PyFields Documentation

Release 0.1

Gaby Launay

Mar 18, 2018

Contents

PyFields is a Python3 package, intending to ease the manipulation and analysis of 2D and 3D fields, like velocity, concentration, temperature, ...

It can handle sets of fields, to analyze time evolution of the aforementioned fields. It includes easy-to-use helpers for importing, analyzing, displaying and exporting fields.

CHAPTER 1

Documentation

PyFields is fully documented inline. Your preferred IDE should be able to give you access to the documentation.

Full documentation is also available on ReadTheDocs.

CHAPTER 2

Installation

Just run:

```
python setup.py install
```

PyFields has some external dependencies (Matplotlib, Scipy, Unum and Numpy) that should be installed automatically.

You can run the tests with:

```
python setup.py pytest
```

CHAPTER 3

Basic usage

Coming soon

CHAPTER 4

Report/contribute

PyFields is hosted on Github, you can report an issue, ask your questions or contribute to this project there.

CHAPTER 5

License

This project is free software: You can redistribute it and/or modify it under the terms of the GNU General Public License, either version 3 of the License, or (at your option) any later version.

5.1 PyFields - 2D and 3D Fields Analysis in Python

PyFields is a Python3 package, intending to ease the manipulation and analysis of 2D and 3D fields, like velocity, concentration, temperature, ...

It can handle sets of fields, to analyze time evolution of the aforementioned fields. It includes easy-to-use helpers for importing, analyzing, displaying and exporting fields.

5.1.1 Documentation

PyFields is fully documented inline. Your preferred IDE should be able to give you access to the documentation.

Full documentation is also available on ReadTheDocs.

5.1.2 Installation

Just run:

```
python setup.py install
```

PyFields has some external dependencies (Matplotlib, Scipy, Unum and Numpy) that should be installed automatically.

You can run the tests with:

```
python setup.py pytest
```

5.1.3 Basic usage

Coming soon

5.1.4 Report/contribute

PyFields is hosted on Github, you can report an issue, ask your questions or contribute to this project there.

5.1.5 License

5.2 API

5.2.1 Classes

5.2.1.1 ScalarField class

class PyFields.core.scalarfield.**ScalarField**(*axis_x*, *axis_y*, *values*, *mask=None*, *unit_x=''*, *unit_y=''*, *unit_values=''*)

Bases: *PyFields.core.field.Field*

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

change_unit(*axis*, *new_unit*)

Change the unit of an axis.

Parameters

- **axis** (*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 scalar field.

Parameters

- **intervx** (*array, optional*) – Wanted interval along x.
- **intervy** (*array, optional*) – Wanted interval 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, else (default), a copy is returned.

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='values'*, *kind='imshow'*, *annotate=True*, ***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).
- **annotate** (*boolean*) – If True (default) add label and legedn to the graph.
- ****plotargs** (*dict*) – Arguments passed to the plotting function.

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

Return type figure reference

dtype

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

Parameters

- **nmb_right, nmb_up, nmb_down** (*nmb_left,*) – Number of lines/columns to add in each direction.
- **inplace** (*bool*) – If 'True', extend the field in place, else (default), return an extended copy of the field.

Returns **Extended_field** – Extended field.

Return type Field object

Note: If the axis values are not equally spaced, a linear extrapolation is used to obtain the new axis values.

---

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

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(*bins=200*, *cum=False*, *normalized=False*)
Return the scalarfield values histogram.

Parameters

- **cum** (*boolean*) – If True, get a cumulative histogram.
- **normalized** (*boolean*) – If True, normalize the histogram.
- **bins** (*integer*) – Number of bins (default to 200).

Returns **hist** – Histogram.

Return type Profile object.

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

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

5.2.1.1.1 Example

```
>>> interp = SF.get_interpolator(interp='linear')
>>> print(interp(4, 5.3))
... [34.3]
>>> print(interp([3, 4, 5], 5.3))
... [23, 34.3, 54]
```

get_profile(*x=None*, *y=None*, *ind=False*, *interp='linear'*)
Return a profile of the scalar field, at the given position. If position is an interval, the fonction return an average profile in this interval.

Parameters

- **y** (*x,*) – Position of the wanted profile.
- **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_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 axis (interpolated if necessary).

integrate()
Return the integral of the field. If you want the integral on a subset of the field, use 'crop' before.

Returns

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

Note: Discretized integral is computed with a very rustic algorithm which just sum the value on the surface.

make_evenly_spaced(*interp='linear'*, *res=1*, *inplace=False*)
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. A value of 2 means half this resolution.
- **inplace** (*boolean*) – If True, modify the scalar field in place, else, return a modified version of it.

mask

mask_as_sf

max

mean

min

reduce_resolution(*fact*, *inplace=False*)
Reduce the spatial resolution of the scalar 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 (signe in trigonometric convention). In order to preserve the orthogonal grid, only multiples of 90° are accepted.
- **inplace** (*boolean, optional*) – If True, the field is rotated in place, else (default), a rotated copy is returned.

Returns **rotated_field** – Rotated field.

Return type Field object

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.

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)

std

unit_values

values

5.2.1.2 VectorField class

class PyFields.core.vectorfield.**VectorField**(*axis_x*, *axis_y*, *comp_x*, *comp_y*, *mask=None*, *unit_x=''*, *unit_y=''*, *unit_values=''*)

Bases: *PyFields.core.field.Field*

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

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

Parameters

- **axis** (*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.

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 vector field.

Parameters

- **intervx** (*array, optional*) – Wanted interval along x.
- **intervy** (*array, optional*) – Wanted interval 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, else (default), a copy is returned.

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*, *annotate=True*, ***plotargs*)
Display the vector field.

Parameters

- **component** (*string, optional*) – Component to display, can be 'V', 'x', 'y', 'magnitude' or 'mask'
- **kind** (*string, optinnal*) – can be 'quiver', 'stream', 'imshow', 'contour', 'contourf'
- **annotate** (*boolean*) – If True (default) add label and legedn to the graph.
- ****plotargs** (*dict*) – Arguments passed to the plotting function.

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

Return type figure reference

dtype

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

Parameters

- **nmb_right, nmb_up, nmb_down** (*nmb_left,*) – Number of lines/columns to add in each direction.
- **inplace** (*bool*) – If 'True', extend the field in place, else (default), return an extended copy of the field.

Returns **Extended_field** – Extended field.

Return type Field object

Note: If the axis values are not equally spaced, a linear extrapolation is used to obtain the new axis values.

---

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

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 vector field 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_histograms (*bins=200*, *cum=False*, *normalized=False*)
Return a vectorfield component histograms.

Parameters

- **cum** (*boolean*) – If True, get a cumulative histogram.
- **normalized** (*boolean*) – If True, normalize the histogram.
- **bins** (*integer*) – Number of bins (default to 200).

Returns **histx, histy** – Histograms.

Return type Profile objects.

get_interpolators (*interp='linear'*)
Return the field interpolators.

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

Returns Interpolators for each components

Return type interpx, interpy

5.2.1.2.1 Example

```
>>> interp = SF.get_interpolator(interp='linear')
>>> print(interp(4, 5.3))
... [34.3]
>>> print(interp([3, 4, 5], 5.3))
... [23, 34.3, 54]
```

get_norm (*norm=2*)
Return the field norm.

Parameters **norm** (*positive integer*) – Norm order.

Returns **norm** – Norm.

Return type number

get_profile (*x=None*, *y=None*, *component='x'*, *ind=False*, *interp='linear'*)
Return a profile of the vector field, at the given position. If position is an interval, the fonction return an average profile in this interval.

Parameters

- **y** (*x,*) – Position of the wanted profile.
- **component** (*string in ['x', 'y']*) – Component to get a profile from. (default to 'x')
- **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 VectorField main properties

get_value(*x*, *y*, *ind=False*, *unit=False*)
Return the vector field components on the point (x, y).

Parameters

- **y** (*x,*) – Positions where to get the components.
- **ind** (*boolean*) – If 'True', x and y are indices, else (default), x and y are in axis units
- **unit** (*boolean*) – If 'True', also return the components unities. (default to False)

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*, *inplace=False*)
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. A value of 2 means half this resolution.
- **inplace** (*boolean*) – If True, modify the vector field in place, else, return a modified version of it.

mask

mask_as_sf

max

mean

min

reduce_resolution(*fact*, *inplace=False*)
Reduce the spatial resolution of the vector 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 (signe in trigonometric convention). In order to preserve the orthogonal grid, only multiples of 90° are accepted.
- **inplace** (*boolean, optional*) – If True, the field is rotated in place, else (default), a rotated copy is returned.

Returns **rotated_field** – Rotated field.

Return type Field object

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.

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)

std

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

5.2.2 Virtual classes

5.2.2.1 Field class

class PyFields.core.field.**Field**(*axis_x*, *axis_y*, *unit_x=''*, *unit_y=''*)
Bases: object

axis_x

axis_y

change_unit(*axis*, *new_unit*)
Put a field axis in the wanted unit. Change the axis value in agreement with the new unit.

Parameters

- **axis** (*string in ['x', 'y']*) – Axis to change the unit for.
- **new_unit** (*Unum.unit object or string*) – New unit.

Note: To associate a completely different unit to an axis (e.g. 'm' to 's'), use 'Field.axis_x = "s"'.

copy()
Return a copy of the Field object.

crop(*intervx=None*, *intervy=None*, *full_output=False*, *ind=False*, *inplace=False*)
Crop the field.

Parameters

- **intervx** (*array, optional*) – Wanted interval along x.
- **intervy** (*array, optional*) – Wanted interval along y.
- **full_output** (*boolean, optional*) – If 'True', cutting indices are also 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, else (default), a copy is returned.

dx

dy

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

Parameters

- **nmb_right, nmb_up, nmb_down** (*nmb_left,*) – Number of lines/columns to add in each direction.
- **inplace** (*bool*) – If 'True', extend the field in place, else (default), return an extended copy of the field.

Returns **Extended_field** – Extended field.

Return type Field object

Note: If the axis values are not equally spaced, a linear extrapolation is used to obtain the new axis values.

get_indice_on_axis(*direction*, *value*, *kind='bounds'*)
Return, on the given axis, the indices of the positions surrounding the given value.

Parameters

- **direction** (*string in ['x', 'y']*) – Axis choice.
- **value** (*number*) –
- **kind** (*string, optional*) – If 'bounds' (default), return the bounding indices. if 'nearest', return the nearest indice if 'decimal', return a decimal indice (interpolated)

Returns **interval** – Bounding, nearest or decimal indice.

Return type 2x1 array of integer or integer

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

Parameters

- **angle** (*integer*) – Angle in degrees (signe in trigonometric convention). In order to preserve the orthogonal grid, only multiples of 90° are accepted.
- **inplace** (*boolean, optional*) – If True, the field is rotated in place, else (default), a rotated copy is returned.

Returns **rotated_field** – Rotated field.

Return type Field object

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

Parameters

- **scaley** (*scalex,*) – Scale to apply on the associated axis
- **inplace** (*boolean, optional*) – If True, scale the field in place, else (default) return a scaled copy.

set_origin(*x=None*, *y=None*)
Modify the axis in order to place the origin at the given point (x, y).

Parameters **y** (*x,*) – Position of the new origin.

shape

unit_x

unit_y

5.2.3 Utilities

5.2.3.1 Units module

PyFields.utils.units.**make_unit**(*string*)
Create an Unum unit from a string. For more details, see the Unum module documentation.

Parameters **string** (*string*) – String representing some units (e.g. 'kg.m^-2.s^-1', or 'kg/m^2/s').

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

Return type unum.Unum object

5.2.3.1.1 Examples

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

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index