hydrointerp Documentation

Release 1.0.0

Mike Kittridge

Apr 29, 2021

MODULES

The hydrointerp package contains a class and associated interpolation functions specifically designed for 2D hydrologic data with time series. The core functionality is derived from the Scipy interpolation functions griddata and map_coordinates.

The GitHub repository is found here

CHAPTER

ONE

INTRODUCTION

There are many hydrological datasets that have a representation as two dimensional spatial data (e.. lon/lat or X/Y) over many time steps (e.g. gridded precipitation, wind, temperature, groundwater levels, etc.). These datasets may be derived from remote sensing equipment (e.g. NASA satellites) or may be derived from point observations (e.g. meteorological stations). For the purposes of hydrologic analysis, there are obvious needs to convert from point station data to gridded, gridded to points, gridded to different spatial resolution gridded, and so on. This package provides these conversions using different types of interpolators.

At the moment, these interpolators are purely 2D spatial interpolators and do not combine interpolations in time and space. Combining the two different dimension types (length and time) is not trivial nor is it obvious how that should be performed. As a consequence, each time step is interpolated independently.

CHAPTER

TWO

INSTALLATION

hydrointerp can be installed via pip or conda:

```
pip install hydrointerp
```

or:

```
conda install -c mullenkamp hydrointerp
```

The core dependencies are Pandas, Scipy, xarray, and pyproj.

Be careful not to cause dependency conflicts due to pyproj. hydrointerp uses pyproj >= 2.1 due to a major improvement in functionality. Unfortunately, some other geographic python packages have not updated to the new version yet (e.g. geopandas). Consequently, you may not be able to have both hydrointerp and these other packages installed simultaneously until they update their package dependencies.

CHAPTER

THREE

HOW TO USE HYDROINTERP

This section will describe how to use the hydrointerp package. Nearly all outputs are either as Pandas DataFrames or Xarray Datasets.

3.1 Necessary imports

For the examples, the numpy, pandas, xarray, and hydrointerp packages are needed.

```
import numpy as np
import pandas as pd
import xarray as xr
from hydrointerp import Interp, datasets
```

3.2 Loading in appropriate input data

The input must be either a grid as an Xarray Dataset or as points as a Pandas DataFrame. Both of these input data must have associated naming parameters.

```
### Input Parameters
In [1]: nc1 = 'nasa_gpm_2015-06-18'

In [2]: csv1 = 'ecan_data_2015-06-18'

In [3]: grid_time_name = 'time'

In [4]: grid_x_name = 'lon'

In [5]: grid_y_name = 'lat'

In [6]: grid_data_name = 'precipitationCal'

In [7]: grid_crs = 4326

In [8]: point_time_name = 'date'

In [9]: point_x_name = 'NZTMX'

In [10]: point_y_name = 'NZTMY'

In [11]: point_data_name = 'precip'
```

(continues on next page)

(continued from previous page)

```
In [12]: point_crs = 2193

### Read input data
In [13]: ds = xr.open_dataset(datasets.get_path(nc1))

In [14]: df1 = pd.read_csv(datasets.get_path(csv1), parse_dates=['date'], infer_
↪datetime_format=True)

In [15]: print(df1.head())
        date  precip      NZTMX      NZTMY
0 2015-06-18    96.5  1507022.0  5266024.0
1 2015-06-19   166.0  1507022.0  5266024.0
2 2015-06-18    63.0  1506391.0  5253154.0
3 2015-06-19    73.0  1506391.0  5253154.0
4 2015-06-18    88.5  1482760.0  5244669.0

### Assign nan toplaces wherethe quality index is below 0.4
In [16]: ds2 = ds[[grid_data_name]].where(ds.precipitationQualityIndex > 0.4)

In [17]: print(ds2)
<xarray.Dataset>
Dimensions:           (lat: 160, lon: 150, time: 2)
Coordinates:
  * time              (time) datetime64[ns] 2015-06-18 2015-06-19
  * lon               (lon) float32 165.1 165.1 165.2 ... 179.8 179.9 179.9
  * lat               (lat) float32 -48.95 -48.85 -48.75 ... -33.15 -33.05
Data variables:
    precipitationCal  (time, lon, lat) float32 0.006805 0.07318 0.0 ... 0.0 0.0

### Close the file (by removing the object)
In [18]: ds.close()

### Create example points
In [19]: points_df = df1.loc[[6, 15, 132], [point_x_name, point_y_name]].copy()

In [20]: points_df.rename(columns={point_x_name: 'x', point_y_name: 'y'},␣
↪inplace=True)
```

3.3 Initialising Interp

The package and general usage is via the main Interp class. It must be initialised with appropriate datasets and name parameters. Bare in mind, it is not required to have both input grids and points. One set is fine, and the appropriate interpolation methods will appear.

```
In [21]: interpc = Interp(ds2, grid_time_name, grid_x_name, grid_y_name, grid_data_
↪name, grid_crs, point_data=df1, point_time_name=point_time_name, point_x_name=point_
↪x_name, point_y_name=point_y_name, point_data_name=point_data_name, point_crs=point_
↪crs)
```

3.4 Nan filling

If your grid has nans (which the example does), fill those nans with the grid_interp_na method.

```
In [22]: nan1 = ds2[grid_data_name].isnull().sum()

In [23]: interpc.grid_interp_na()

In [24]: nan2 = interpc.grid_data['precip'].isnull().sum()

In [25]: assert (nan1 > 0) & (nan2 == 0)
```

3.5 Base Interpolators

All the 2D interpolators you'll need...

```
## Parameters
In [26]: to_crs = 2193

In [27]: grid_res = 10000

In [28]: bbox=None

In [29]: order=2

In [30]: extrapolation='constant'

In [31]: cval=np.nan

In [32]: digits = 2

In [33]: min_lat = -48

In [34]: max_lat = -41

In [35]: min_lon = 170

In [36]: max_lon = 178

In [37]: min_val=0

In [38]: method='linear'

## grid to grid
In [39]: interp1 = interpc.grid_to_grid(grid_res, to_crs, bbox, order, extrapolation,␣
↪min_val=min_val)
Preparing input and output
Running interpolations...
Packaging up the output

In [40]: print(interp1)
<xarray.Dataset>
Dimensions:  (time: 2, x: 140, y: 180)
Coordinates:
  * time     (time) datetime64[ns] 2015-06-18 2015-06-19
```

(continues on next page)

(continued from previous page)

```
  * x        (x) float64 8.568e+05 8.668e+05 8.768e+05 ... 2.237e+06 2.247e+06
  * y        (y) float64 4.548e+06 4.558e+06 4.568e+06 ... 6.328e+06 6.338e+06
Data variables:
    precip   (time, x, y) float32 nan nan nan nan nan ... 0.0 -0.0 0.0 nan nan

## points to grid
In [41]: interp2 = interpc.points_to_grid(grid_res, to_crs, bbox, method,␣
↪extrapolation, min_val=min_val)
Prepare input and output data
Run interpolations...
2015-06-18 00:00:00
2015-06-19 00:00:00
Packaging up the output

In [42]: print(interp2)
<xarray.Dataset>
Dimensions:  (time: 2, x: 35, y: 33)
Coordinates:
  * time     (time) datetime64[ns] 2015-06-18 2015-06-19
  * y        (y) float64 5.018e+06 5.028e+06 5.038e+06 ... 5.328e+06 5.338e+06
  * x        (x) float64 1.329e+06 1.339e+06 1.349e+06 ... 1.659e+06 1.669e+06
Data variables:
    precip   (time, y, x) float64 nan nan nan nan nan ... nan nan nan nan nan

## grid to points
In [43]: interp3 = interpc.grid_to_points(points_df, to_crs, order, min_val=min_val)
Preparing input and output
Running interpolations...
Packaging up the output

In [44]: print(interp3)
                                    precip
time       x         y
2015-06-18 1515946.0 5249806.0  617.679993
           1581327.0 5279063.0   86.559998
           1437006.0 5066885.0   78.220001
2015-06-19 1515946.0 5249806.0   59.439999
           1581327.0 5279063.0   45.299999
           1437006.0 5066885.0    2.220000

## points to points
In [45]: interp4 = interpc.points_to_points(points_df, to_crs, method, min_val=min_
↪val)
2015-06-18 00:00:00
2015-06-19 00:00:00

In [46]: print(interp4)
                                  precip
time       x         y
2015-06-18 1515946.0 5249806.0    45.0
           1581327.0 5279063.0    12.0
           1437006.0 5066885.0    36.5
2015-06-19 1515946.0 5249806.0    55.5
           1581327.0 5279063.0    37.5
           1437006.0 5066885.0    17.0
```

3.6 Adjust grid from points

There is also a method to adjust a grid based on the point_data (bias correction). And a method to run tests on it's accuracy.

```
In [47]: interp5 = interpc.adjust_grid_from_points(grid_res, to_crs)
Preparing input and output
Running interpolations...
Packaging up the output
Preparing input and output
Running interpolations...
Packaging up the output
Prepare input and output data
Run interpolations...
2015-06-18 00:00:00
2015-06-19 00:00:00
Packaging up the output

In [48]: print(interp5)
<xarray.Dataset>
Dimensions:  (time: 2, x: 35, y: 32)
Coordinates:
  * x        (x) float64 1.327e+06 1.337e+06 1.347e+06 ... 1.657e+06 1.667e+06
  * y        (y) float64 5.018e+06 5.028e+06 5.038e+06 ... 5.318e+06 5.328e+06
  * time     (time) datetime64[ns] 2015-06-18 2015-06-19
Data variables:
    precip   (time, x, y) float64 15.05 17.17 15.35 20.94 ... 11.09 11.26 7.764

In [49]: interp6 = interpc.validate_grid_from_points(0.08, grid_res, to_crs)
Preparing input and output
Running interpolations...
Packaging up the output
Preparing input and output
Running interpolations...
Packaging up the output
Prepare input and output data
Run interpolations...
2015-06-18 00:00:00
2015-06-19 00:00:00
Packaging up the output
2015-06-18 00:00:00
2015-06-19 00:00:00
Preparing input and output
Running interpolations...
Packaging up the output

In [50]: print(interp6)
                              precip  point_precip  grid_precip
time       x         y
2015-06-18 1418364.0 5135227.0    5.8         15.22         9.04
           1499654.0 5232199.0   33.0         46.40        34.14
           1510512.0 5187224.0   31.0         23.38        26.83
           1515640.0 5225591.0   49.5         40.46        24.75
           1569400.0 5166693.0    6.5         15.57        17.80
           1674522.0 5337964.0    1.5           NaN          NaN
2015-06-19 1418364.0 5135227.0   14.2          8.42         6.91
           1499654.0 5232199.0   49.5         70.27       112.43
```

(continues on next page)

(continued from previous page)

```
1510512.0 5187224.0    31.5         49.47         51.22
1515640.0 5225591.0    41.5         48.58         95.60
1569400.0 5166693.0    28.0         16.37         18.26
1674522.0 5337964.0     8.5           NaN           NaN
```

CHAPTER

FOUR

PACKAGE REFERENCES

4.1 Base class

class hydrointerp.**Interp**(*grid_data=None*, *grid_time_name=None*, *grid_x_name=None*, *grid_y_name=None*, *grid_data_name=None*, *grid_crs=None*, *point_data=None*, *point_time_name=None*, *point_x_name=None*, *point_y_name=None*, *point_data_name=None*, *point_crs=None*)

Base Interp class to prepare the input data for the interpolation functions.

Parameters

- **data** (*DataFrame or Dataset*) – A pandas DataFrame containing four columns as shown in the below parameters or an Xarray Dataset.
- **time_name** (*str*) – If grid is a DataFrame, then time_name is the time column name. If grid is a Dataset, then time_name is the time coordinate name.
- **x_name** (*str*) – If grid is a DataFrame, then x_name is the x column name. If grid is a Dataset, then x_name is the x coordinate name.
- **y_name** (*str*) – If grid is a DataFrame, then y_name is the y column name. If grid is a Dataset, then y_name is the y coordinate name.
- **data_name** (*str*) – If grid is a DataFrame, then data_name is the data column name. If grid is a Dataset, then data_name is the data variable name.
- **from_crs** (*int or str or None*) – The projection info for the input data if the result should be reprojected to the to_crs projection (either a proj4 str or epsg int).

returns

rtype Interp object

4.2 Nan interpolator

Interp.**_grid_interp_na**(*method='linear'*, *min_val=None*, *inplace=True*)

Function to fill nan's in the grid_data to make it complete. Necessary for grid_to_* functions.

Parameters

- **method** (*str*) – The scipy griddata interpolation method to be applied. Options are 'nearest', 'linear', and 'cubic'. See https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html for more details.
- **min_val** (*int, float, or None*) – The minimum value for the results. All results below min_val will be assigned min_val.

- **inplace** (*bool*) – Should the new grid overwrite the grid_data from initialisation?

Returns

Return type Xarray Dataset

4.3 Base interpolators

Interp.**_grid_to_grid**(*grid_res*, *to_crs=None*, *bbox=None*, *order=3*, *extrapolation='constant'*, *fill_val=nan*, *digits=2*, *min_val=None*)

Function to interpolate regularly or irregularly spaced values over many time stamps. Each time stamp of spatial values are interpolated independently (2D interpolation as opposed to 3D interpolation). Returns an xarray Dataset with the 3 dimensions. Uses the scipy interpolation function called map_coordinates.

Parameters

- **grid_res** (*int or float*) – The resulting grid resolution in the unit of the final projection (usually meters or decimal degrees).
- **to_crs** (*int or str or None*) – The projection for the output data similar to from_crs.
- **bbox** (*tuple of int or float*) – The bounding box for the output interpolation in the to_crs projection. None will return a similar grid extent as the input. The tuple should contain four ints or floats in the following order: (x_min, x_max, y_min, y_max)
- **order** (*int*) – The order of the spline interpolation, default is 3. The order has to be in the range 0-5. An order of 1 is linear interpolation.
- **extrapolation** (*str*) – The equivalent of 'mode' in the map_coordinates function. Options are: 'constant', 'nearest', 'reflect', 'mirror', and 'wrap'. Most reseaonable options for this function will be either 'constant' or 'nearest'. See https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html for more details.
- **fill_val** (*int or float*) – If 'constant' if passed to the extrapolation parameter, fill_val assigns the value outside of the boundary. Defaults to numpy.nan.
- **digits** (*int*) – The number of digits to round the output.
- **min_val** (*int, float, or None*) – The minimum value for the results. All results below min_val will be assigned min_val.

Returns

Return type xarray Dataset

Interp.**_points_to_grid**(*grid_res*, *to_crs=None*, *bbox=None*, *method='linear'*, *extrapolation='contstant'*, *fill_val=nan*, *digits=2*, *min_val=None*)

Function to take a dataframe of point value inputs (df) and interpolate to a grid. Uses the scipy griddata function for interpolation.

Parameters

- **grid_res** (*int or float*) – The resulting grid resolution in the unit of the final projection (usually meters or decimal degrees).
- **to_crs** (*int or str or None*) – The projection for the output data similar to from_crs.

- **bbox** (*tuple of int or float*) – The bounding box for the output interpolation in the to_crs projection. None will return a similar grid extent as the input. The tuple should contain four ints or floats in the following order: (x_min, x_max, y_min, y_max)
- **method** (*str*) – The scipy griddata interpolation method to be applied. Options are 'nearest', 'linear', and 'cubic'. See https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html for more details.
- **extrapolation** (*str*) – Either 'constant' or 'nearest'.
- **fill_val** (*int or float*) – If 'constant' if passed to the extrapolation parameter, fill_val assigns the value outside of the boundary. Defaults to numpy.nan.
- **digits** (*int*) – The number of digits to round the output.
- **min_val** (*int, float, or None*) – The minimum value for the results. All results below min_val will be assigned min_val.

returns

rtype xarray Dataset

Interp.**_grid_to_points**(*point_data*, *to_crs=None*, *bbox=None*, *order=3*, *digits=2*, *min_val=None*)

Function to take a dataframe of point value inputs (df) and interpolate to other points (point_data). Uses the scipy griddata function for interpolation.

Parameters

- **point_data** (*str or DataFrame*) – Path to geometry file of points to be interpolated (e.g. shapefile). Can be any file type that fiona/gdal can open. It can also be a DataFrame with 'x' and 'y' columns and the crs must be the same as to_crs.
- **to_crs** (*int or str or None*) – The projection for the output data similar to from_crs.
- **bbox** (*tuple of int or float*) – The bounding box for the output interpolation in the to_crs projection. None will return a similar grid extent as the input. The tuple should contain four ints or floats in the following order: (x_min, x_max, y_min, y_max)
- **order** (*int*) – The order of the spline interpolation, default is 3. The order has to be in the range 0-5. An order of 1 is linear interpolation.
- **digits** (*int*) – The number of digits to round the output.
- **min_val** (*int, float, or None*) – The minimum value for the results. All results below min_val will be assigned min_val.

Returns

Return type DataFrame

Interp.**_points_to_points**(*point_data*, *to_crs=None*, *method='linear'*, *digits=2*, *min_val=None*)

Function to interpolate regularly or irregularly spaced values over many time stamps. Each time stamp of spatial values are interpolated independently (2D interpolation as opposed to 3D interpolation). Returns an xarray Dataset with the 3 dimensions. Uses the scipy interpolation function called map_coordinates.

Parameters

- **point_data** (*str or DataFrame*) – Path to geometry file of points to be interpolated (e.g. shapefile). Can be any file type that fiona/gdal can open. It can also be a DataFrame with 'x' and 'y' columns and the crs must be the same as to_crs.
- **to_crs** (*int or str or None*) – The projection for the output data similar to from_crs.

- **method** (*str*) – The scipy griddata interpolation method to be applied. Options are 'nearest', 'linear', and 'cubic'.
- **digits** (*int*) – The number of digits to round the output.
- **min_val** (*int, float, or None*) – The minimum value for the results. All results below min_val will be assigned min_val.

Returns

Return type DataFrame

4.4 Adjust grid from points

Interp.**_adjust_grid_from_points**(*grid_res=None*, *to_crs=None*, *order=2*, *method='linear'*, *digits=2*, *min_val=0*)

Method to adjust a grid by forcing it through point data.

Parameters

- **grid_res** (*int, float, or None*) – The resulting grid resolution in the unit of the final projection (usually meters or decimal degrees).
- **to_crs** (*int or str or None*) – The projection for the output data similar to from_crs.
- **order** (*int*) – The order of the spline interpolation, default is 3. The order has to be in the range 0-5. An order of 1 is linear interpolation.
- **method** (*str*) – The scipy griddata interpolation method to be applied. Options are 'nearest', 'linear', and 'cubic'. See https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html for more details.
- **digits** (*int*) – The number of digits to round the output.
- **min_val** (*int, float, or None*) – The minimum value for the results. All results below min_val will be assigned min_val.

Returns

Return type Dataset

Interp.**_validate_grid_from_points**(*test_perc=0.1*, *grid_res=None*, *to_crs=None*, *order=2*, *method='linear'*, *digits=2*, *min_val=0*)

Method to validate a grid created by the adjust_grid_from_points method.

Parameters

- **grid_res** (*int, float, or None*) – The resulting grid resolution in the unit of the final projection (usually meters or decimal degrees).
- **to_crs** (*int or str or None*) – The projection for the output data similar to from_crs.
- **order** (*int*) – The order of the spline interpolation, default is 3. The order has to be in the range 0-5. An order of 1 is linear interpolation.
- **method** (*str*) – The scipy griddata interpolation method to be applied. Options are 'nearest', 'linear', and 'cubic'. See https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html for more details.
- **digits** (*int*) – The number of digits to round the output.

- **min_val** (*int, float, or None*) – The minimum value for the results. All results below min_val will be assigned min_val.

Returns

Return type DataFrame

4.5 API Pages

CHAPTER

FIVE

LICENSE AND TERMS OF USAGE

This package is licensed under the terms of the Apache License Version 2.0 and can be found on the GitHub project page.

INDEX

Symbols

I