octapy

Release 1.0

Jason Tilley

Apr 27, 2021

CONTENTS

CHAPTER

ONE

SUBMODULES

CHAPTER

TWO

OCTAPY.DATA MODULE

class octapy.data.**Data**

Bases: object

a Cython data container class

sal
salinity

ssh
sea surface height

temp
temperature

u
u-velocity (eastward velocity)

v
v-velocity (northward velocity)

w
w-velocity (upward velocity)

CHAPTER

THREE

OCTAPY.GET_DATA_AT_INDEX MODULE

octapy.get_data_at_index.**get_data_at_index**()

Cython function for getting data from a netCDF file at a given index location

Parameters

- **filepath** – path to the netCDF file
- **index** – the ravelled index to extract 3D variables in the netCDF file
- **surf_index** – the ravelled index to extract sea surface height in the netCDF file
- **dims** – the dimensions of the model

Returns a Data object instance

CHAPTER

FOUR

OCTAPY.INTERP_IDW MODULE

octapy.interp_idw.**interp_idw**()
a Cython inverse distance weighted interpolation function

Parameters

- **particle** – a Particle instance
- **data** – a Data instance
- **weights** – weights used for inverse distance weighted interpolation
- **dims** – dimensions of the Model instance

Returns a particle instance with interpolated environmental data

CHAPTER

FIVE

OCTAPY.TOOLS MODULE

octapy.tools.**get_filepath**(*datetime64*, *model_name*, *submodel_name*, *data_dir*)
Get the filename for a given timestep

Parameters

- **datetime64** – a numpy.datetime64 object for the data's time
- **model_name** – the oceanographic model being used

Returns string of the filepath

octapy.tools.**get_extent**(*grid*)
Get the spatial extent of a grid as a list

Parameters **grid** – a Grid instance

Returns a list of coordinates representing the extent as [minimum longitude, maximum longitude, minimum latitude, maximum latitude]

octapy.tools.**netcdf_to_csv**(*file_list*, *outfile*)
convert output netcdf files to a .csv file

Parameters

- **file_list** – a list of paths to the netcdf files to be converted
- **outfile** – path of the .csv file to be created

Returns None

octapy.tools.**plot_csv_output**(*file_list*, *extent*, *step=2*, *plot_type='lines'*, *colors=None*)
plot output trajectories contained in a .csv file

Parameters

- **file_list** – a list of paths to the .csv files to be plotted
- **extent** – a list of coordinates representing the extent as [minimum longitude, maximum longitude, minimum latitude, maximum latitude]

Returns None

octapy.tools.**plot_netcdf_output**(*file_list*, *extent*, *out_file=None*, *step=2*, *plot_type='lines'*, *colors=None*, *drifter=None*, *contour_file=None*, *contour_idx=0*)
plot output trajectories contained in netcdf files

Parameters

- **file_list** – a list of paths to the .csv files to be plotted
- **extent** – a list of coordinates representing the extent as [minimum longitude, maximum longitude, minimum latitude, maximum latitude]

- **out_file** – path for the output image file
- **step** – integer representing the number of trajectory points to plot (e.g, step=2 means you will show every two points)
- **plot_type** – type of plot for each trajectory, may be 'lines' or 'scatter'. Defaults is 'lines'.
- **colors** – list of python colors to customize line colors. Must be same length as file_list.
- **drifter** – path to .csv file containing drifter track containing columns 'lats' and 'lons'
- **contour_file** – path to netcdf file containing spatial temperature data
- **contour_idx** – index for the sigma level of temperature data to plot (0 = surface)

Returns None

octapy.tools.**build_skill_release**(*drifter_file*, *model*, *period=Timedelta('3 days 00:00:00')*, *data_freq=Timedelta('0 days 01:00:00')*)

Determine the model skill against a particular drifter

Parameters

- **drifter_file** – a .csv file of the drifter data with the columns 'datetime', 'lat', and 'lon'
- **model** – an initialized octapy.tracking.Model object
- **period** – a pandas Timedelta object representing the time period for which to calculate the skill
- **data_freq** – a pandas Timedelta object representing the frequency of the drifter data

Returns None

octapy.tools.**get_drifter_data**(*drifter_file*, *drifter_id*)

Gets drifter data from a given .csv file for a particular drifter id

Parameters

- **drifter_file** – a .csv file of the drifter data with the columns 'datetime', 'lat', and 'lon'
- **drifter_id** – the drifter id with the same type as in the drifter file

Returns a Pandas DataFrame

octapy.tools.**run_skill_analysis**(*drifter_file*, *drifter_id*, *skill_files*, *date_range*, *grid*, *period=Timedelta('3 days 00:00:00')*, *data_freq=Timedelta('0 days 01:00:00')*)

Calculate the skill score for a given particle run. Here, the skill score is calculated as in Liu and Weisberg (2011).

Parameters

- **drifter_file** – a .csv file of the drifter data with the columns 'datetime', 'lat', and 'lon'
- **skill_files** – a sorted list of the model output netCDF files that contain the tracks from the model runs for skill
- **date_range** – a numpy.ndarray object of numpy.datetime64 objects, the drifter and output data frequencies should match
- **period** – a pandas Timedelta object representing the time period for which to calculate the skill
- **data_freq** – a pandas Timedelta object representing the frequency of the drifter data

Returns a numpy array of particle times, trajectory lengths, separation distances, and skill scores

CHAPTER

SIX

OCTAPY.TRACKING MODULE

class octapy.tracking.**Model**(*release_file=None*, *model=None*, *submodel=None*, *data_dir='data'*, *direction=1*, *dims=2*, *diffusion=False*, *depth=None*, *extent=None*, *data_date_range=None*, *timestep=numpy.timedelta64(60, 'm')*, *data_freq=numpy.timedelta64(60, 'm')*, *data_timestep=numpy.timedelta64(60, 'm')*, *interp='linear'*, *leafsize=9*, *vert_migration=False*, *vert_array=None*, *output_file=None*, *output_freq=numpy.timedelta64(60, 'm')*)

Bases: object

A particle tracking Model object

Parameters

- **release_file** – a file containing the particle coordinates and release times
- **model** – name string of the ocean model used for input data (e.g, 'HYCOM')
- **submodel** – name string of the submodel and/or experiment used for input data (e.g, 'GOMl0.04/expt_31.0')
- **data_dir** – data directory path
- **direction** –

  forcing direction through time. Must be 1 for forward or -1 for backward
- **dims** – dimensionality of the model, must be 2 or 3
- **diffusion** – if true, enables diffusion
- **data_vars** – an numpy.ndarray of the variable names in the data file
- **depth** – forcing depth if the model is 2-dimensional
- **extent** – a list of extent coordinates as [minlat, maxlat, minlon, maxlon]
- **data_date_range** – a numpy.ndarray object of numpy.datetime64 objects containing the dates of the input data. WARNING: Must match frequency of data
- **timestep** – a numpy.timedelta64 object representing the timestep of the tracking model in minutes (e.g., np.timedelta64(60,'m'))
- **data_freq** – a numpy.timedelta64 object representing the frequency of data on the data server (e.g., np.timedelta64(60,'m'))
- **data_timestep** – a numpy.timedelta64 object representing the timestep of the data to be downloaded in minutes (e.g., np.timedelta64(60,'m'))
- **interp** – interpolation method. Supported are 'linear', 'nearest', 'splinef2d', and 'idw'.
- **leafsize** – number of nearest neighbors for some interpolation schemes

- **vert_migration** – if True, the particle will undergo daily vertical migrations which will override w velocities
- **vert_array** – an array of length 24 representing the depth of a particle over a 24-hour period when vert_mirgration is set to True
- **output_file** – base output file name
- **output_freq** – a numpy.timedelta64 object representing how often the particle data will be output to the output file in minutes (e.g., np.timedelta64(60,'m'))

Returns A Model object

__init__(*release_file=None*, *model=None*, *submodel=None*, *data_dir='data'*, *direction=1*, *dims=2*, *diffusion=False*, *depth=None*, *extent=None*, *data_date_range=None*, *timestep=numpy.timedelta64(60, 'm')*, *data_freq=numpy.timedelta64(60, 'm')*, *data_timestep=numpy.timedelta64(60, 'm')*, *interp='linear'*, *leafsize=9*, *vert_migration=False*, *vert_array=None*, *output_file=None*, *output_freq=numpy.timedelta64(60, 'm')*)

Initialize self. See help(type(self)) for accurate signature.

class octapy.tracking.**Grid**(*model*)

Bases: object

A Grid object. You must have already downloaded the data into the data directory.

Parameters **model** – Model instance to which the Grid instance will belong

Returns A Grid object

__init__(*model*)

Initialize self. See help(type(self)) for accurate signature.

src_crs

a cartopy.crs projection object representing the data's source projection (e.g., cartopy.crs.LambertCylindrical())

tgt_crs

a cartopy.crs projection object representing the model's target projection

file

name of the file from which the grid was produced.

depths

depths of the grid

lons

longitudes of the grid

lats

latitudes of the grid

x

the x coordinates of the grid in meters

y

the y coordinates of the grid in meters

points

an array of grid coordinates in meters

tree

a scipy.spatial.cKDTree instance representing the grid

class octapy.tracking.**Particle**(*num=None*, *lat=None*, *lon=None*, *depth=None*, *timestamp=None*, *x=None*, *y=None*, *u=None*, *v=None*, *w=None*, *temp=None*, *sal=None*, *ssh=None*, *filepath=None*)

Bases: object

A Particle object

Parameters

- **lat** – Latitude of the particle
- **lon** – Longitude of the particle
- **depth** – Depth of the particle in meters
- **timestamp** – a numpy.datetime64 instance representing the time of the particle

Returns A Particle object

__init__(*num=None*, *lat=None*, *lon=None*, *depth=None*, *timestamp=None*, *x=None*, *y=None*, *u=None*, *v=None*, *w=None*, *temp=None*, *sal=None*, *ssh=None*, *filepath=None*)
Initialize self. See help(type(self)) for accurate signature.

x
the x coordinate in meters

y
the y coordinate in meters

u
the u-velocity (eastward velocity) at the particle's location

v
the v-velocity (northward velocity) at the particle's location

w
the w-velocity (upward velocity) at the particle's location

temp
the temperature at the particle's location

sal
the salinity at the particle's location

ssh
the sea surface height at the particle's location

filepath
the expected file path given the particles timestamp

octapy.tracking.**deepcopy**(*particle*)
make a deep copy of a particle

Parameters **particle** – A particle instance

Returns A Particle object

octapy.tracking.**transform**(*src_crs*, *tgt_crs*, *lon*, *lat*)
transform the longitude and latitude of a point to x and y coordinates for a given coordinate reference system

Parameters

- **src_crs** – the source coordinate reference system
- **tgt_crs** – the target coordinate reference system
- **lon** – the longitude of the point

- **lat** – the latitude of the point

Returns the x and y coordinates of the point

octapy.tracking.**download_hycom_data**(*model*)
Download the input data for a given Model instance

Parameters **model** – a model instance

Returns None

octapy.tracking.**get_physical**(*particle*, *grid*, *model*)
wrapper for the interp3d function that interpolates for time if necessary

Parameters

- **particle** – a particle instance
- **grid** – a grid instance
- **model** – a model instance

Returns a particle instance

octapy.tracking.**interp3d**(*particle*, *grid*, *model*, *power=1.0*)
fills the interpolated environmental attributes for a particle instance

Parameters

- **particle** – a particle instance
- **grid** – a grid instance
- **model** – a model instance
- **power** – the power used in the inverse distance weighted interpolation

Returns a particle instance

octapy.tracking.**interp_for_time**(*particle*, *particle1*, *particle2*, *dims=2*)
linear interpolation of the environmental parameters for a particle in time

Parameters

- **particle** – a Particle instance at an invalid timestep for which environmental parameters are being interpolated in time
- **particle1** – a Particle instance at the last valid data timestep
- **particle2** – a Particle instance at the next valid data timestep
- **dims** – dimensions of the Model instance

Returns a Particle instance

octapy.tracking.**force_particle**(*particle*, *grid*, *model*)
force the particle to the next timestep

Parameters

- **particle** – a particle instance
- **grid** – a grid instance
- **model** – a model instance

Returns a particle instance

octapy.tracking.**add_row_to_arr**(*arr*, *particle*)
add a row containing the Particle attributes to an array

Parameters

- **arr** – a NumPy array
- **particle** – a Particle instance

Returns a NumPy array

octapy.tracking.**run_2d_model**(*model*, *grid*)
generate the trajectories for a 2D Model instance

Parameters

- **grid** – a grid instance
- **model** – a 2D model instance

Returns None

octapy.tracking.**run_3d_model**(*model*, *grid*)
generate the trajectories for a 3D Model instance

Parameters

- **grid** – a grid instance
- **model** – a 3D model instance

Returns None

CHAPTER

SEVEN

MODULE CONTENTS

PYTHON MODULE INDEX

O

INDEX

Symbols

A

B

D

F

G

I

L

M

N

O

P

R

S

T

U

V

W

X

Y