotlib default style to something nicer

5.2.2.4 pod module

class IMTreatment.pod.pod.**ModalFields**(*decomp_type*, *mean_field*, *modes*, *modes_numbers*, *temporal_evolutions*, *eigvals=None*, *eigvects=None*, *ritz_vals=None*, *mode_norms=None*, *growth_rate=None*, *pulsation=None*)
Bases: *IMTreatment.core.field.Field*

Class representing the result of a modal decomposition.

augment_temporal_resolution(*fact=2*, *interp='linear'*, *inplace=True*, *verbose=False*)
Augment the temporal resolution (to augmente the temporal resolution of the reconstructed fields).

Parameters

- **fact** (*integer*) – Resolution augmentation needed (default is '2', for a result profile with twice more points)
- **interp** (*string in ['linear', 'nearest', 'slinear', 'quadratic',]*) – 'cubic'] Specifies the kind of interpolation as a string (Default is 'linear'). slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order.
- **inplace** (*bool*) – .
- **verbose** (*bool*) – .

crop(*intervx=None*, *intervy=None*, *intervt=None*, *ind=False*, *inplace=False*)
Crop the POD modes and/or temporal evolutions, according to the given intervals.

Parameters

- **intervx** (*2x1 array of numbers*) – Interval of x to keep.
- **intervy** (*2x1 array of numbers*) – Interval of y to keep.
- **intervt** (*2x1 array of numbers*) – Interval of time to keep.
- **ind** (*boolean, optional*) – If 'True', intervals are understood as indices along axis. If 'False' (default), intervals are understood in axis units.
- **inplace** (*boolean, optional*) – If 'True', the field is croped in place.

crop_modal_base(*modes_to_keep=None*, *modes_to_remove=None*, *inplace=True*)
Remove some modes from the modal base.

Parameters

- **modes_to_keep** (*array of integers or integer*) – If an integer, the first N modes are kept, if an array of indices, all the associated modes are conserved.
- **modes_to_remove** (*array of integers or integer*) – If an integer, the last N modes are removed, if an array of indices, all the associated modes are removed.

display()

display_recap(*figsize=(15, 10)*)
Display some important diagram for the decomposition.

get_critical_kappa(*Nx*, *Ny=None*)
Return the critical value of kappa. A mode with a kappa value superior to the critical value have only .3% chance to be a random mode.

get_modes_energy(*cum=False*, *raw=False*)
Return a profile whith the modes mean energy.

Parameters

- **cum** (*boolean, optional*) – If 'False' (default), return the modes energy If 'True', return the cumulative modes energy
- **raw** (*bool, optional*) – If 'False' (default), a Profile object is returned If 'True', an array is returned

Returns **modes_nrj** – Energy for each modes.

Return type array or Profile object

get_spatial_coherence(*raw=False*)
Return a profile where each value represent the probability for a mode to be spatialy non-random (values from 0 to 1). Can be used to determine the modes to take to filter the turbulence (and so perform a tri-decomposition (mean + coherent + turbulent))

Parameters **raw** (*bool, optional*) – If 'False' (default), a Profile object is returned If 'True', an array is returned

Returns **var_spec** – Probability estimation for each mode of being coherent in space.

Return type array or Profile object

5.2.2.4.1 Notes

Returned values are, for each modes, the variance of the normalized two-dimensional spectrum of Vx and Vy. Variance is high when spectrum show predominant frequencies (coherent behavior), inversely, variance is low when spectrum is nearly uniform (random behavior).

get_temporal_coherence(*raw=False*)
Return a profile where each value represent the probability for a mode to be temporaly non-random (values above 0 have only 5% chance to be associated with random modes). Can be used to determine the modes to take to filter the turbulence (and so perform a tri-decomposition (mean + coherent + turbulent))

Parameters **raw** (*bool, optional*) – If 'False' (default), a Profile object is returned If 'True', an array is returned

Returns **var_spec** – Probability estimation for each mode of being coherent in time.

Return type array or Profile object

5.2.2.4.2 Notes

Returned values are, for each modes, the variance of the normalized spectrum of the temporal evolution. Variance is high when spectrum show predominant frequencies (coherent behavior), inversely, variance is low when spectrum is nearly uniform (random behavior).

modes_as_tf

reconstruct(*wanted_modes='all'*, *times=None*)
Recontruct fields resolved in time from modes.

Parameters

- **wanted_modes** (*string or number or array of numbers, optional*) – wanted modes for reconstruction : If 'all' (default), all modes are used If an array of integers, the wanted modes are used If an integer, the wanted first modes are used.
- **times** (*tuple of numbers*) – If specified, reconstruction is computed on the wanted times, else, times used for decomposition are used.

Returns **TF** – Reconstructed fields.

Return type *TemporalFields* (*TemporalScalarFields* or *TemporalVectorFields*)

scale(*scalex=1.0*, *scaley=1.0*, *scalet=1.0*, *scalev=1.0*, *inplace=False*)

smooth_spatial_evolutions(*tos='uniform'*, *size=None*, *inplace=True*)
Smooth the spatial evolutions (to do before reconstructing)

Parameters

- **tos** (*string, optional*) – Type of smoothing, can be 'uniform' (default) or 'gaussian' (See ndimage module documentation for more details)
- **size** (*number, optional*) – Size of the smoothing (is radius for 'uniform' and sigma for 'gaussian'). Default is 3 for 'uniform' and 1 for 'gaussian'.

smooth_temporal_evolutions(*tos='uniform'*, *size=None*, *inplace=True*)
Smooth the temporal evolutions (to do before reconstructing)

Parameters

- **tos** (*string, optional*) – Type of smoothing, can be 'uniform' (default) or 'gaussian' (See ndimage module documentation for more details)
- **size** (*number, optional*) – Size of the smoothing (is radius for 'uniform' and sigma for 'gaussian'). Default is 3 for 'uniform' and 1 for 'gaussian'.

IMTreatment.pod.pod.**modal_decomposition**(*obj*, *kind='pod'*, *obj2=None*, *wanted_modes='all'*, *max_vecs_per_node=1000*, *verbose=True*)
Compute POD modes of the given fields using the snapshot method.

Parameters

- **obj** (*TemporalFields, Profile or Points object*) – Fields to extract modes from
- **obj2** (*same as obj*) – Only used as second dataset for BPOD
- **kind** (*string, optional*) – Kind of decomposition, can be 'pod' (default), 'bpod' or 'dmd'.
- **wanted_modes** (*string or number or array of numbers*) – If 'all', extract all modes, If a number, extract first modes, If an array, extract the associated modes.
- **max_vecs_per_node** (*integer, optional*) – Number of fields that can be charged in memory. (More is faster but can lead to MemoryError)
- **verbose** (*boolean, optional*) – If 'True', display information.

Returns **modal_field** – .

Return type ModalField object

5.2.2.4.3 Notes

You can use partially masked fields as input. If so, the asked values are lineary interpolated before doing the decomposition.

5.2.2.5 utils package

Utlities used by IMTreatment modules

5.2.2.5.1 utils.codeinteraction module

class IMTreatment.utils.codeinteraction.**RemoveFortranOutput**
Bases: object

Context object to remove Fortran output.

to be used with 'with' statement.

5.2.2.5.2 Examples

```
>>> with RemoveFortranOutput():
>>>     # put some fortran functions here
```

5.2.2.5.3 utils.files module

class IMTreatment.utils.files.**Files**
Bases: object

add_file(*path*)

build_tree()
Build a file tree

copy()

delete_existing_files(*recursive=False*)

get_tree_representation(*max_file_list=10*, *hide_top=False*)

load_files_from_regex(*rootpath*, *regex*, *load_files=True*, *load_dirs=True*, *depth='all'*)
Load existing files from a regular expression.

remove_empty_directories()

remove_files(*arg*)
Remove some files from the files set.

Parameters **arg** (*integer, regex or array of integer or regex*) – If integer, remove the associated path, if a regex, remove the paths that match, if an array, delete paths for each element.

IMTreatment.utils.files.**remove_files_in_dirs**(*rootpath*, *dir_regex*, *file_regex*, *depth='all'*, *remove_dir=False*, *remove_files=True*)
make a recursive search for directories satisfying "dir_regex' from "rootpath", and remove all the files satisfying 'file_regex' in it.

Parameters

- **rootpath** (*string*) – Path where to begin searching.
- **dir_regex** (*string*) – Regular expression matching the directory where we want to remove stuff.
- **file_regex** (*string*) – Regular expression matching the files we want to delete.
- **depth** (*integer or 'all'*) – Number of directory layer to go through.
- **remove_dir** –

5.2.2.5.4 utils.multithreading module

class IMTreatment.utils.multithreading.**MultiThreading**(*funct*, *data*, *threads='all'*)

Bases: object

add_progress_counter(*init_mess='Beginning'*, *end_mess='Done'*, *name_things='things'*, *perc_interv=5*)

run()

5.2.2.5.5 utils.progresscounter module

class IMTreatment.utils.progresscounter.**ProgressCounter**(*init_mess*, *nmb_max*, *end_mess=None*, *name_things='things'*, *perc_interv=5*, *pgbar_len=15*)

Bases: object

Declare wherever you want and execute 'print_progress' at the begining of each loop.

print_progress()

start_chrono()

5.2.2.5.6 utils.types module

class IMTreatment.utils.types.**ReturnTest**(*typ*)

Bases: object

decorator(*function*)

enabled = True

class IMTreatment.utils.types.**TypeTest**(**arg_types*, ***kwargs_types*)

Bases: object

decorator(*function*)

enabled = True

extract_var_info(*function*)

5.2.2.5.7 utils.units module

IMTreatment.utils.units.**make_unit**(*string*)
Function helping for the creation of units. For more details, see the Unum module documentation.

Parameters **string** (*string*) – String representing some units.

Returns **unit** – unum object representing the given units

Return type unum.Unum object

5.2.2.5.8 Examples

```
>>> make_unit("m/s")
1 [m/s]
>>> make_unit("N/m/s**3")
1 [kg/s4]
```

IMTreatment.utils.units.**make_unit_old**(*string*)
Function helping for the creation of units. For more details, see the Unum module documentation.

Parameters **string** (*string*) – String representing some units.

Returns **unit** – unum object representing the given units

Return type unum.Unum object

5.2.2.5.9 Examples

```
>>> make_unit("m/s")
1 [m/s]
>>> make_unit("N/m/s**3")
1 [kg/s4]
```

5.2.3 Flow specific modules

5.2.3.1 boundary_layer module

class IMTreatment.boundary_layer.boundary_layer.**BlasiusBL**(*Uinf*, *nu*, *rho*)
Bases: object

Class representing a Blasius-like boundary layer.

Parameters

- **Uinf** (*number*) – Flown velocity away from the wall (m/s).
- **nu** (*number*) – Kinematic viscosity (m^2/s).
- **rho** (*number*) – Density (kg/m^3)

get_BL_properties(*x*, *allTurbulent=False*, *bl_perc='99'*)
Return the boundary layer properties according to blasius theory.

Parameters

- **x** (*number or array of number*) – Position where the boundary layer thickness is computed (m)
- **be a list)** (*(can*) –
- **allTurbulent** (*bool, optional*) – if True, the all boundary layer is considered turbulent.
- **bl_perc** (*string in ['95', '99']*) – Percentage of maximum velocity defining the BL thickness

Returns

- **delta** (*array of numbers*) – Boundary layer thickness profile (m)
- **delta2** (*array of numbers*) – Boundary layer momentum thickness (m)
- **delta_star** (*array of numbers*) – Boundary layer displacement thickness (m)
- **H** (*array of numbers*) – Shape factor
- **Cf** (*array of numbers*) – Friction coefficient profile
- **Rex** (*array of numbers*) – Reynolds number on the distance from the border.
- **tau_w** (*array of numbers*) – Wall shear stress (Pa)

get_Rex(*x*)
Return the Reynolds number based on the distance from the beginning of the plate.

get_profile(*x*, *y=None*, *allTurbulent=False*)
Return a Blasius-like (laminar) profile at the given position.

Parameters

- **x** (*number*) – Position of the profile along x axis
- **y** (*array of numbers*) – Point along y where to compute the profile (if not specified, 200 homogeneously placed points are used)
- **allTurbulent** (*bool, optional*) – if True, the boundary layer is considered turbulent.

Returns **prof** – Wanted Blasius-like profile.

Return type Profile Object

get_profile_with_confinement(*x*, *h*, *y=None*, *allTurbulent=False*)
Return a Blasius-like (laminar) profile at the given position, ajusted for confined BL.

Parameters

- **x** (*number*) – Position of the profile along x axis
- **h** (*number*) – Pater level.
- **y** (*array of numbers*) – Point along y where to compute the profile (if not specified, 200 homogeneously placed points are used)
- **allTurbulent** (*bool, optional*) – if True, the boundary layer is considered turbulent.

Returns **prof** – Wanted Blasius-like profile.

Return type Profile Object

get_thickness_with_confinement(*x*, *h*, *allTurbulent=False*)
Return the boundary layer thickness and the friction coefficient according to blasius theory and adapted for use with low water levels. (Only valid in laminar BL) Fonction ——— delta, Cf = BlasiusBL(allTurbulent=False)

Parameters

- **x** (*number or array of number*) – Position where the boundary layer thickness is computed (m) (can be a list).
- **h** (*number*) – Water depth (m).
- **allTurbulent** (*bool, optional*) – if True, the all boundary layer is considered turbulent.

Returns **delta** – Boundary layer thickness profile (m)

Return type Profile object

get_x_from_delta(*delta*, *allTurbulent=False*)
Return a the x value that give the wanted delta.

class IMTreatment.boundary_layer.boundary_layer.**FalknerSkanBL**(*nu*, *m*, *c0*, *L*)
Bases: object

get_f_function(*relerr=1e-05*, *max_it=1000*, *verbose=False*)
Return the 'f' function appearing in the Falkner-skan equation, and its derivatives.

5.2.3.1.1 Notes

- Falkner-Skan equation :

$$f''' + ff'' + \beta\left(1 - f'^2\right) = 0$$
$$f(0) = f'(0) = 0 \text{ and } f'(\infty) = 1$$

- Associated constants :

$$\beta = \frac{2m}{m+1}$$
$$-0.0905 \leq m \leq 2$$

- Remark on numerical resolution :

  The Falkner-Skan equation is a ODE system with BVP (boundary value problem). Classical ODE algorithm such as Runge-Kutta or Vode cannot take care of the $f'(\infty) = 0$. This ODE system is so solved with a shooting method.

get_mu(*x*, *y*)
return the mu parameter at the position '(x, y)'.

get_profile(*x*, *relerr=1e-05*, *max_it=1000*, *verbose=False*)
return the velocity profile at 'x' position.

get_u_e(*x*)
Return the external velocity at position 'x'.

get_y(*x*, *mu*)
Return the 'y' vaue associated with mu parameter.

class IMTreatment.boundary_layer.boundary_layer.**ThwaitesBL**(*u_e*, *nu=1e-06*)

Bases: object

get_BL_properties(*x*)

Return the boundary layer properties for given values of x

Parameters **x** (*number or array of numbers*) – Position along the flat plan until where we want to solve the momentum equation. (has no influence on the resoluion accuracy)

Returns

- **lambda** (*array of numbers*) – Dimensionless pressure gradient. (According to Kays and Crawford, a value of -0.082 is caracteristic of the separation phenomenon.)
- **delta2** (*array of numbers*) – Boundary layer momentum thickness
- **delta_star** (*array of numbers*) – Boundary layer displacement thickness
- **H** (*array of numbers*) – H factor

get_momentum_thikness(*x*)

Return the evolution of the boundary layer momentum thikness.

Parameters **x** (*number or array of numbers*) – Position along the flat plan until where we want to solve the momentum equation. (has no influence on the resoluion accuracy)

Returns **delta2**

Return type position and value of momentum thicknesses

u_e_pow(*x*)

Function u_e(x)**4.68 (used int numerical resolution)

class IMTreatment.boundary_layer.boundary_layer.**WakeLaw**(*h*, *tau*, *delta*, *visc_c=1e-06*, *rho=1000*)

Bases: *IMTreatment.boundary_layer.boundary_layer.WallLaw*

Class representing a law of the wake profile using Coles theory. By default, the used liquid is water.

Parameters

- **h** (*number*) – Water depth (m)
- **tau** (*number*) – The wall shear stress (Pa)
- **delta** (*number*) – The boundary layer thickness (m)
- **Cc** (*number, optional*) – The Coles parameters (n.u) (0.45 by default)
- **visc_c** (*number, optional*) – Kinematic viscosity (m^2/s) (defaul = 1e-6)
- **rho** (*number, optional*) – liquid density (kg/m^3) (default = 1000)

class IMTreatment.boundary_layer.boundary_layer.**WallLaw**(*h*, *tau*, *delta*, *visc_c=1e-06*, *rho=1000*)

Bases: object

Class representing a law of the wall profile. By default, the used liquid is water.

Parameters

- **h** (*number*) – Water depth (m)
- **tau** (*number*) – The wall shear stress (Pa)
- **visc_c** (*number, optional*) – Kinematic viscosity (m^2/s)

- **rho** (*number, optional*) – liquid density (kg/m^3)

display(*dy*, ***plotArgs*)
Display the velocity profile.

Parameters **dy** (*number*) – Resolution along the water depth (m).

Returns **fig** – Reference to the displayed figure.

Return type figure reference

get_profile(*y*)
Return a log-defect profile, according to the given parameters.

Parameters **y** (*array*) – Value of y in which calculate the profile (m).

Returns **prof** – the profile for values of 'y'

Return type Profile object

integral(*x*, *y*)

IMTreatment.boundary_layer.boundary_layer.**get_bl_thickness**(*obj*, *direction=1*, *perc=0.95*)
Return a boundary layer thickness if 'obj' is a Profile. Return a profile of boundary layer thicknesses if 'obj' is a ScalarField. WARNING : the wall must be at x=0.

Parameters

- **obj** (*Profile or ScalarField object*) – Vx field.
- **direction** (*integer, optional*) – If 'obj' is a ScalarField, determine the swept axis (1 for x and 2 for y).
- **perc** (*float, optionnal*) – Percentage used in the bl calculation (95% per default).

Returns **BLT** – Boundary layer thickness, in axe x unit.

Return type float or profile

IMTreatment.boundary_layer.boundary_layer.**get_clauser_thickness**(*obj*, *direction=1*, *rho=1000*, *nu=1e-06*, *tau=None*)
Return the profile Clauser's thickness defined in 'Clauser (1956)'. (Delta_star = integrale_0_h (u_top - u)/u_star dy)

Parameters

- **obj** (*Profile or ScalarField object*) –
- **direction** (*integer, optional*) – If 'obj' is a ScalarField, determine the swept axis (1 for x and 2 for y).
- **rho** (*number, optional*) – Density of the fluid (default fo water : 1000 kg/m^3)
- **nu** (*number, optional*) – Kinematic viscosity for the fluid (default for water : 1e-6 m^2/s)
- **tau** (*number, optional*) – Wall shear stress, if not specified, 'get_shear_stress' is used to compute it.

Returns **Delta_star** – Boundary layer Clauser thickness, in axe x unit.

Return type float or *Profile*

IMTreatment.boundary_layer.boundary_layer.**get_displ_thickness**(*obj*, *direction=1*)

Return a displacement thickness if 'obj' is a Profile. Return a profile of displacement thicknesses if 'obj' is a Scalarfield. WARNING : the wall must be at x=0.

Parameters

- **obj** (*Profile or ScalarField object*) –
- **direction** (*integer, optional*) – If 'obj' is a ScalarField, determine the swept axis (1 for x and 2 for y).

Returns **delta** – Boundary layer displacement thickness, in axe x unit.

Return type float or *Profile*

IMTreatment.boundary_layer.boundary_layer.**get_momentum_thickness**(*obj*, *direction=1*)

Return a momentum thickness if 'obj' is a Profile. Return a profile of momentum thicknesses if 'obj' is a Scalarfield. WARNING : the wall must be at x=0.

Parameters

- **obj** (*Profile or ScalarField object*) –
- **direction** (*integer, optional*) – If 'obj' is a ScalarField, determine the swept axis (1 for x and 2 for y).

Returns **delta** – Boundary layer momentum thickness, in axe x unit.

Return type float or *Profile*

IMTreatment.boundary_layer.boundary_layer.**get_separation_position**(*obj*, *wall_direction*, *wall_position*, *interval=None*, *nmb_lines=4*)

Compute and return the separation points position. Separation points position is computed by searching zero streamwise velocities on surrounding field lines and by extrapolating at the wanted 'wall_position'. If specified, 'interval' must include separation points on the 4 nearest field line. If multiples changments of streamwise velocity are found, return the mean positions of those points.

Parameters

- **obj** (*ScalarField, VectorField, VectorField or TemporalVelocityField*) – If 'VectorField' or 'VectorField', wall_direction is used to determine the interesting velocity component.
- **wall_direction** (*integer*) – 1 for a wall at a given value of x, 2 for a wall at a given value of y.
- **wall_position** (*number*) – Position of the wall.
- **interval** (*2x1 array of numbers, optional*) – Optional interval in which search for the detachment points.
- **nmb_lines** (*int*) – Number of lines to take into account to make the extrapolation. (default is 4)

IMTreatment.boundary_layer.boundary_layer.**get_shape_factor**(*obj*, *direction=1*)

Return a shape factor if 'obj' is a Profile. Return a profile of shape factors if 'obj' is a Scalarfield. WARNING : the wall must be at x=0.

Parameters

- **obj** (*Profile or ScalarField object*) –
- **direction** (*integer, optional*) – If 'obj' is a ScalarField, determine the swept axis (1 for x and 2 for y).

Returns **delta** – Boundary layer shape factor, in axe x unit.

Return type float or *Profile*

IMTreatment.boundary_layer.boundary_layer.**get_shear_stress**(*obj*, *direction=1*, *method='simple'*, *respace=False*, *tau_w_guess=1e-06*, *rho=1000.0*, *nu=1e-06*)

Return the wall shear stress. If velocities values are missing near the wall, an extrapolation (bad accuracy) is used. Warning : the wall must be at x=0

Parameters

- **obj** (*Profile or ScalarField object*) – .
- **viscosity** (*number, optional*) – Dynamic viscosity (default to water : 1e-3)
- **direction** (*integer, optional*) – If 'obj' is a ScalarField, determine the swept axis (1 for x and 2 for y).
- **method** (*string, optional*) – 'simple' (default) : use simple gradient computation 'wall_law_lin' : use the linear part of the 'law of the wall' model (need some points in the viscous sublayer) 'wall_law_log' : use the log part of the 'law of the wall' model (only valid in the log layer)
- **respace** (*bool, optional*) – Use linear interpolation to create an evenly spaced profile.
- **tau_w_guess** (*number, optional*) – For 'Wall_law_log' method, initial guess for tau_w resolution.
- **rho** (*number, optional*) – Density of the fluid (default fo water : 1000 kg/m^3)
- **nu** (*number, optional*) – Kinematic viscosity for the fluid (default for water : 1e-6 m^2/s)

5.2.3.2 potential_flow module

class IMTreatment.potential_flow.potential_flow.**Panel**(*xya*, *xyb*, *sigma=0.0*)

Bases: object

Contains information related to a panel.

display(*wall_vel=True*, *nodes=True*)

Display the panel

on_panel(*x*, *y*, *eps_abs=0.0001*)

Check if the point is on the panel.

vector

class IMTreatment.potential_flow.potential_flow.**System**(*u_inf=0.0*, *alpha=0.0*)

Bases: object

Representing a potential system (boundaries, sources, freestream, ...)

add_object(*dim*, *coords*, *kind='wall'*, *res=1*)
Add an object to the system

Parameters

- **dim** (*integer*) – Can be '0' for a point, '1' for a boundary or '2' for a solid.
- **coords** (*array of number*) – Coordinates of the object (1x2 array if 'dim=1', nx2 arrays else).
- **kind** (*string, optional*) – Kind of object (default is 'wall').
- **res** (*integer*) – Resolution, length of the wanted segments.

compute_pressure_from_velocity(*obj*, *raw=False*)
Return the pressure coefficient computed from velocity data.

compute_velocity(*x*, *y*)
Return the velocity at the given point.

compute_velocity_on_grid(*grid_x*, *grid_y*, *raw=False*, *remove_solid=False*)
Returns the velocity field on the given grid.

compute_velocity_on_line(*xya*, *xyb*, *res=100*, *raw=False*, *remove_solid=False*)
Return the velocity on the given line.

display(*solid=True*, *panels=True*)
Display the current geometry.

get_solid_panels()
Return all the solid panels.

get_solid_paths()
Return the solid paths.

solving_sigma()
Solve sigma values for the current boundaries.

IMTreatment.potential_flow.potential_flow.**get_gradP_length**(*D=1.0*, *Vd=1.0*, *perc=0.1*, *nu=1e-06*, *res_pot=20*, *res_int=500*, *eps=1e-06*)
Return the position before the obstacle where the pressure is equal at the wanted percentage of the pressure at the obstacle, using potential flow theory.

Parameters

- **D** (*number*) – Obstacle diameter [m].
- **Vd** (*number*) – Bulk velocity [m/s].
- **perc** (*number*) – wanted percentage (default to 0.1).
- **nu** (*number*) – Kinematic viscosity [].
- **res_pot** (*integer*) – Resolution for potential flow model (default=20).
- **res_int** (*integer*) – Resolution for Thwaites integral resolution (default=500).
- **eps** (*number*) – Wanted precision on the position (for the iterative solver)

Returns **dP_len** – Position of 10% pressure.

Return type number

IMTreatment.potential_flow.potential_flow.**get_separation_position**(*D=1.0*, *Vd=1.0*, *x_obst=10.0*, *theta_max=inf*, *nu=1e-06*, *res_pot=20*, *res_int=500*)

Use potential flow to get velocity distribution and Thwaites BL equation to get the separation position.

Parameters

- **D** (*number*) – Obstacle diameter [m].
- **Vd** (*number*) – Bulk velocity [m/s].
- **x_obst** (*number*) – Distance from CL birth to obstacle [m].
- **theta_max** (*number*) – Maximum value of BL momentum thickness (in case of confinement) [m].
- **nu** (*number*) – Kinematic viscosity [].
- **res_pot** (*integer*) – Resolution for potential flow model (default=20).
- **res_int** (*integer*) – Resolution for Thwaites integral resolution (default=500).

Returns **x_sep** – Separation position

Return type number

IMTreatment.potential_flow.potential_flow.**integral**(*x*, *y*, *panel*, *dxdz*, *dydz*)

Evaluates the contribution of a panel at one point.

Parameters

- **y -- Cartesian coordinates of the point.** (*x,*) –
- **-- panel which contribution is evaluated.** (*panel*) –
- **-- derivative of x in the z-direction.** (*dxdz*) –
- **-- derivative of y in the z-direction.** (*dydz*) –

Returns

Return type Integral over the panel of the influence at one point.

class IMTreatment.potential_flow.potential_flow.**object_0D**(*x*, *y*)

Bases: object

Representing a Source or a sink.

class IMTreatment.potential_flow.potential_flow.**object_1D**(*coords*, *kind*, *res*)

Bases: object

Representing a boundary (wall, source or sink).

display(*panels=True*)

Display the 1D object

inverse_normals()

Inverse normals.

refine(*res*)

Refine each segments of the solid.

class IMTreatment.potential_flow.potential_flow.**object_2D**(*coords*, *kind*, *res*)
Bases: *IMTreatment.potential_flow.potential_flow.object_1D*

Representing a solid (wall, source or sink).

display(*solid=True*, *panels=True*)
Display the 3D object

5.2.3.3 vortex_creation module

class IMTreatment.vortex_creation.vortex_creation.**BurgerVortex**(*x0=0.0*, *y0=0.0*, *alpha=1e-06*, *ksi=1.0*, *viscosity=1e-06*, *movable=True*, *idi=0*)
Bases: *IMTreatment.vortex_creation.vortex_creation.Vortex*

Representing a Burger Vortex, a stationnary self-similar flow, caused by the balance between vorticity creation at the center and vorticity diffusion.

5.2.3.3.1 Notes

Analytical Vortex Solutions to the Navier-Stokes Equation. Thesis for the degree of Doctor of Philosophy, Växjö University, Sweden 2007.

get_vector(*x*, *y*)
Return the velocity vector at the given point.

class IMTreatment.vortex_creation.vortex_creation.**CustomField**(*funct*, *unit_values=''*, *idi=0*)
Bases: object

Representing a custom field.

Parameters **funct** (*function*) – Representing the field. Has to take (x, y) as input and return (Vx, Vy).

copy()

get_vector(*x*, *y*)

get_vector_field(*axe_x*, *axe_y*, *unit_x=''*, *unit_y=''*)

class IMTreatment.vortex_creation.vortex_creation.**FreeVortex**(*x0=0.0*, *y0=0.0*, *gamma=1.0*, *movable=True*, *idi=0*)
Bases: *IMTreatment.vortex_creation.vortex_creation.Vortex*

Representing a Free (irrotational) Vortex. Due to its definition, the center of the vortex is a singular point (V = inf) set to 0 in this implementation.

get_vector(*x*, *y*)
Return the velocity vector at the given point.

class IMTreatment.vortex_creation.vortex_creation.**HillVortex**(*x0=0*, *y0=0*, *U=1.0*, *rv=1.0*, *rot_dir=1*, *unit_values=''*, *movable=True*, *idi=0*)

Bases: *IMTreatment.vortex_creation.vortex_creation.Vortex*

Representing a Hill Vortex, a convected vortex sphere in a inviscid flow.

5.2.3.3.2 Notes

Analytical Vortex Solutions to the Navier-Stokes Equation. Thesis for the degree of Doctor of Philosophy, Växjö University, Sweden 2007.

get_vector(*x*, *y*)

Return the velocity vector at the given point.

class IMTreatment.vortex_creation.vortex_creation.**HsvSystem**(*u_D*, *D*, *h*, *delta*, *vertical_wall=True*)

Bases: *IMTreatment.vortex_creation.vortex_creation.VortexSystem*

class IMTreatment.vortex_creation.vortex_creation.**LambChaplyginVortex**(*x0=0*, *y0=0*, *U=1.0*, *rv=1.0*, *Bessel_root_nmb=1*, *movable=True*, *idi=0*)

Bases: *IMTreatment.vortex_creation.vortex_creation.Vortex*

Representing a Lamb-Chaplygin dipole vortex, with potential flow in the exterior region, and a linear relation between stream function and vorticity in the inner region.

5.2.3.3.3 Notes

Analytical Vortex Solutions to the Navier-Stokes Equation. Thesis for the degree of Doctor of Philosophy, Växjö University, Sweden 2007.

J(*order*, *x*)

Return the value of the Bessel function with the given order ar the given point.

Parameters

- **order** (*number*) – Order of the Bessel function
- **x** (*number*) – Value where we want the Bessel function evaluation.

Returns **y** – Bessel function value at 'x'

Return type number

get_vector(*x*, *y*)

Return the velocity vector at the given point.

class IMTreatment.vortex_creation.vortex_creation.**LambOseenVortex**(*x0=0*, *y0=0*, *ksi=1.0*, *t=1.0*, *viscosity=1e-06*, *movable=True*, *idi=0*)

Bases: *IMTreatment.vortex_creation.vortex_creation.Vortex*

Representing a Lamb-Oseen Vortex, a vortex with decay due to viscosity. (satisfy NS)

5.2.3.3.4 Notes

Analytical Vortex Solutions to the Navier-Stokes Equation. Thesis for the degree of Doctor of Philosophy, Växjö University, Sweden 2007.

get_vector(*x*, *y*)
Return the velocity vector at the given point.

class IMTreatment.vortex_creation.vortex_creation.**PersoVortex**(*x0=0*, *y0=0*, *radius=1.0*, *vort_max=None*, *circ=None*, *node_ratio=0.0*, *nu=1e-06*, *movable=True*, *idi=0*)

Bases: *IMTreatment.vortex_creation.vortex_creation.Vortex*

Representing a slice of a 3D vortex.

get_vector(*x*, *y*)
Return the velocity vector at the given point.

class IMTreatment.vortex_creation.vortex_creation.**RankineVortex**(*x0=0.0*, *y0=0.0*, *circ=1.0*, *rv=1.0*, *movable=True*, *idi=0*)

Bases: *IMTreatment.vortex_creation.vortex_creation.Vortex*

Representing a Rankine Vortex, with an inner zone or forced vortex, and an outer zone of free vortex.

5.2.3.3.5 Notes

Giaiotti, DARIO B., et FULVIO Stel. « The Rankine vortex model ». PhD course on Environmental Fluid Mechanics-ICTP/University of Trieste, 2006.

get_vector(*x*, *y*)
Return the velocity vector at the given point.

class IMTreatment.vortex_creation.vortex_creation.**SolidVortex**(*x0=0.0*, *y0=0.0*, *omega=1.0*, *movable=True*, *idi=0*)

Bases: *IMTreatment.vortex_creation.vortex_creation.Vortex*

Representing a solid rotation.

get_vector(*x*, *y*)
Return the velocity vector at the given point.

class IMTreatment.vortex_creation.vortex_creation.**StepSystem**(*u_D*, *H*, *h*, *delta*)
Bases: *IMTreatment.vortex_creation.vortex_creation.VortexSystem*

class IMTreatment.vortex_creation.vortex_creation.**Vortex**(*x0*, *y0*, *movable*, *idi=0*)
Bases: object

copy()

get_vector_field(*axe_x*, *axe_y*, *unit_x=''*, *unit_y=''*)

symetrize()
Return a symetrized copy of the vortex

x0

y0

class IMTreatment.vortex_creation.vortex_creation.**VortexSystem**
Bases: object

Representing a set of vortex.

add_custom_field(*custfield*)
Add a custom field to the vortex system

Parameters **custfield** (*CustomField object*) – Custom field to add.

add_vortex(*vortex*)
Add a vortex, or a custom field to the set.

Parameters **vortex** (*Vortex or CustomField object*) – vortex or field to add to the set

add_wall(*wall*)
Add a wall to the vortex system

Parameters **wall** (*Wall object*) – Wall to add.

copy()
Return a copy.

display()
Display a representation of the vortex system

display_compounded_vector(*x*, *y*, *scale=1.0*, *detailed=False*)
Display the compounded vector in 'x', 'y' position.

get_compounded_vector(*x*, *y*, *detailed=False*)
Return a list f vector, for each structure of the system

Parameters

- **y** (*x,*) – Wanted vector coordinates
- **detailed** (*boolean, optional*) – If 'False' (default), vortex contribution and its imaginary vortex contribution are summed up. Else, return one vector for each contribution.

Returns

- **vects** (*Nx2 array of numbers*) – Vectors associated to vortex contribution

- **vortex_ids** (*Nx1 array of integer*) – Associated id
- **verts_cf** (*Mx2 array of numbers*) – Vectors associated to custom fields contribution
- **cf_ids** (*Mx1 array of integer*) – Associated id

get_evolution(*dt=1.0*)
Change the position of the vortex, according to the resulting velocity field and the time step.

Parameters **dt** (*number*) – time step.

Returns **vs** – New vortex system at t+dt

Return type VortexSystem object

get_temporal_vector_field(*dt*, *axe_x*, *axe_y*, *nmb_it*, *unit_x=''*, *unit_y=''*, *unit_time=''*)

get_vector(*x*, *y*, *vortex_ids='all'*, *cf_ids='all'*)
Return the resulting velocity vector, at the given point.

Parameters

- **y** (*x,*) – Position of the wanted vector.
- **vortex_ids** (*list of integers, 'all' or 'none'*) – Can be used to filter the wanted vortex influence by ids.
- **cf_ids** (*list of integers, 'all' or 'none'*) – Can be used to filter the wanted custom fields influence by ids.

Returns **Vx, Vy** – Velocity components.

Return type numbers

get_vector_field(*axe_x*, *axe_y*, *unit_x=''*, *unit_y=''*)
Return a vector field on the given grid

Parameters

- **axe_y** (*axe_x,*) – x and y axis
- **unit_y** (*unit_x,*) – Axis unities

Returns **vf** – .

Return type VectorField object

get_velocity_map(*axe_x*, *axe_y*, *vort*, *unit_x='m'*, *unit_y='m'*)
Return the velocity map of vortex 'vort' on the vortex system (velocity that the vortex 'vort' will have if placed at different space position on the vortex system)

get_vortex_evolution(*dt*, *nmb_it*, *unit_time=''*)

refresh_imaginary_vortex()

remove_vortex(*ind*)
Remove a vortex or a custom field from the set.

Parameters **ind** (*integer*) – Vortex indice to remove.

class IMTreatment.vortex_creation.vortex_creation.**Wall**(*x=None*, *y=None*)
Bases: object

Representing a wall

get_symmetry(*pt*)
Give the symmetry of 'pt' according to the wall.

5.2.3.4 vortex_detection module

class IMTreatment.vortex_detection.vortex_detection.**CritPoints**(*unit_time='s'*)
Bases: object

Class representing a set of critical point associated to a VectorField.

Parameters **unit_time** (*string or Unit object*) – Unity for the time.

add_point(*foc=None*, *foc_c=None*, *node_i=None*, *node_o=None*, *sadd=None*, *time=None*)
Add a new point to the CritPoints object.

Parameters

- **foc_c, node_i, node_o, sadd** (*foc,*) – Representing the critical points at this time.
- **time** (*number*) – Time.

break_trajectories(*x=None*, *y=None*, *smooth_size=5*, *inplace=False*)
Break the trajectories in pieces. Usefull to do before using 'get_mean_trajectory'

Parameters

- **y** (*x,*) – Position of breaking If not specified, break the trajectories where dx/dv=0
- **smooth_size** (*number*) – Smoothing used for the dx/dv=0 detection

change_unit(*axe*, *new_unit*)
Change the unit of an axe.

Parameters

- **axe** (*string*) – 'y' for changing the y axis unit 'x' for changing the x axis unit 'time' for changing the time unit
- **new_unit** (*Unum.unit object or string*) – The new unit.

clean_traj(*min_nmb_in_traj*)
Remove some isolated points.

Parameters **min_nmb_in_traj** (*integer*) – Trajectories that have less than this amount of points are deleted. Associated points are also deleted.

compute_traj(*epsilon=None*, *close_traj=False*)
Compute cp trajectory from cp positions.

Parameters

- **epsilon** (*number, optional*) – Maximal distance between two successive points. default value is Inf.
- **close_traj** (*bool*) – If 'True', try to close the trajectories (better to get the cp fusion position)

copy()
Return a copy of the CritPoints object.

crop(*intervx=None*, *intervy=None*, *intervt=None*, *ind=False*, *inplace=False*)
crop the point field.

display(*fields=None*, *cpkw={}*, *lnkw={}*, *display_traj=False*, *buffer_size=100*, ***kwargs*)
Display some critical points.

Parameters **fields** (*TemporalVectorFields object, optional*) – If specified, critical points are displayed on the given field. critical lines are also computed and displayed

display_3D(*xlabel=''*, *ylabel=''*, *zlabel=''*, *title=''*, ***plotargs*)

display_animate(*TF*, ***kw*)
Display an interactive windows with the velocity fields from TF and the critical points.

TF [TemporalFields] .

kw [dict, optional] Additional arguments for 'TF.display()'

display_arch(*indice=None*, *time=None*, *field=None*, *cpkw={}*, *lnkw={}*)
Display some critical points.

Parameters

- **time** (*number, optional*) – If specified, critical points associated to this time are displayed.
- **indice** (*integer, optional*) – If specified, critical points associated to this indice are displayed.
- **field** (*VectorField object, optional*) – If specified, critical points are displayed on the given field. critical lines are also computed and displayed

display_traj(*data='default'*, *filt=None*, ***kw*)
Display the stored trajectories.

Parameters

- **data** (*string*) – If 'default', trajectories are plotted in a 2-dimensional plane. If 'x', x position of cp are plotted against time. If 'y', y position of cp are plotted against time.
- **filt** (*array of boolean*) – Filter on CP types.
- **kw** (*dict, optional*) – Arguments passed to plot.

display_traj_3D(*xlabel=''*, *ylabel=''*, *zlabel=''*, *title=''*, ***plotargs*)

display_traj_len_repartition()
display profiles with trajectories length repartition (usefull to choose the right epsilon)

get_mean_trajectory(*cp_type*, *min_len=20*, *min_nmb_to_avg=10*, *rel_diff_epsilon=0.03*, *rel_len_epsilon=0.25*, *verbose=False*)
Return a mean trajectory (based on a set of trajectories)

Parameters

- **cp_type** (*string in ['foc', 'foc_c', 'node_i', 'node_o', 'sadd']*) – or a set of trajectories Trajectory type to average.
- **min_len** (*number*) – Ignore trajectories with length smaller.
- **min_nmb_to_avg** (*number*) – Minimum number of values necessary to make an average
- **rel_diff_epsilon** (*number*) – Maximum relative difference used to determine the different mean trajectories.
- **rel_len_epsilon** (*number*) – Maximum relative length difference used to determine the different mean trajectories.

Returns

- **mean_traj** (*MeanTrajectory object*) – .
- **skipped_traj** (*tuple of Points objects*) – Skipped trajectories

5.2.3.4.1 Notes

-You may want to separate your lonf trajectories with 'break_trajectories' before using this function -Checking trajectory resemblance ake into account length of the

trajectories and integrale of their differences.

get_points_density(*kind*, *bw_method=None*, *resolution=100*, *output_format='normalized'*)
Return the presence density map for the given point type.

Parameters

- **kind** (*string*) – Type of critical point for the density map (can be 'foc', 'foc_c', 'sadd', 'node_i', 'node_o')
- **bw_method** (*str, scalar or callable, optional*) – The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If a scalar, this will be used as std (it should aprocimately be the size of the density node you want to see). If a callable, it should take a gaussian_kde instance as only parameter and return a scalar. If None (default), 'scott' is used. See Notes for more details.
- **resolution** (*integer or 2x1 tuple of integers, optional*) – Resolution for the resulting field. Can be a tuple in order to specify resolution along x and y.
- **output_format** (*string, optional*) – 'normalized' (default) : give position probability (integral egal 1). 'absolute' : sum of integral over all points density egal 1. 'ponderated' : give position probability ponderated by the number

  or points (integral egal number of points).

  'concentration' : give local concentration (in point per surface).

get_traj_direction_changement(*cp_type*, *direction*, *smoothing=None*)
Return the position of trajectories direction changement.

Parameters

- **cp_type** (*string in ['foc', 'foc_c', 'node_i', 'node_o', 'sadd']*) – CP type to use
- **direction** (*string in ['x', 'y']*) – Direction along which look
- **smoothing** (*number, optional*) – Smoothing size (performed on value before gradient search).

Returns **chg_pts_1, chg_pts_2** – .

Return type Points objects

iter

iter_traj

refine_cp_position(*cp_type*, *fields*, *inplace=True*, *verbose=True*, *extrema='max'*)
Refine the position of the critical points by putting them on the given scalar field extrema.

Parameters

- **cp_type** (*string in ['foc', 'foc_c', 'sadd', 'node_i', 'node_o']*) – Critical points type to refine.
- **fields** (*TemporalScalarFields object*) – fields where to search for extrema.

- **extrema** (*string in ['min', 'max']*) – If 'max', cp are displaced on field maxima, if 'min', cp are displaced on field minima.
- **inplace** (*boolean*) – .
- **verbose** (*boolean*) – .

remove_point (*time=None*, *indice=None*)
Remove some critical points.

Parameters

- **time** (*number, optional*) – If specified, critical points associated to this time are removed.
- **indice** (*integer, optional*) – If specified, critical points associated to this indice are removed.

scale (*scalex=1.0*, *scaley=1.0*, *scalev=1.0*, *inplace=False*)
Change the scale of the axis.

Parameters

- **scaley, scalev** (*scalex,*) – scales along x, y and v
- **inplace** (*boolean, optional*) – If 'True', scaling is done in place, else, a new instance is returned.

set_origin (*x=None*, *y=None*)

smooth_traj (*tos='uniform'*, *size=None*)
Smooth the CP trajectories.

Parameters

- **tos** (*string, optional*) – Type of smoothing, can be 'uniform' (default) or 'gaussian' (See ndimage module documentation for more details)
- **size** (*number, optional*) – Size of the smoothing (is radius for 'uniform' and sigma for 'gaussian'). Default is 3 for 'uniform' and 1 for 'gaussian'.

topo_simplify (*dist_min*, *kind='replacement'*)
Simplify the topological points field.

Parameters

- **dist_min** (*number*) – Minimal distance between two points in the simplified field.
- **kind** (*string, optional*) – Algorithm used to simplify the field, can be 'replacement'(default) for iterative replacement with center of mass or 'only_delete' to eliminate only the first order associates.

Returns **simpl_CP** – Simplified topological points field.

Return type CritPoints object

unit_time

unit_x

unit_y

class IMTreatment.vortex_detection.vortex_detection.**MeanTrajectory**(*xy=array([], shape=(0, 2), dtype=float64), time=[], associated_real_times=[], associated_std_x=[], associated_std_y=[], nmb_traj_used=0, base_trajs=[], unit_x='', unit_y='', unit_times='', name='')*

Bases: *IMTreatment.core.points.Points*

crop(*intervx=None*, *intervy=None*, *intervt=None*, *inplace=True*, *ind=False*)
Crop the points cloud.

Parameters

- **intervx** (*2x1 tuple*) – Interval on x axis
- **intervy** (*2x1 tuple*) – Interval on y axis
- **intervt** (*2x1 tuple*) – Interval on time

Returns **tmp_pts** – croped version of the point cloud.

Return type Points object

display(**args*, ***kwargs*)
Display the set of points.

Parameters

- **kind** (*string, optional*) – Can be 'plot' (default if points have not values). or 'scatter' (default if points have values). or 'colored_plot'.
- **axe_y, axe_color** (*axe_x,*) – To determine wich value has to be plotted along which axis, and whith value is used to color the scattered points. Default plot 'y' to 'x' with colors from 'v'.
- ****plotargs** (*dic*) – Additionnal arguments sent to 'plot' or 'scatter'

display_error_bars(*kind='bar'*, ***kwargs*)
Display the error bar according to the trajectories used to compute the mean trajectory.

Parameters **kind** (*string in ['bar', 'envelope']*) – If 'bar', display error bar, if 'envelope', display envelope around trajectory

reconstruct_fields(*TF*)
Do a conditionnal averaging based on the CP positions.

Parameters **TF** (TemporalFields) – .

scale(*scalex=1.0*, *scaley=1.0*, *scalet=1.0*, *inplace=False*)
Change the scale of the axis.

Parameters

- **scaley, scalev** (*scalex,*) – scales along x, y and v
- **inplace** (*boolean, optional*) – If 'True', scaling is done in place, else, a new instance is returned.

time

unit_times

used_pts_number

used_traj_number

class IMTreatment.vortex_detection.vortex_detection.**TopoPoints**
Bases: object

Represent a topological points field. (only for topo simplification, in fact, should be merged with CritPoints)

Parameters

- **xy** (*Nx2 array of numbers*) – Points positions
- **types** (*Nx1 array of integers*) – Type code (1:focus, 2:focus_c, 3:saddle, 4:node_i, 5:node_o)

NL_simplifiy(*window_size*)
Simplify the topological field using Non-local criterions.

Parameters **window_size** (*number*) – Window size used to compute non-local criterions.

display()

import_from_CP(*CP_obj*, *wanted_ind*)
Import from a CritPoints object, the critical points from the time associated with the given indice.

import_from_arrays(*xy*, *types*)

simplify(*dist_min*, *kind='replacement'*)
Simplify the topological points field.

Parameters

- **dist_min** (*number*) – Minimal distance between two points in the simplified field.
- **kind** (*string, optional*) – Algorithm used to simplify the field, can be 'replacement'(default) for iterative replacement with center of mass or 'only_delete' to eliminate only the first order associates.

Returns **simpl_TP** – Simplified topological points field.

Return type TopoPoints object

class IMTreatment.vortex_detection.vortex_detection.**VF**(*vx*, *vy*, *axe_x*, *axe_y*, *mask*, *theta*, *time*)
Bases: object

export_to_velocityfield()
Return the field as VectorField object.

get_cp_cell_position()
Return critical points cell positions and their associated PBI. (PBI : Poincarre_Bendixson indice) Positions are returned in axis unities (axe_x and axe_y) at the center of a cell.

Returns

- **pos** (*2xN array*) – position (x, y) of the detected critical points.

- **type** (*1xN array*) – type of CP :

ind	CP type
0	saddle point
1	unstable focus
2	stable focus
3	unstable node
4	stable node

get_cp_position(*thread=1*)

Return critical points positions and their associated PBI using bilinear interpolation. (PBI : Poincarre_Bendixson indice) Positions are returned in axis unities (axe_x and axe_y).

Parameters **thread** (*integer or 'all'*) – Number of thread to use (multiprocessing).

Returns

- **pos** (*2xN array*) – position (x, y) of the detected critical points.
- **pbis** (*1xN array*) – PBI (1 indicate a node, -1 a saddle point)

Note: Using the work of : [1]F. Effenberger and D. Weiskopf, "Finding and classifying critical points of 2D vector fields: a cell-oriented approach using group theory," Computing and Visualization in Science, vol. 13, no. 8, pp. 377–396, Dec. 2010.

---

get_field_around_pt(*xy*, *nmb_lc*)

Return a field around the point given by x and y. nm_lc give the number of line and column to take around the point.

get_pbi(*direction*)

Return the PBI along the given axe.

IMTreatment.vortex_detection.vortex_detection.**get_critical_points**(*obj*, *time=0*, *unit_time=''*, *window_size=4*, *kind='pbi'*, *mirroring=None*, *mirror_interp='linear'*, *smoothing_size=0*, *verbose=False*, *thread=1*)

For a VectorField of a TemporalVectorField object, return the critical points positions and informations on their type.

Parameters

- **obj** (*VectorField or TemporalVectorFields objects*) – Base vector field(s)
- **time** (*number, optional*) – if 'obj' is a VectorField, 'time' is the time associated to the field.

- **unit_time** (*string or Unum.units object*) – Unit for 'time'
- **window_size** (*integer, optional*) – Size of the interrogation windows for the computation of critical point position (defaut : 4).
- **kind** (*string, optional*) – Method used to compute the critical points position. can be : 'pbi_cell' for simple (fast) Poincarre-Bendixson sweep. 'pbi' for PB sweep and bilinear interpolation inside cells. 'pbi_crit' for PB sweep and use of non-local criterions. 'gam_vort' for gamma criterion extremum detection (only detect vortex).
- **mirroring** (*array of numbers*) – If specified, use mirroring to get the critical points on the eventual walls. should be an array of '[direction ('x' or 'y'), position]*N'.
- **mirror_interp** (*string, optional*) – Method used to fill the gap at the wall. 'value' : fill with the given value, 'nearest' : fill with the nearest value, 'linear' (default): fill using linear interpolation (Delaunay triangulation), 'cubic' : fill using cubic interpolation (Delaunay triangulation)
- **smoothing_size** (*number, optional*) – If specified, a gaussian smoothing of the wanted size is used before detecting the CP.
- **verbose** (*boolean, optional*) – If 'True', display message on CP detection advancement.
- **thread** (*integer or 'all'*) – Number of thread to use (multiprocessing). (Only implemented for 'kind = 'pbi'')

5.2.3.4.2 Notes

If the fields have masked values, saddle streamlines ar not computed.

IMTreatment.vortex_detection.vortex_detection.**get_vortex_position**(*obj*, *criterion=<function get_residual_vorticity>*, *criterion_args={}*, *threshold=0.5*, *rel=True*)

Return the position of the vortex (according to the given criterion) on vector field(s).

Parameters

- **vectorfield** (*VectorField or TemporalVectorFields object*) – .
- **criterion** (*function*) – Criterion used to highlight vortex position. Should be a function, taking a VectorField object and returning a ScalarField object.
- **criterion_args** (*dict*) – Additional arguments to give to the criterion function
- **threshold** (*number*) – Threshold value determining the vortex zone.
- **rel** (*Boolean*) – If 'rel' is 'True' (default), 'threshold' is relative to the extremum values of the field. If 'rel' is 'False', 'threshold' is treated like an absolut values.

IMTreatment.vortex_detection.vortex_detection.**velocityfield_to_vf**(*vectorfield*, *time*)

Create a VF object from a VectorField object.

5.2.3.5 vortex_criterions module

IMTreatment.vortex_criterions.vortex_criterions.**get_NL_residual_vorticity**(*vectorfield*, *radius=None*, *ind=False*, *mask=None*, *raw=False*)

Return the residual vorticity computed with non-local gradients.

Parameters

- **vectorfield** (*VectorField object*) – .
- **radius** (*number, optionnal*) – The radius used to choose the zone where to compute gamma for each point. If not mentionned, a value is choosen in ordre to have about 8 points in the circle. It allow to get good result, without big computation cost.
- **ind** (*boolean*) – If 'True', radius is expressed on number of vectors. If 'False' (default), radius is expressed on axis unit.
- **mask** (*array of boolean, optionnal*) – Has to be an array of the same size of the vector field object, gamma will be compute only where mask is 'False'.
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.

IMTreatment.vortex_criterions.vortex_criterions.**get_Nk_criterion**(*vectorfield*, *mask=None*, *raw=False*)

Return the scalar field of the 2D Nk criterion . Define as "||Omega||/||S||" , with "||Omega||" the rotation rate tensor norm and ||S|| the shear rate tensor norm.

Parameters

- **vectorfield** (*VectorField object*) –
- **mask** (*array of boolean, optional*) – Has to be an array of the same size of the vector field object, Nk criterion will be compute only where zone is 'False'.
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.

5.2.3.5.1 Notes

See J. Jeong and F. Hussain, "On the identification of a vortex," Journal of Fluid Mechanics, vol. 285, pp. 69–94, 1995.

IMTreatment.vortex_criterions.vortex_criterions.**get_angle_deviation**(*vectorfield*, *radius=None*, *ind=False*, *mask=None*, *raw=False*, *local_treatment='none'*, *order=1*)

Return the angle deviation field.

Parameters

- **vectorfield** (*VectorField object*) – .
- **radius** (*number, optionnal*) – The radius used to choose the zone where to compute for each field oint. If not mentionned, a value is choosen in ordre to have about 8 points in the circle. It allow to get good result, without big computation cost.
- **ind** (*boolean*) – If 'True', radius is expressed on number of vectors. If 'False' (default), radius is expressed on axis unit.
- **mask** (*array of boolean, optionnal*) – Has to be an array of the same size of the vector field object, gamma will be compute only where mask is 'False'.
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.
- **local_treatment** (*string, optional*) – If 'None' (default), angles are taken directly from the velocity field If 'galilean_inv', angles are taken from localy averaged velocity field if 'local', angles are taken from velocity fields where the velocity of the central point is localy substracted.
- **order** (*number, optional*) – Order used to compute the deviation (default 1 for sum of differences, 2 for standart deviation (std) or more)

IMTreatment.vortex_criterions.vortex_criterions.**get_delta_criterion**(*vectorfield*, *mask=None*, *raw=False*)

Return the scalar field of the 2D Delta criterion . Define as "(Q/3)**3 + (R/2)**2" , with "Q" the Q criterion, and "R" the determinant of the jacobian matrice of the velocity.

Parameters

- **vectorfield** (*VectorField object*) –
- **mask** (*array of boolean, optional*) – Has to be an array of the same size of the vector field object, iota2 will be compute only where zone is 'False'.
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.

Note: Negative values of Delta mean that the local streamline pattern is closed or spiraled.

IMTreatment.vortex_criterions.vortex_criterions.**get_divergence**(*vf*, *raw=False*)

Return a scalar field with the 2D divergence.

Parameters

- **vf** (*VectorField or TemporalVectorfields*) – Field(s) on which compute shear stress
- **raw** (*boolean, optional*) – If 'True', return an arrays, if 'False' (default), return a ScalarField object.

Returns **div** – Divergence field(s)

Return type *ScalarField* or *TemporalScalarFields*

IMTreatment.vortex_criterions.vortex_criterions.**get_enstrophy**(*vectorfield*, *radius=None*, *ind=False*, *mask=None*, *raw=False*)

Return the enstriphy field.

Parameters

- **vectorfield** (*VectorField object*) – .
- **radius** (*number, optionnal*) – The radius used to choose the zone where to integrate enstrophy for each point. If not mentionned, a value is choosen in ordre to have about 8 points in the circle.
- **ind** (*boolean*) – If 'True', radius is expressed on number of vectors. If 'False' (default), radius is expressed on axis unit.
- **mask** (*array of boolean, optionnal*) – Has to be an array of the same size of the vector field object, gamma will be compute only where mask is 'False'.
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.

IMTreatment.vortex_criterions.vortex_criterions.**get_gamma**(*vectorfield*, *radius=None*, *ind=False*, *kind='gamma1'*, *mask=None*, *raw=False*, *dev_pass=False*)

Return the gamma scalar field. Gamma criterion is used in vortex analysis. The fonction recognize if the field is ortogonal, and use an apropriate algorithm.

Parameters

- **vectorfield** (*VectorField or TemporalVectorFields object*) – .
- **radius** (*number, optionnal*) – The radius used to choose the zone where to compute gamma for each point. If not mentionned, a value is choosen in ordre to have about 8 points in the circle. It allow to get good result, without big computation cost.
- **ind** (*boolean*) – If 'True', radius is expressed on number of vectors. If 'False' (default), radius is expressed on axis unit.
- **kind** (*string*) – If 'gamma1' (default), compute gamma1 criterion. If 'gamma1b', compute gamma1 criterion with velocity corrector. (multiply with the mean velocity) If 'gamma2', compute gamma2 criterion (with relative velocities) If 'gamma2b', compute gamma2 criterion with a velocity corrector. (hide uniform velocity zone)
- **mask** (*array of boolean, optionnal*) – Has to be an array of the same size of the vector field object, gamma will be compute only where mask is 'False'.
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.
- **dev_pass** (*boolean, optional*) – If 'True', the algorithm compute gamma criterion only where the velocity angles deviation is strong (faster if there is few points). Work only with 'gamma1'

IMTreatment.vortex_criterions.vortex_criterions.**get_improved_swirling_strength**(*vf*, *raw=False*)

Return a scalar field with the improved swirling strength

Parameters

- **vf** (*VectorField or Velocityfield*) – Field on which compute shear stress
- **raw** (*boolean, optional*) – If 'True', return an arrays, if 'False' (default), return a ScalarField object.

5.2.3.5.2 Notes

Chakraborty, Pinaki, S. Balachandar, et Ronald J. Adrian. « On the Relationships between Local Vortex Identification Schemes ». Journal of Fluid Mechanics 535 (5 juillet 2005): 189-214.

IMTreatment.vortex_criterions.vortex_criterions.**get_iota**(*vectorfield*, *mask=None*, *radius=None*, *ind=False*, *raw=False*)

Return the iota scalar field. iota criterion is used in vortex analysis. The fonction is only usable on orthogonal fields. Warning : This function is minimum at the saddle point center, and maximum around this point.

Parameters

- **vectorfield** (*VectorField object*) –
- **mask** (*array of boolean, optionnal*) – Has to be an array of the same size of the vector field object, iota2 will be compute only where zone is 'False'.
- **radius** (*number, optionam*) – If specified, the velocity field is smoothed with gaussian filter of the given radius before computing the vectors angles.
- **ind** (*boolean, optional*) – If 'True', radius is an indice number, if 'False', radius if in the field units (default).
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.

IMTreatment.vortex_criterions.vortex_criterions.**get_kappa**(*vectorfield*, *radius=None*, *ind=False*, *kind='kappa1'*, *mask=None*, *raw=False*, *dev_pass=False*)

Return the kappa scalar field. Kappa criterion is used in vortex analysis. The fonction recognize if the field is ortogonal, and use an apropriate algorithm.

Parameters

- **vectorfield** (*VectorField object*) –
- **radius** (*number, optionnal*) – The radius used to choose the zone where to compute kappa for each point. If not mentionned, a value is choosen in ordre to have about 8 points in the circle. It allow to get good result, without big computation cost.
- **ind** (*boolean*) – If 'True', radius is expressed on number of vectors. If 'False' (default), radius is expressed on axis unit.
- **kind** (*string*) – If 'kappa1' (default), compute kappa1 criterion. If 'kappa2', compute kappa2 criterion (with relative velocities).
- **mask** (*array of boolean, optionnal*) – Has to be an array of the same size of the vector field object, kappa will be compute only where mask is 'False'.
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.

- **dev_pass** (*boolean, optional*) – If 'True', the algorithm compute gamma criterion only where the velocity angles deviation is strong (faster if there is few points)

IMTreatment.vortex_criterions.vortex_criterions.**get_lambda2**(*vectorfield*, *mask=None*, *raw=False*)

Return the lambda2 scalar field. According to ... vortex are defined by zone of negative values of lambda2. The fonction is only usable on orthogonal fields.

Parameters

- **vectorfield** (*VectorField object*) –
- **mask** (*array of boolean, optionnal*) – Has to be an array of the same size of the vector field object, iota2 will be compute only where zone is 'False'.
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.

IMTreatment.vortex_criterions.vortex_criterions.**get_q_criterion**(*vectorfield*, *mask=None*, *raw=False*)

Return the scalar field of the 2D Q criterion . Define as "1/2*(R**2 - S**2)" , with "R" the deformation tensor, norm and "S" the rate of rotation tensor norm.

Parameters

- **vectorfield** (*VectorField object*) –
- **mask** (*array of boolean, optional*) – Has to be an array of the same size of the vector field object, Q criterion will be compute only where zone is 'False'.
- **raw** (*boolean, optional*) – If 'False' (default), a ScalarField is returned, if 'True', an array is returned.

IMTreatment.vortex_criterions.vortex_criterions.**get_residual_vorticity**(*vf*, *raw=False*)

Return a scalar field with the residual of the vorticity. (see Kolar (2007)).

Parameters

- **vf** (*VectorField or Velocityfield*) – Field on which compute shear stress
- **raw** (*boolean, optional*) – If 'True', return an arrays, if 'False' (default), return a ScalarField object.

IMTreatment.vortex_criterions.vortex_criterions.**get_shear_vorticity**(*vf*, *raw=False*)

Return a scalar field with the shear vorticity. (see Kolar (2007)).

Parameters

- **vf** (*VectorField or VectorFields*) – Field on which compute shear stress
- **raw** (*boolean, optional*) – If 'True', return an arrays, if 'False' (default), return a ScalarField object.

IMTreatment.vortex_criterions.vortex_criterions.**get_stokes_vorticity**(*vf*, *window_size=2*, *raw=False*)

Return a scalar field with the z component of the vorticity using Stokes' theorem.

Parameters

- **vf** (*VectorField or Velocityfield*) – Field on which compute shear stress

- **window_size** (*integer, optional*) – Window size for stokes approximation of the vorticity.
- **raw** (*boolean, optional*) – If 'True', return an arrays, if 'False' (default), return a ScalarField object.

5.2.3.5.3 Notes

Seal et al., "Quantitative characteristics of a laminar, unsteady necklace vortex system at a rectangular block-flat plate juncture," Journal of Fluid Mechanics, vol. 286, pp. 117–135, 1995.

IMTreatment.vortex_criterions.vortex_criterions.**get_swirling_strength**(*vf*, *raw=False*)

Return a scalar field with the swirling strength (imaginary part of the eigenvalue of the velocity Jacobian)

Parameters

- **vf** (*VectorField or Velocityfield*) – Field on which compute shear stress
- **raw** (*boolean, optional*) – If 'True', return an arrays, if 'False' (default), return a ScalarField object.

5.2.3.5.4 Notes

Zhou, J., R. J. Adrian, S. Balachandar, et T. M. Kendall. « Mechanisms for generating coherent packets of hairpin vortices in channel flow ». Journal of Fluid Mechanics 387 (mai 1999): 353-96.

IMTreatment.vortex_criterions.vortex_criterions.**get_vorticity**(*vf*, *raw=False*)

Return a scalar field with the z component of the vorticity.

Parameters

- **vf** (*VectorField or TemporalVectorfields*) – Field(s) on which compute shear stress
- **raw** (*boolean, optional*) – If 'True', return an arrays, if 'False' (default), return a ScalarField object.

Returns **vort** – Vorticity field(s)

Return type *ScalarField* or *TemporalScalarFields*

5.2.3.6 vortex_properties module

IMTreatment.vortex_properties.vortex_properties.**get_vortex_circulation**(*VF*, *vort_center*, *epsilon=0.1*, *output_unit=False*, *verbose=False*)

Return the circulation of the given vortex.

$Gamma = int_S omega dS$ avec : S : surface su vortex ($| omega | > epsilon$)

Recirculation is representative of the swirling strength.

Warning : integral on complex domain is complex (you don't say?), here is just implemented a sum of accessible values on the domain.

Parameters

- **VF** (*vectorfield object*) – Velocity field on which compute gamma2.
- **vort_center** (*2x1 array*) – Approximate position of the vortex center.
- **epsilon** (*float, optional*) – Relative seuil for the vorticity integral (default is 0.1).
- **output_unit** (*boolean, optional*) – If 'True', circulation unit is returned.

Returns **circ** – Vortex virculation.

Return type float

IMTreatment.vortex_properties.vortex_properties.**get_vortex_property**(*VF*, *vort_center*, *size_crit=None*, *size_crit_lim=0.1*, *prop_crit=None*, *output_unit=False*, *verbose=False*)

Return a property of a particular vortex.

Parameters

- **VF** (*vectorfield object*) – Base velocity field.
- **vort_center** (*2x1 array*) – Approximate position of the vortex center.
- **size_crit** (*function or 'value'*) – Function applied to 'VF' and returning a ScalarField used to get the vortex area. (Default is residual vorticity) If 'value', only the value at the given point is returned.
- **size_crit_lim** (*number*) – Used to determine the size criterion interval defining the vortex area (i.e. the vortex area is the area around the vortex center where the size criterion is superior to 'size_crit_lim' times the value at the center) (Default is 0.1 (10%)) Useless if 'size_crit='value'
- **prop_crit** (*function*) – Function applied to 'VF' and returning a ScalarField used to get the property value (Default is residual vorticity)
- **output_unit** (*boolean, optional*) – If 'True', return the associated unit.
- **verbose** (*bool*) – If 'True', display information and graph along computation.

Returns **prop** – Property associated to the vortex. (Is the integral of 'prop_crit' result on the area defined by 'size_crit')

Return type number

IMTreatment.vortex_properties.vortex_properties.**get_vortex_property_time_evolution**(*TVFs*, *vort_center*, *size_crit=N*, *size_crit_li*, *prop_crit=*, *output_unit=F*, *verbose=0*)

Return a property of a particular vortex.

Parameters

- **TVFs** (*TemporalVectorFields object*) – Base velocity fields.
- **vort_center** (*Points object*) – Approximate position of the vortex centers along times.
- **size_crit** (*function or 'value'*) – Function applied to 'VF' and returning a ScalarField used to get the vortex area. (Default is residual vorticity) if 'value', only return the value at point.
- **size_crit_lim** (*number*) – Used to determine the size criterion interval defining the vortex area (i.e. the vortex area is the area around the vortex center where the size criterion is superior to 'size_crit_lim' times the value at the center) (Default is 0.1 (10%)) Useless if "size_crit='value".
- **prop_crit** (*function*) – Function applied to 'VF' and returning a ScalarField used to get the property value (Default is residual vorticity)
- **verbose** (*integer*) – specified the number of fields to verbosify. Default is 0.

Returns **prop** – Evolution of the property associated with the vortex long time.

Return type Profile object

IMTreatment.vortex_properties.vortex_properties.**get_vortex_radius**(*VF*, *vort_center*, *NL_radius=None*, *eps_detection=0.1*, *output_center=False*, *output_unit=False*)

Return the radius of the given vortex, use the residual vorticity.

Parameters

- **VF** (*vectorfield object*) – Velocity field on which compute gamma2.
- **vort_center** (*2x1 array*) – Approximate position of the vortex center.
- **NL_radius** (*number, optional*) – IF specified, radius used for the non-local computation of the gradients.
- **eps_detection** (*number*) – epsilon used to determine the edge of the vortex (default is 0.1 for 10% of the maximum vorticity value).
- **output_center** (*boolean, optional*) – If 'True', return the associated vortex center, computed using center of mass algorythm.
- **; boolean, optional** (*output_unit*) – If 'True', return the associated unit.

Returns

- **radius** (*number*) – Average radius of the vortex. If no vortex is found, 0 is returned.
- **center** (*2x1 array of numbers*) – If 'output_center' is 'True', contain the newly computed vortex center.
- **unit_radius** (*Unit object*) – Radius unity

IMTreatment.vortex_properties.vortex_properties.**get_vortex_radius_time_evolution**(*TVFS*, *traj*, *NL_radius=N*, *eps_detection=*, *output_center=F*, *verbose=False*)

Return the radius evolution in time for the given vortex center trajectory.

Use the criterion $|gamma2| > 2/pi$. The returned radius is an average value if the vortex zone is not circular.

Parameters

- **TVFS** (*TemporalField object*) – Velocity field on which compute gamma2.
- **traj** (*Points object*) – Trajectory of the vortex.
- **NL_radius** (*number, optional*) – IF specified, radius used for the non-local computation of the gradients.
- **eps_detection** (*number*) – epsilon used to determine the edge of the vortex (default is 0.1 for 10% of the maximum vorticity value).
- **output_center** (*boolean, optional*) – If 'True', return a Points object with associated vortex centers, computed using center of mass algorythm.
- **verbose** (*boolean*) – .

Returns

- **radius** (*Profile object*) – Average radius of the vortex. If no vortex is found, 0 is returned.
- **center** (*Points object*) – If 'output_center' is 'True', contain the newly computed vortex center.

5.2.4 Virtual classes

5.2.4.1 Field class

class IMTreatment.core.field.**Field**

Bases: object

axe_x

axe_y

change_unit(*axe*, *new_unit*)

Change the unit of an Field.

Parameters

- **axe** (*string*) – 'y' for changing the profile y axis unit 'x' for changing the profile x axis unit
- **new_unit** (*Unum.unit object or string*) – The new unit.

check_consistency()

Raise errors if the field show some incoherences.

copy()

Return a copy of the Field object.

crop(*intervx=None*, *intervy=None*, *full_output=False*, *ind=False*, *inplace=False*)
Crop the field in respect with given intervals.

Parameters

- **intervx** (*array, optional*) – interval wanted along x
- **intervy** (*array, optional*) – interval wanted along y
- **full_output** (*boolean, optional*) – If 'True', cutting indices are alson returned
- **ind** (*boolean, optional*) – If 'True', intervals are understood as indices along axis. If 'False' (default), intervals are understood in axis units.
- **inplace** (*boolean, optional*) – If 'True', the field is croped in place.

dx

dy

extend(*nmb_left=0*, *nmb_right=0*, *nmb_up=0*, *nmb_down=0*, *inplace=False*)
Add columns or lines of masked values at the field.

Parameters

- **nmb_****** (*integers*) – Number of lines/columns to add in each direction.
- **inplace** (*bool*) – If 'False', return a new extended field, if 'True', modify the field inplace.

Returns **Extended_field** – Extended field.

Return type Field object, optional

get_indice_on_axe(*direction*, *value*, *kind='bounds'*)
Return, on the given axe, the indices representing the positions surrounding 'value'. if 'value' is exactly an axe position, return just one indice.

Parameters

- **direction** (*int*) – 1 or 2, for axes choice.
- **value** (*number*) –
- **kind** (*string*) – If 'bounds' (default), return the bounding indices. if 'nearest', return the nearest indice if 'decimal', return a decimal indice (interpolated)

Returns **interval**

Return type 2x1 or 1x1 array of integer

get_points_around(*center*, *radius*, *ind=False*)
Return the list of points or the scalar field that are in a circle centered on 'center' and of radius 'radius'.

Parameters

- **center** (*array*) – Coordonate of the center point (in axes units).
- **radius** (*float*) – radius of the cercle (in axes units).
- **ind** (*boolean, optional*) – If 'True', radius and center represent indices on the field. if 'False', radius and center are expressed in axis unities.

Returns **indices** – Array contening the indices of the contened points. [(ind1x, ind1y), (ind2x, ind2y), ...]. You can easily put them in the axes to obtain points coordinates

Return type array

rotate(*angle*, *inplace=False*)
Rotate the field.

Parameters

- **angle** (*integer*) – Angle in degrees (positive for trigonometric direction). In order to preserve the orthogonal grid, only multiples of 90° are accepted (can be negative multiples).
- **inplace** (*boolean, optional*) – If 'True', Field is rotated in place, else, the function return a rotated field.

Returns **rotated_field** – Rotated field.

Return type Field object, optional

scale(*scalex=None*, *scaley=None*, *inplace=False*, *output_reverse=False*)
Scale the Field.

Parameters

- **scaley** (*scalex,*) – Scale for the axis
- **inplace** (*boolean*) – .

set_origin(*x=None*, *y=None*)
Modify the axis in order to place the origin at the givev point (x, y)

Parameters

- **x** (*number*) –
- **y** (*number*) –

shape

unit_x

unit_y

5.2.4.2 Fields class

class IMTreatment.core.fields.**Fields**
Bases: object

Class representing a set of fields. These fields can have differente positions along axes, or be successive view of the same area. It's recommended to use TemporalVelocityFields or SpatialVelocityFields instead of this one.

add_field(*field*, *copy=True*)
Add a field to the existing fields.

Parameters **field** (*sf.VectorField or sf.ScalarField object*) – The field to add.

copy()
Return a copy of the velocityfields

remove_field(*fieldnumbers*)
Remove a field of the existing fields.

Parameters **fieldnumber** (*integer or list of integers*) – Velocity field(s) number(s) to remove.

rotate(*angle*, *inplace=False*)
Rotate the fields.

Parameters

- **angle** (*integer*) – Angle in degrees (positive for trigonometric direction). In order to preserve the orthogonal grid, only multiples of 90° are accepted (can be negative multiples).
- **inplace** (*boolean, optional*) – If 'True', fields is rotated in place, else, the function return rotated fields.

Returns **rotated_field** – Rotated fields.

Return type *TemporalFields* or child object, optional

scale(*scalex=None*, *scaley=None*, *scalev=None*, *inplace=False*)
Scale the Fields.

Parameters

- **scaley, scalev** (*scalex,*) – Scale for the axis and the values.
- **inplace** (*boolean*) – .

set_origin(*x=None*, *y=None*)
Modify the axis in order to place the origin at the actual point (x, y)

Parameters

- **x** (*number*) –
- **y** (*number*) –

smooth(*tos='uniform'*, *size=None*, *inplace=False*, ***kw*)
Smooth the fields in place. Warning : fill up the field (should be used carefully with masked field borders)

Parameters

- **tos** (*string, optional*) – Type of smoothing, can be 'uniform' (default) or 'gaussian' (See ndimage module documentation for more details)
- **size** (*number, optional*) – Size of the smoothing (is radius for 'uniform' and sigma for 'gaussian') in indice number. Default is 3 for 'uniform' and 1 for 'gaussian'.
- **inplace** (*boolean, optional*) – If True, Field is smoothed in place, else, the smoothed field is returned.
- **kw** (*dic*) – Additional parameters for ndimage methods (See ndimage documentation)

5.2.4.3 TemporalFields class

class IMTreatment.core.temporalfields.**TemporalFields**
Bases: *IMTreatment.core.fields.Fields*, *IMTreatment.core.field.Field*

Class representing a set of time evolving fields. All fields added to this object has to have the same axis system.

add_field(*field*, *time=0.0*, *unit_times=''*, *copy=True*)
Add a field to the existing fields.

Parameters

- **field** (*vf.VectorField or sf.ScalarField object*) – The field to add.
- **time** (*number*) – time associated to the field.
- **unit_time** (*Unum object*) – time unit.

augment_temporal_resolution(*fact=2*, *inplace=False*)
Augment the temporal resolution using temporal interpoalation.

Parameters

- **fact** (*integer*) – Temporal resolution ratio.
- **inplace** (*bool*) – .

axe_x

axe_y

change_unit(*axe*, *new_unit*)
Change the unit of an axe.

Parameters

- **axe** (*string*) – 'y' for changing the profile y axis unit 'x' for changing the profile x axis unit 'values' for changing values unit 'time' for changing time unit
- **new_unit** (*Unum.unit object or string*) – The new unit.

copy()
Return a copy of the velocityfields

crop(*intervx=None*, *intervy=None*, *intervt=None*, *full_output=False*, *ind=False*, *inplace=False*)
Return a croped field in respect with given intervals.

Parameters

- **intervx** (*array, optional*) – interval wanted along x
- **intervy** (*array, optional*) – interval wanted along y
- **intervt** (*array, optional*) – interval wanted along time
- **full_output** (*boolean, optional*) – If 'True', cutting indices are alson returned
- **inplace** (*boolean, optional*) – If 'True', fields are croped in place.

crop_masked_border(*hard=False*, *inplace=False*)
Crop the masked border of the velocity fields in place.

Parameters

- **hard** (*boolean, optional*) – If 'True', partially masked border are croped as well.
- **inplace** (*boolean, optional*) – If 'True', crop the F in place, else, return a croped TF.

display(*compo=None*, *kind=None*, *sharecb=True*, *buffer_size=100*, ***plotargs*)
Create a windows to display temporals field, controlled by buttons.

Parameters

- **compo** (*string*) – Component to plot.
- **kind** (*string*) – Kind of plot to use.
- **sharecb** (*boolean*) – Do all the vector field serie has to share the same colorbar or not.
- **buffer_size** (*number*) – Number of displays to keep in memory (faster, but use memory).
- ****plotargs** (*dic*) – Arguments passed to the plot command.
- **control** (*Display*) –

- **---------------** –
- **display can be controlled useing the button, but also the keyboard** (*The*) –
- **right arrow or +** (*space,*) –
- **left arrow or -** (*backspace,*) –
- **arrow** (*down*) –
- **arrow** –
- **+ enter** (*number*) –
- **p** (*play the animated fields*) –
- **+ i** (*number*) –
- **+ t** (*number*) –
- **q** (*close*) –
- **s** (*save an image*) –

display_animate(*compo=None*, *interval=500*, *fields_inds=None*, *repeat=True*, ***plotargs*)
Display fields animated in time.

Parameters

- **compo** (*string*) – Composante to display
- **interval** (*number, optionnal*) – interval between two frames in milliseconds.
- **fields_ind** (*array of indices*) – Indices of wanted fields. by default, all the fields are displayed
- **repeat** (*boolean, optional*) – if True, the animation is repeated infinitely.
- **arguments can be passed (scale, vmin, vmax,..)** (*additional*) –

display_multiple(*component=None*, *kind=None*, *inds=None*, *sharecb=False*, *sharex=False*, *sharey=False*, *ncol=None*, *nrow=None*, ***plotargs*)
Display a component of the velocity fields.

Parameters

- **component** (*string, optional*) – component to display
- **kind** (*string, optional*) – Kind of display wanted.
- **fields_ind** (*array of indices*) – Indices of fields to display.
- **samecb** (*boolean, optional*) – If 'True', the same color system is used for all the fields. You have to pass 'vmin' and 'vmax', to have correct results.
- **nrow** (*ncol,*) – Wanted number of columns and rows. If not specified, these values are computed so that ncol ~ nrow.
- **plotargs** (*dict, optional*) – Arguments passed to the function used to display the vector field.

dt

extend(*nmb_left=0*, *nmb_right=0*, *nmb_up=0*, *nmb_down=0*, *inplace=False*)
Add columns or lines of masked values at the fields.

Parameters

- **nmb_****** (*integers*) – Number of lines/columns to add in each direction.
- **inplace** (*bool*) – If 'False', return a new extended field, if 'True', modify the field inplace.

Returns **Extended_field** – Extended field.

Return type TemporalFields object, optional

get_fluctuant_fields(*nmb_min_mean=1*)
Calculate the fluctuant fields (fields minus mean field).

Parameters **nmb_min_mean** (*number, optional*) – Parameter for mean computation (see 'get_mean_field' doc).

Returns **fluct_fields** – Contening fluctuant fields.

Return type *TemporalScalarFields* or TemporalVectorFields object

get_interpolated_field(*time*)
Return the interpolated field happening at the time 'time'.

get_mean_field(*nmb_min=1*, *dtype=None*)
Calculate the mean velocity field, from all the fields.

Parameters

- **nmb_min** (*integer, optional*) – Minimum number of values used to make a mean. else, the value is masked
- **dtype** (*type*) – Specify the output values type (default to the same one as fields).

get_recurrence_map(*norm=2*, *verbose=False*, *bandwidth=None*, *normalized=False*)
Return the recurrence map associated with the 2-norm.

Returns **rec_map** – .

Return type sf.ScalarField object

get_spatial_spectrum(*component*, *direction*, *intervx=None*, *intervy=None*, *intervtime=None*, *welch_seglen=None*, *scaling='base'*, *fill='linear'*)
Return a spatial spectrum. If more than one time are specified, spectrums are averaged.

Parameters

- **component** (*string*) – Should be an attribute name of the stored fields.
- **direction** (*string*) – Direction in which perform the spectrum ('x' or 'y').
- **and intervy** (*intervx*) – To chose the zone where to calculate the spectrum. If not specified, the biggest possible interval is choosen.
- **intervtime** (*2x1 array, optional*) – Interval of time on which averaged the spectrum.
- **welch_seglen** (*integer, optional*) – If specified, welch's method is used (dividing signal into overlapping segments, and averaging periodogram) with the given segments length (in number of points).
- **scaling** (*string, optional*) – If 'base' (default), result are in component unit. If 'spectrum', the power spectrum is returned (in unit^2). If 'density', the power spectral density is returned (in unit^2/(1/unit_axe))
- **fill** (*string or float*) – Specifies the way to treat missing values. A value for value filling. A string ('linear', 'nearest' or 'cubic') for interpolation.

5.2.4.3.1 Notes

If there is missing values on the field, 'fill' is used to linearly interpolate the missing values (can impact the spectrum).

get_spectrum_map(*comp*, *welch_seglen=None*, *nmb_pic=1*, *spec_smooth=None*, *verbose=True*)
Return the temporal spectrum map.

Parameters

- **comp** (*string*) – Component to get the spectrum from.
- **welch_seglen** (*integer*) – .
- **nmb_pic** (*integer*) – Number of succesive spectrum pic to detect
- **spec_smooth** (*number*) – .
- **verbose** (*bool*) – .

Returns

- *map_freq_sf* – .
- *map_freq_quality_sf* – .

get_temporal_spectrum(*component*, *pt*, *ind=False*, *wanted_times=None*, *welch_seglen=None*, *scaling='base'*, *fill='linear'*, *mask_error=True*, *detrend='constant'*)
Return a Profile object, with the temporal spectrum of 'component' on the point 'pt'.

Parameters

- **component** (*string*) – .
- **pt** (*2x1 array of numbers*) – .
- **ind** (*boolean*) – If true, 'pt' is read as indices, else, 'pt' is read as coordinates.
- **wanted_times** (*2x1 array, optional*) – Time interval in which compute spectrum (default is all).
- **welch_seglen** (*integer, optional*) – If specified, welch's method is used (dividing signal into overlapping segments, and averaging periodogram) with the given segments length (in number of points).
- **scaling** (*string, optional*) – If 'base' (default), result are in component unit. If 'spectrum', the power spectrum is returned (in unit^2). If 'density', the power spectral density is returned (in unit^2/Hz)
- **fill** (*string or float*) – Specifies the way to treat missing values. A value for value filling. A string ('linear', 'nearest', 'zero', 'slinear', 'quadratic, 'cubic' where 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order) for interpolation.
- **mask_error** (*boolean*) – If 'False', instead of raising an error when masked value appear on time profile, '(None, None)' is returned.
- **detrend** (*string, optional*) – Method used to detrend the profile. Can be 'none', 'constant' (default) or 'linear'.

Returns **magn_prof** – Magnitude spectrum.

Return type prof.Profile object

get_temporal_spectrum_over_area(*component*, *intervx*, *intervy*, *ind=False*, *welch_seglen=None*, *scaling='base'*, *fill='linear'*, *detrend='constant'*)

Return a prof.Profile object, contening a mean spectrum of the given component, on all the points included in the given intervals.

Parameters

- **component** (*string*) – Scalar component ('Vx', 'Vy', 'magnitude', ...).
- **intervy** (*intervx,*) – Defining the square on which averaging the spectrum. (in axes values)
- **ind** (*boolean*) – If true, 'pt' is read as indices, else, 'pt' is read as coordinates.
- **welch_seglen** (*integer, optional*) – If specified, welch's method is used (dividing signal into overlapping segments, and averaging periodogram) with the given segments length (in number of points).
- **scaling** (*string, optional*) – If 'base' (default), result are in component unit. If 'spectrum', the power spectrum is returned (in unit^2). If 'density', the power spectral density is returned (in unit^2/Hz)
- **fill** (*string or float*) – Specifies the way to treat missing values. A value for value filling. A string ('linear', 'nearest', 'zero', 'slinear', 'quadratic, 'cubic' where 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order) for interpolation.
- **detrend** (*string, optional*) – Method used to detrend the profile. Can be 'none', 'constant' (default) or 'linear'.

Returns **magn_prof** – Averaged magnitude spectrum.

Return type prof.Profile object

get_time_profile(*component*, *pt*, *wanted_times=None*, *ind=False*)

Return a profile contening the time evolution of the given component.

Parameters

- **component** (*string*) – Should be an attribute name of the stored fields.
- **pt** (*2x1 array of numbers, or pts.Points object*) – Wanted position for the time profile, in axis units.
- **wanted_times** (*2x1 array of numbers*) – Time interval in which getting profile (default is all).
- **ind** (*boolean, optional*) – If 'True', values are undersood as indices.

Returns **profile**

Return type prof.Profile object

inject_time_profile(*comp*, *pt*, *prof*, *ind=False*)

Overwrite the value at the given points with data from the time profile.

Parameters

- **comp** (*string*) – Should be an attribute name of the stored fields.
- **pt** (*2x1 array of numbers, or pts.Points object*) – Wanted position for the time profile, in axis units.
- **prof** (*Profile object*) – Profile used to overwrite data at point

- **ind** (*boolean, optional*) – If 'True', values are understood as indices.

make_evenly_spaced(*interp='linear'*, *res=1*)
Use interpolation to make the fields evenly spaced

Parameters

- **interp** (*{'linear', 'cubic', 'quintic'}, optional*) – The kind of spline interpolation to use. Default is 'linear'.
- **res** (*number*) – Resolution of the resulting field. A value of 1 meaning a spatial resolution equal to the smallest space along the two axis for the initial field.

mask

mask_as_sf

mask_cum

mask_cum_as_sf

mirroring(*direction, position, inds_to_mirror='all', mir_coef=1.0, inplace=False, interp=None, value=[0, 0]*)
Return the fields with additional mirrored values.

Parameters

- **direction** (*integer*) – Axe on which place the symetry plane (1 for x and 2 for y)
- **position** (*number*) – Position of the symetry plane along the given axe
- **inds_to_mirror** (*integer*) – Number of vector rows to symetrize (default is all)
- **mir_coef** (*number or 2x1 array, optional*) – Optional coefficient(s) applied only to the mirrored values. It can be an array first value is for 'comp_x' and second one to 'comp_y' (for vector fields)
- **inplace** (*boolean, optional*) – .
- **interp** (*string, optional*) – If specified, method used to fill the gap near the symetry plane by interpoaltion. 'value' : fill with the given value, 'nearest' : fill with the nearest value, 'linear' (default): fill using linear interpolation (Delaunay triangulation), 'cubic' : fill using cubic interpolation (Delaunay triangulation)
- **value** (*array, optional*) – Value at the symetry plane, in case of interpolation

record_animation(*anim*, *filepath*, *kind='gif'*, *fps=30*, *dpi=100*, *bitrate=50*, *imagemagick_path=None*)
Record an animation in a gif file. You must create an animation (using 'display_animate' for example) before calling this method. You may have to specify the path to imagemagick in orfer to use it.

reduce_spatial_resolution(*fact*, *inplace=False*, *verbose=False*)
Reduce the spatial resolution of the fields by a factor 'fact'

Parameters

- **fact** (*int*) – Reducing factor.
- **inplace** (*boolean, optional*) – .

reduce_temporal_resolution(*nmb_in_interval*, *mean=True*, *inplace=False*)
Return a TemporalVelocityFields, contening one field for each 'nmb_in_interval' field in the initial TFVS.

Parameters

- **nmb_in_interval** (*integer*) – Length of the interval. (one field is kept for each 'nmb_in_interval fields)

- **mean** (*boolean, optional*) – If 'True', the resulting fields are average over the interval. Else, fields are taken directly.
- **inplace** (*boolean, optional*) –

Returns TVFS

Return type TemporalVelocityFields

remove_fields (*fieldnumbers*)
Remove field(s) of the existing fields.

Parameters **fieldnumber** (*integer or list of integers*) – Velocity field(s) number(s) to remove.

remove_weird_fields (*std_coef=3.29*, *treatment='interpolate'*, *inplace=False*)
Look at the time evolution of spatial mean magnitude to identify and replace weird fields.

Parameters

- **std_coef** (*number*) – Fields associated with mean magnitude outside the interval [mean - std_coef*std, mean - std_coef*std] are treated as weird fields. Default value of '3.29' corespond for a 99.9% interval.
- **treatment** (*string in ['remove', 'interpolate']*) – Type of treatment for the weird fields (default is 'interpolate')
- **inplace** (*bool*) – .

Returns **tf** – treated temporal field

Return type TemporalField

scale (*scalex=None*, *scaley=None*, *scalev=None*, *scalet=None*, *inplace=False*)
Scale the Fields.

Parameters

- **scaley, scalev** (*scalex,*) – Scale for the axis and the values.
- **inplace** (*boolean*) – .

set_origin (*x=None*, *y=None*)
Modify the axis in order to place the origin at the actual point (x, y)

Parameters

- **x** (*number*) –
- **y** (*number*) –

times

unit_times

unit_values

unit_x

unit_y

5.2.4.4 SpatialFields class

class IMTreatment.core.spatialfields.**SpatialFields**
Bases: *IMTreatment.core.fields.Fields*

add_field(*field*, *copy=True*)

copy()
Return a copy of the velocityfields

display(*compo=None*, ***plotargs*)

get_profile(*direction*, *position*, *component=None*)
Return a profile of the current fields.

Parameters

- **direction** (*integer*) – Direction along which we choose a position (1 for x and 2 for y)
- **position** (*float, interval of float*) – Position, interval in which we want a profile
- **component** (*string*) – Component wanted for the profile.

Returns **profile** – Wanted profile

Return type Profile object

get_value(*x*, *y*, *unit=False*, *error=True*)
Return the field component(s) on the point (x, y).

Parameters

- **y** (*x,*) – Point coordinates
- **unit** (*boolean, optional*) – If 'True', component(s) is(are) returned with its unit.
- **error** (*boolean, optional*) – If 'True', raise an error if the asked point is outside the fields. If 'False', return 'None'

get_values_on_grid(*axe_x*, *axe_y*)
Return a all the fields in a single evenly-spaced grid. (Use interpolation to get the data on the grid points)

Parameters **axe_y** (*axe_x,*) – Representing the grid axis.

mask

mask_as_sf

unit_values

unit_x

unit_y

x_max

x_min

y_max

y_min

5.3 Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y