so-fronts

Simon Thomas

Nov 02, 2021

CONTENTS:

In this work, we present two related methods for calculating fronts from oceanographic data. fronts, with the advantage that it can highlight the movement of individual eddy-like features (such as the Agulhas rings). We will present our python GitHub repository, which will allow researchers to easily apply these methods to their own datasets.

Paper: https://doi.org/10.5194/os-17-1545-2021

Preprint: https://doi.org/10.5194/os-2021-40

Presentation at AGU2021: https://doi.org/10.1002/essoar.10507114.1 ; https://youtu.be/J8TMZlteN9s

CHAPTER

ONE

SRC PACKAGE

1.1 Subpackages

1.1.1 src.data_loading package

Submodules

src.data_loading.bsose_download module

Download the data from dropbox links.

'get_zip' 1694.64639 s

'un_zip' 444.41240 s

'clean_up' 4.04602 s

'get_and_unzip' 2143.10502 s

Example

Import statement:: from src.data_loading.bsose_download import get_data

Run statement:: get_data()

`src.data_loading.bsose_download.`**`get_and_unzip`**`(`*direc*, *url*, *name*`)`
Get the data and unzip it.

Parameters

- **direc** (*str*) – directory to put the data in.
- **url** (*str*) – url of the zip file.
- **name** (*str*) – name of file.

Return type None

`src.data_loading.bsose_download.`**`get_data`**`()`
Downloads data.

Return type None

src.data_loading.io_names module

Defaults for naming files.

src.data_loading.io_names.**return_folder**(*k_clusters*, *pca_components*)
Return return folder name.

Parameters

- **k_clusters** (*int*) – The number of classes.
- **pca_components** (*int*) – The number of pcas.

Returns file names.

Return type str

src.data_loading.io_names.**return_name**(*k_clusters*, *pca_components*)
Return name.

Parameters

- **k_clusters** (*int*) – The number of classes.
- **pca_components** (*int*) – The number of pcas.

Returns file names.

Return type str

src.data_loading.io_names.**return_pair_i_metric**(*k_clusters=5*, *pca=3*, *save_nc=True*, *t_index=40*)
Return pair i metric.

Parameters

- **k_clusters** (*int, optional*) – Number of clusters. Defaults to cst.K_CLUSTERS.
- **pca** (*int, optional*) – Number of principal components. Defaults to cst.D_PCS.
- **save_nc** (*bool, optional*) – Whether or not to save the resulting dataset. Defaults to True.
- **t_index** (*int, optional*) – time index cst.EXAMPLE_TIME_INDEX.

Returns pair i metric.

Return type xr.DataArray

src.data_loading.io_names.**return_plot_folder**(*k_clusters*, *pca_components*)
Return plot folder name.

Parameters

- **k_clusters** (*int*) – The number of classes.
- **pca_components** (*int*) – The number of pcas.

Returns file names.

Return type str

src.data_loading.xr_loader module

Xarray values.

src.data_loading.xr_loader.**order_indexes**(*dataarray*, *index_list*)
Order indices.

Goes from a datarray to a numpy array, ideally guaranteeing that the ordering of the numpy array is the same as would be expected.

Parameters

- **dataarray** (*xr.DataArray*) – [description]
- **index_list** (*list*) – [description]

Returns the numpy array which has been correctly ordered.

Return type np.ndarray

Module contents

1.1.2 src.models package

Submodules

src.models.batch_i_metric module

Make i metric in a batch.

Example

Usage:: python3 src/models/batch_i_metric.py

src.models.batch_i_metric.**merge_and_save**(*k_clusters=5*, *pca=3*)
Merge and save joint.

Return type None

src.models.batch_i_metric.**pca_from_interpolated_year**(*pcm_object*, *pca='Depth'*, *k_clusters=5*, *time_i=40*, *max_depth=2000*, *remove_init_var=True*)
[summary]

[extended_summary]

Parameters

- **pcm_object** (*pyxpcm.pcm*) – the pcm object which has already been trained.
- **pca** (*int, optional*) – How many principal components were chosen to be fitted. Defaults to cst.D_COORD.
- **k_clusters** (*int, optional*) – how many Guassians were fitted. Defaults to cst.K_CLUSTERS.
- **time_i** (*int, optional*) – [description]. Defaults to cst.EXAMPLE_TIME_INDEX.
- **max_depth** (*float, optional*) – The maximum_depth (in pcm_object) that the data is fitted to. Defaults to cst.MAX_DEPTH.
- **remove_init_var** (*bool, optional*) – Whether or not to remove the initial variables. Defaults to True.

Return type None

src.models.batch_i_metric.**run_through**()
Run through.

Return type None

src.models.batch_i_metric.**run_through_sep**(*k_clusters=5*, *pca=3*)
Run through joint.

Parameters

- **k_clusters** (*int, optional*) – [description]. Defaults to 5.
- **pca** (*int, optional*) – [description]. Defaults to 3.

Return type None

src.models.make_pair_metric module

To pair i metric.

src.models.make_pair_metric.**make_all_pair_i_metric**(*cart_prod*, *i_metric*, *sorted_version*, *threshold*)
Make all pair i metric.

Parameters

- **cart_prod** (*list*) – cartesian product
- **i_metric** (*np.ndarray*) – i metric.
- **sorted_version** (*np.ndarray*) – [description]
- **threshold** (*float*) – [description]

Returns tuple.

Return type Tuple[list]

src.models.make_pair_metric.**make_one_pair_i_metric**(*pair*, *i_metric*, *sorted_version*, *threshold*)
Make a pair i metric.

Parameters

- **pair** (*tuple*) – [description]
- **i_metric** (*np.ndarray*) – [description]
- **sorted_version** (*np.ndarray*) – [description]
- **threshold** (*float*) – threshold to nan things out below.

Returns list of numpy arrrays.

Return type Sequence[np.array]

src.models.make_pair_metric.**pair_i_metric**(*ds*, *threshold=0.05*)
Pair i metric.

new loading order (to be changed) ds.A_B.values.shape (2, 12, 60, 240) sorted_version.shape (2, 12, 60, 240) i_metric (12, 60, 240) list_no [0, 1, 2, 3, 4] https://numpy.org/doc/stable/reference/generated/numpy.swapaxes.html https://numpy.org/doc/stable/reference/generated/numpy.moveaxis.html "time"].values.shape[0]), ("rank", dataarray.coords["rank"].values.shape[0]), ("x", dataarray.coords["XC"].values.shape[0]), ("y", dataarray.coords["YC"]

Parameters

- **ds** (*xr.Dataset*) – dataset.
- **threshold** (*float, optional*) – threshold to nan out below. Defaults to 0.05.

Returns pair i metric dataset.

Return type xr.DataArray

src.models.sobel module

Test sobel vs gradient.

src.models.sobel.**grad_v**()
Gradient in v direction.

Return type None

src.models.sobel.**sobel_np**(*values*)
Sobel operator on np array.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve2d.html

Parameters **values** (*np.ndarray*) – values to differentiate.

Returns gx, gy

Return type Tuple[np.ndarray, np.ndarray]

src.models.sobel.**sobel_scharr_test**()
Test scharr / sobel.

Return type None

src.models.sobel.**sobel_vs_grad**()
Sobel versus dimension.

Return type None

src.models.train_pyxpcm module

Train i metric.

Example

To test:: python3 src/models/train_pyxpcm.py

src.models.train_pyxpcm.**train_on_interpolated_year**(*time_i=40*, *k_clusters=5*, *maxvar=3*, *min_depth=300*, *max_depth=2000*, *remove_init_var=True*, *separate_pca=False*, *interp=True*, *remake=False*)
Train on interpolated year.

Parameters

- **time_i** (*int, optional*) – time index. Defaults to cst.EXAMPLE_TIME_INDEX.
- **k_clusters** (*int, optional*) – clusters. Defaults to cst.K_CLUSTERS.
- **maxvar** (*int, optional*) – num pca. Defaults to cst.D_PCS.
- **min_depth** (*float, optional*) – minimum depth for column. Defaults to cst.MIN_DEPTH.
- **max_depth** (*float, optional*) – maximum depth for column. Defaults to cst.MAX_DEPTH.
- **separate_pca** (*bool, optional*) – separate the pca. Defaults to True.
- **remove_init_var** (*bool, optional*) – remove initial variables. Defaults to True.

Returns the fitted object and its corresponding dataset.

Return type Tuple[pyxpcm.pcm, xr.Dataset]

Module contents

1.1.3 src.plot package

Submodules

src.plot.clust_3d module

The purpose of this is the algorithm in 3D.

src.plot.clust_3d.**comp_3d**(*weights*, *means*, *covariances*, *ds*)
This will hopefully plot fig2a and fig2b with automatic labelling.

Parameters

- **weights** (*np.ndarray*) – weights numpy array.
- **means** (*np.ndarray*) – means numpy array.
- **covariances** (*np.ndarray*) – covariances numpy array.
- **ds** (*xr.Dataset*) – dataset.

Return type None

src.plot.i_map module

The purpose of this is to visualise the i_metric on the southern ocean map.

src.plot.i_map.**map_imetric**(*da_i*, *da*)
Plot map i metric clusters.

Returns void (although matplotlib will be storing the figure).

Parameters

- **da_i** (*xr.DataArray*) – xarray.dataarray object.
- **da** (*xr.DataArray*) – xarray.dataarray object.

Return type None

src.plot.profiles module

Cluster profiles plots.

src.plot.profiles.**make_profiles**(*ds*)
Make cluster profiles (mean and std dev).

Parameters **ds** (*xr.Dataset*) – the dataset.

Returns new_dataset to plot.

Return type xr.Dataset

src.plot.profiles.**plot_profiles**(*ds*)
Plot profiles.

Originally from: https://scitools.org.uk/iris/docs/v1.6/examples/graphics/atlantic_profiles.html A program to plot profiles, originally of the original components etc. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.plot.html There's a fair deal of duplication in this function. Could probably half its length without changing its functionality.

Parameters **ds** (*xr.Dataset*) – Profile dataset to plot.

Return type None

Module contents

1.1.4 src.plot_utils package

Submodules

src.plot_utils.colors module

Color utilities.

src.plot_utils.colors.**cluster_cmap**(*number_clusters*)
Cluster cmap.

Parameters **number_clusters** (int) – The number of clusters.

Returns cmap.

src.plot_utils.colors.**cluster_colors**(*number_clusters*)
Cluster colors.

Parameters **number_clusters** (int) – The number of clusters.

Return type ndarray

Returns np.ndarray

src.plot_utils.colors.**fading_colormap**(*from_color*, *fade_to_white=True*)
Takes a hex or color name, and returns a fading color map.

example usage:

cmap_a = fading_colormap('blue') # cmap_b = fading_colormap('#96f97b')

Parameters **from_color** (str) – either a hex or a name

Returns cmap –> a colormap that can be used as a parameter in a plot.

src.plot_utils.colors.**replacement_color_list**(*number_of_colors*)
Replacement color list.

Parameters **number_of_colors** (int) –

Return type list

Returns

src.plot_utils.colors.**return_list_of_colormaps**(*number*, *fade_to_white=True*)
Retunr list of colormaps.

Parameters

- **number** (*int*) – number of colormaps needed.
- **fade_to_white** (*bool, optional*) – Whether or not. Defaults to True.

Returns list of colormaps.

Return type list

src.plot_utils.ellipses module

Ellipses.

src.plot_utils.ellipses.**ellispes**(*pcm*, *ax1*)
Plot ellipses.

Parameters

- **pcm** (*pyxpcm.pcm*) – pcm object.
- **ax1** (*matplotlib.axes.Axes*) – axes.

Return type None

src.plot_utils.ellipses.**plot_ellipsoid**(*fig*, *ax*, *covariance_matrix*, *mean*, *weight*, *color*, *print_properties=False*, *additional_rotation=array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])*)
A function for drawing 3d-multivariate guassians with method initially from: https://stackoverflow.com/questions/7819498/plotting-ellipsoid-with-matplotlib

Parameters

- **fig** (Figure) – The figure matplotlib.pyplot object
- **ax** (Axes) – The axis matplotlib.pylot object with Axes3D extension
- **covariance_matrix** (array) – A covariance matrix input from the multivariate guassian to be plotted.
- **mean** (array) – ditto
- **weight** (array) – ditto
- **color** (any) – ditto

Return type Tuple[Figure, Axes]

Returns fig and ax so that they can be used by further plotting steps.

src.plot_utils.ellipses.**plot_ellipsoid_test**()
Plot ellipsoid test. Runs from unittest.

Return type None

src.plot_utils.gen_panels module

Label subplots. is the main function

src.plot_utils.gen_panels.**label_subplots**(*axs*, *labels=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']*, *start_from=0*, *fontsize=10*, *x_pos=0.02*, *y_pos=0.95*)
Adds e.g. (a), (b), (c) at the top left of each subplot panel.

Labelling order achieved through ravelling the input *list* or *np.array*.

Parameters

- **axs** (*Sequence[matplotlib.axes.Axes]*) – *list* or *np.array* of *matplotlib.axes.Axes*.
- **labels** (*Sequence[str]*) – A sequence of labels for the subplots.
- **start_from** (*int, optional*) – skips first *start_from* labels. Defaults to 0.
- **fontsize** (*int, optional*) – Font size for labels. Defaults to 10.
- **x_pos** (*float, optional*) – Relative x position of labels. Defaults to 0.02.
- **y_pos** (*float, optional*) – Relative y position of labels. Defaults to 0.95.

Return type None

Returns void; alters the *matplotlib.axes.Axes* objects

Example

Here is an example of using this function::

```
>>> label_subplots(axs, start_from=0, fontsize=10)
```

src.plot_utils.ko_plot module

A program by sdat2 to plot the Southern Ocean with a variety of fronts. Currently plots Kim and Orsi 2014 (KO).

Usage:

Example usage:: ko.draw_fronts_kim(ax)

src.plot_utils.ko_plot.**draw_fronts_kim**(*ax*)
A function to read the Kim and Orsi (2014) data and plot it on SO.

Now also includes the data from Kim (c.1995) for the STF.

Parameters **ax** (*matplotlib.axes.Axes*) – draw the kim fronts on this axis.

Return type None

src.plot_utils.ko_plot.**is_too_far**(*lat_a=0.0*, *lat_b=0.0*, *lon_a=0.0*, *lon_b=0.0*, *max_allowable_square=1.0*)
Check if points are too far apart to draw a line between.

Parameters

- **lat_a** (*float, optional*) – [description]. Defaults to 0.0.
- **lat_b** (*float, optional*) – [description]. Defaults to 0.0.
- **lon_a** (*float, optional*) – [description]. Defaults to 0.0.
- **lon_b** (*float, optional*) – [description]. Defaults to 0.0.
- **max_allowable_square** (*float, optional*) – Max allowable square. Defaults to 1.

Returns whether or not.

Return type bool

src.plot_utils.ko_plot.**plot_list_of_lists**(*ax*, *lol_of_xs=[[0.0], [0.0]]*, *lol_of_ys=[[0.0], [0.0]]*, *color='red'*, *markersize=0.3*, *label='UNLABELED'*, *line_type='-'*)
Matlplotlib cannot plot a list of lists Addresses none of our questsions, makes no prediction, and cannot be falsified. If one's prediction can't predict anything, it is just wrong, and one must try something else.

Parameters

- **ax** (*matplotlib.axes.Axes*) – [description]
- **lol_of_xs** (*List[list], optional*) – [description]. Defaults to [[0.0], [0.0]].
- **lol_of_ys** (*List[list], optional*) – [description]. Defaults to [[0.0], [0.0]].
- **color** (*str, optional*) – [description]. Defaults to "red".
- **markersize** (*float, optional*) – [description]. Defaults to 0.3.
- **label** (*str, optional*) – [description]. Defaults to "UNLABELED".
- **line_type** (*str, optional*) – [description]. Defaults to "-".

Return type None

src.plot_utils.ko_plot.**run_so_map**()
Run through and plot.

Return type None

src.plot_utils.ko_plot.**split_into_list_of_lists**(*max_square=1*, *list_of_xs=[0.0]*, *list_of_ys=[0.0]*)
Split into list of lists.

Parameters

- **max_square** (*float, optional*) – [description]. Defaults to 1.
- **list_of_xs** (*list, optional*) – [description]. Defaults to [0.0].
- **list_of_ys** (*list, optional*) – [description]. Defaults to [0.0].

Returns list of lists (lol) for Xs and Ys

Return type Tuple[List[list], List[list]]

src.plot_utils.latex_style module

Plotting style file.

import src.plot_utils.latex_style as lsty

usage:

ds = xr.open_dataset('example.nc')

lsty.ds_for_grahing(ds).plot()

src.plot_utils.latex_style.**ds_for_graphing**(*input_da*)
Transform dataset for graphing.

Parameters **input_da** (*xr.Dataset*) – dataest input

Returns transformed dataset.

Return type xr.Dataset

src.plot_utils.latex_style.**get_dim**(*width=398.3386*, *fraction_of_line_width=1*, *ratio=0.6180339887498949*)
Return figure height, width in inches to avoid scaling in latex.

Default width is *src.constants.REPORT_WIDTH*. Default ratio is golden ratio, with figure occupying full page width.

Parameters

- **width** (*float, optional*) – Textwidth of the report to make fontsizes match. Defaults to *src.constants.REPORT_WIDTH*.
- **fraction_of_line_width** (*float, optional*) – Fraction of the document width which you wish the figure to occupy. Defaults to 1.
- **ratio** (*float, optional*) – Fraction of figure width that the figure height should be. Defaults to (5 ** 0.5 - 1)/2.

Returns Dimensions of figure in inches

Return type fig_dim (tuple)

Example

Here is an example of using this function::

```
>>> dim_tuple = get_dim(fraction_of_line_width=1, ratio=(5 ** 0.5 - 1) / 2)
```

src.plot_utils.latex_style.**mpl_params**(*quality='high'*, *use_tex=True*, *dpi=600*)

Apply plotting style to produce nice looking figures.

Call this at the start of a script which uses *matplotlib*. Can enable *matplotlib* LaTeX backend if it is available.

Parameters

- **use_tex** (*bool, optional*) – Whether or not to use latex matplotlib backend. Defaults to True.
- **dpi** (*int, optional*) – Which dpi to set for the figures. Defaults to 600 dpi (high quality). 150 dpi probably fine for notebooks. Largest dpi needed for presentations.

Examples

Basic setting the plotting defaults::

```
>>> mpl_params()
```

Setting defaults for a jupyter notebook::

```
>>> mpl_params(use_tex=False, dpi=150)
```

Return type None

src.plot_utils.latex_style.**proper_units**(*text*)

Function for changing units to a better form.

Parameters **text** (*str*) – text to check.

Returns reformatted text with better units.

Return type str

src.plot_utils.latex_style.**set_dim**(*fig*, *width=398.3386*, *fraction_of_line_width=1*, *ratio=0.6180339887498949*)

Set aesthetic figure dimensions to avoid scaling in latex.

Default width is *src.constants.REPORT_WIDTH*. Default ratio is golden ratio, with figure occupying full page width.

Parameters

- **fig** (*matplotlib.figure.Figure*) – Figure object to resize.
- **width** (*float*) – Textwidth of the report to make fontsizes match. Defaults to *src.constants.REPORT_WIDTH*.
- **fraction_of_line_width** (*float, optional*) – Fraction of the document width which you wish the figure to occupy. Defaults to 1.
- **ratio** (*float, optional*) – Fraction of figure width that the figure height should be. Defaults to (5 ** 0.5 - 1)/2.

Return type None

Returns void; alters current figure to have the desired dimensions

Example

Here is an example of using this function::

```
>>> set_dim(fig, fraction_of_line_width=1, ratio=(5 ** 0.5 - 1) / 2)
```

src.plot_utils.latex_style.**tex_escape**(*text*)
It is better to plot in TeX, but this involves escaping strings.

from: https://stackoverflow.com/questions/16259923/ how-can-i-escape-latex-special-characters-inside-django-templates :param text: a plain text message :return: the message escaped to appear correctly in LaTeX

 # removed unicode(key) from re.escape because this seemed an unnecessary, and was throwing an error.

Return type str

src.plot_utils.map module

map.py by sdat2 - the different maps options.

southern_ocean_axes_setup - SO - up to 30 deg South to 90 degrees south.

src.plot_utils.map.**southern_ocean_axes_setup**(*ax*, *fig*, *add_gridlines=True*)
This function sets up the subplot so that it is a cartopy map of the southern ocean.

returns void as the ax and figure objects are pointers not data.

Parameters

- **ax** (*matplotlib.axes.Axes*) – The axis object to add the map to.
- **fig** (*matplotlib.figure.Figure*) – The figure object for the figure in general.
- **add_gridlines** (*bool*) – whether or not to add gridlines to the plot.

Return type None

src.plot_utils.xarray_panels module

Xarray panels scripts.

src.plot_utils.xarray_panels.**plot_several_pair_i_metrics**(*da_list*, *plot_ko=False*)
Plot several pair i metrics.

USAGE: plot_several_pair_i_metrics([run_through_plot(K=2).isel(time=0),

run_through_plot(K=4).isel(time=0)])

plt.tight_layout() plt.savefig(

"../FBSO-Report/images/fig5-new.png", dpi=900, bbox_inches="tight"

)

Parameters

- **da_list** (*Sequence[xr.DataArray]*) – list of dataarrays.
- **plot_ko** (*bool*) – whether or not to plot the kim and orsi 2014 fronts.

Return type None

src.plot_utils.xarray_panels.**plot_single_i_metric**(*da*)
Plot single i metric plot with viridis colormap.

Parameters **da** (*xr.DataArray*) – i metric dataarray.

Return type None

src.plot_utils.xarray_panels.**sep_plots**(*da_list*, *var_list*, *min_max_list=None*, *cmap_list=None*)
Separate plots.

Parameters

- **da_list** (*Sequence[xr.DataArray]*) – list of xr.DataArray.
- **var_list** (*List[str]*) – list of variable names.
- **min_max_list** (*Union[List[list], any], optional*) – vmin and vmax. Defaults to None.
- **cmap_list** (*Union[List[list], any], optional*) – vmin and vmax. Defaults to None.

Return type None

Module contents

1.1.5 src.preprocessing package

Submodules

src.preprocessing.gsw_transformations module

Module contents

1.2 Submodules

1.3 src.animate module

Animate da.

src.animate.**animate_imetric**(*video_path='output.gif'*, *k_clusters=5*)
Animate an *xr.DataArray*.

Parameters

- **video_path** (*str, optional*) – Video path. Defaults to “output.mp4”.
- **k_clusters** (*int, opitonal*) – k clusters. Defaults to cst.K_CLUSTERS.

Return type None

1.4 src.constants module

Constants.py program for storing paths and variables names.

1.5 src.make_figures module

Make figures: run through all the paper figures and make them.

Takes roughly 5 minutes the first time it is run.

src.make_figures.**make_all_figures**()
Make all the figures in the paper in a sequence.

Takes roughly 15 minutes on jasmin.

Return type None

1.6 src.move_figures module

Move figures.

src.move_figures.**move**(*copy_command='cp'*)
Move the files to the project.

Parameters **copy_command** (*str, optional*) – which command is needed to move files in operating system. Defaults to "cp".

Return type None

1.7 src.time_wrapper module

Times utilities.

src.time_wrapper.**timeit**(*method*)
This timeit function is a wrapper for performance analysis.

Should return the time taken for a function to run, :type method: Callable :param method: the function that it takes as an input :rtype: Callable :return: timed

Example

Usage example:: import src.time_wrapper as twr @twr.timeit

1.8 Module contents

CHAPTER

TWO

DEFINING SOUTHERN OCEAN FRONTS USING UNSUPERVISED CLASSIFICATION

2.1 Paper: https://doi.org/10.5194/os-17-1545-2021

2.2 Preprint: https://doi.org/10.5194/os-2021-40

2.3 Presentation at AGU2021: https://doi.org/10.1002/essoar.10507114.1

2.4 Short description

In the Southern Ocean, fronts delineate water masses, which correspond to upwelling and downwelling branches of the overturning circulation. Classically, oceanographers define Southern Ocean fronts as a small number of continuous linear features that encircle Antarctica. However, modern observational and theoretical developments are challenging this traditional framework to accommodate more localized views of fronts [Chapman et al. 2020].

Here we present code for implementing two related methods for calculating fronts from oceanographic data. The first method uses unsupervised classification (specifically, Gaussian Mixture Modeling or GMM) and a novel interclass metric to define fronts. This approach produces a discontinuous, probabilistic view of front location, emphasising the fact that the boundaries between water masses are not uniformly sharp across the entire Southern Ocean.

The second method uses Sobel edge detection to highlight rapid changes [Hjelmervik & Hjelmervik, 2019]. This approach produces a more local view of fronts, with the advantage that it can highlight the movement of individual eddy-like features (such as the Agulhas rings).

1. Chapman, C. C., Lea, M.-A., Meyer, A., Sallee, J.-B. & Hindell, M. Defining Southern Ocean fronts and their influence on biological and physical processes in a changing climate. Nature Climate Change (2020). https://doi.org/10.1038/s41558-020-0705-4
2. Maze, G. et al. Coherent heat patterns revealed by unsupervised classification of Argo temperature profiles in the North Atlantic Ocean. Progress in Oceanography (2017). https://doi.org/10.1016/j.pocean.2016.12.008, https://doi.org/10.5281/zenodo.3906236
3. Hjelmervik, K. B. & Hjelmervik, K. T. Detection of oceanographic fronts on variable water depths using empirical orthogonal functions. IEEE Journal of Oceanic Engineering (2019). https://doi.org/10.1109/JOE.2019.2917456

2.5 I metric for K=5

2.6 Getting started

- Make the environment:

```
make env
```

- Activate the environment in conda:

```
conda activate ./env
```

- Change the settings in `src.constants` to set download location etc.
- Download data (`get_zip`: 1694.64639 s):

```
python3 src/data_loading/bsose_download.py
```

- Make I-metric:

```
python3 src/models/batch_i_metric.py
```

- Make figures:

```
python3 main.py
```

2.7 Project Organization

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make env` or `make `
├── README.md          <- The top-level README for developers using this project.
├── main.py            <- The main python script to run.
│
├── figures            <- .png images with non-enumerated names.
│
├── requirements       <- Directory containing the requirement files.
│
├── setup.py           <- makes project pip installable (pip install -e .) so src␣
↪can be imported from jupyter notebooks etc.
│
├── src                <- Source code for use in this project.
│   │
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- KO fronts to plot, other data.
│   │
│   ├── data_loading   <- Scripts to download and name data.
│   │
│   ├── models         <- Make I-metric and Sobel edge detection directory.
│   │
│   ├── plot           <- plotting functions directory
│   │
```

(continues on next page)

(continued from previous page)

```
│   ├── plot_utils       <- plotting utilities directory
│   │
│   ├── preprocessing    <- preprocessing scripts (to transform to density etc.).
│   │
│   ├──  tests           <- Scripts for unit tests of your functions
│   │
│   ├── animate.py       <- animate i-metric.
│   ├── constants.py     <- contains majority of run parameters that can be changed.
│   ├── make_figures.py  <- make all figures in one long script.
│   ├── move_figures.py  <- Move figures script (now unnecessary).
│   │                       Changes figure names to Figure-X.png etc.
│   └── time_wrapper.py  <- time wrapper to time parts of the program.
│
└── setup.cfg            <- setup configuration file for linting rules
```

2.8 Requirements

- Anaconda, with `conda` working in shell.
- `make` in shell.
- Python 3.6+ (final run for paper used `python==3.8.8`)

Project template created by the Cambridge AI4ER Cookiecutter.

CHAPTER

THREE

ANIMATIONS GALLERY

Animations of the I-metric for the full time-period of BSOSE-i106. Choosing different K values, the initial period includes the spin-up of the reanalysis product, and so is not typical of the full period.

3.1 I metric for K=5

3.2 I metric for K=4

3.3 I metric for K=2

CHAPTER

FOUR

CURRENT STATE OF CODE

```
$ cloc --report-file=docs/lang.txt --sum-one  $(git ls-files)
```

```
github.com/AlDanial/cloc v 1.84  T=0.13 s (452.2 files/s, 75619.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                          32            750            793           2463
Jupyter Notebook                 5              0           4220            953
Markdown                         8             76              0            195
reStructuredText                 8             83            145             99
make                             2             20             42             84
YAML                             3             10             15             51
Bourne Shell                     2              2             13             20
-------------------------------------------------------------------------------
SUM:                            60            941           5228           3865
-------------------------------------------------------------------------------
```

CHAPTER

FIVE

UNSUPERVISED CLASSIFICATION

5.1 Perception

One of the essential problems with unsupervised machine learning is that, despite what it might appear, there are an infinite number of ways to perceive the world, and none need be canonical. If you want to perceive the world, you first need to define some goal which the categorisation scheme can be allowed to maximise. In this particular repository, the goal was to maximise the expectation of the data given that we assume the data is made up of `K` Gaussians.

That this goal is analogous to our attempts to divide up the Southern Ocean is not obvious. In some sense, it might be easier to justify a categorisation of the ocean if its purpose was to predict some obvious physical variable, i.e. `supervised learning`, which provides either a clear cost function to optimise, or a series of similar cost functions that could be maximised.

5.2 Possible objectives

The fronts of the Southern Ocean could correspond to:

(a) The boundary between different water masses in the Southern Ocean.

- where bondaries are distinct masses of water with distinct histories.
- and there are a small and unambiguous number, `K`, of regions of water.

or

(b) The regions at which properties change suddenly, which could correspond to those points at which the thermal wind balance leads to a jet.

- A large gradient in density.
- A large gradient in potential vorticity.

or

(c) The regions where there is a great deal of mixing upwards, and therefore there is a greater ammount of biological productivity, leading to a source of food for the Southern Ocean ecosystem.

or

(d) The divisions between different distinct ecological communities.

(a) and (d), (b) and (c) are quite closely related.

CHAPTER

SIX

ABOUT

6.1 Co-authors

- Simon Thomas (BAS, U. Cam)
- Dan Jones (BAS)
- Anita Faul (BAS)
- Erik Mackie (BAS, U. Cam)
- Etienne Pauthenet (L'Ocean)

6.2 Funding

This work originated as a Natural Environment Research Council (NERC) Research Experience Placement (REP) project funded by the SPITFIRE Doctoral Training Partnership (reference NE/S007210/1). ST is supported by studentship 2413578 from the UKRI Centre for Doctoral Training in Application of Artificial Intelligence to the study of Environmental Risks (reference EP/S022961/1). DJ is supported by a UKRI Future Leaders Fellowship (reference MR/T020822/1). DJ and ST also received funding from the NERC ACSIS project (reference NE/N018028/1). EP received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement 637770).

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

R

S

T