pyEDM Version 0.1.2 May 27, 2019

pyEDM is a Python package interface to the cppEDM C++ library of empirical dynamic modeling (EDM) algorithms. It is loaded within Python by `import EDM` and returns Pandas DataFrame objects.

Table of Contents

University of California at San Deigo
Scripps Institute of Oceangraphy
Sugihara Lab

Joseph Park, Cameron Smith

Introduction

pyEDM is a Python interface to the C++ library cppEDM. Input and output objects are based on Pandas DataFrame objects. Core algorithms are listed in table 1.

Algorithm	API Interface	Reference
Simplex projection	`Simplex()`	Sugihara and May (1990)
Sequential Locally Weighted Global Linear Maps (S-map)	`SMap()`	Sugihara (1994)
Predictions from multivariate embeddings	`Simplex(), SMap()`	Dixon et. al. (1999)
Convergent cross mapping	`CCM()`	Sugihara et. al. (2012)
Multiview embedding	`Multiview()`	Ye and Sugihara (2016)

Convenience functions to prepare and evaluate data are listed in table 2.

Function	Purpose	Parameter Range
`Embed()`	Timeseries delay dimensional embedding	User defined
`MakeBlock()`	Timeseries delay dimensional embedding	User defined
`EmbedDimension()`	Evaluate prediction skill vs. embedding dimension	E = [1, 10]
`PredictInterval()`	Evaluate prediction skill vs. forecast interval	Tp = [1, 10]
`PredictNonlinear()`	Evaluate prediction skill vs. SMap nonlinear localisation	θ = 0.01, 0.1, 0.3, 0.5, 0.75, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, 9
`ComputeError()`	Pearson ρ, RMSE, MAE	
`Examples()`	Example function calls and plots	

Installation

OSX and Windows (Python 3):

pyEDM is available from the Python Package Index (PyPI):
https://pypi.org/project/EDM_SugiharaLab

```
pip install EDM_SugiharaLab
```
or
```
python -m pip install EDM_SugiharaLab
```

Linux:

PyPI does not support C++ extension libraries on Linux.

1) Build the cppEDM libEDM.a from github.com/SugiharaLab/cppEDM by running "`make`" in the cppEDM/src/ directory with the "`-fPIC`" compiler flag:
```
make CFLAGS="-std=c++11 -DCCM_THREADED -DMULTIVIEW_VALUES_OVERLOAD -O3 -fPIC"
```

2) Download pyEDM from github.com/SugiharaLab/pyEDM. Copy the cppEDM/lib/libEDM.a into pyEDM/lib.

3) Install the EDM Python module from pyEDM/:
```
"python -m pip install . --user"
```

Usage

```
>>> import EDM
>>> EDM.Examples()
```

or

```
>>> from EDM import *
>>> Examples()
```

Parameters

API parameter names and purpose are listed in table 3.

Parameter	Type	Default	Purpose
`pathIn`	`string`	`"./"`	Input data file path
`dataFile`	`string`	`""`	Data file name
`dataFrame`	`Pandas DataFrame`	`None`	Input DataFrame
`pathOut`	`string`	`"./"`	Output file path
`predictFile`	`string`	`""`	Prediction output file
`lib`	`string`	`""`	library start : stop row indices
`pred`	`string`	`""`	prediction start : stop row indices
`E`	`int`	`0`	Data dimension
`Tp`	`int`	`0`	Prediction interval
`knn`	`int`	`0`	Number nearest neighbors
`tau`	`int`	`1`	Embedding delay
`theta`	`float`	`0`	SMap localisation
`columns`	`string`	`""`	Column names or indices for prediction
`target`	`string`	`""`	Target library column name or index
`embedded`	`bool`	`false`	Is data an embedding?
`verbose`	`bool`	`false`	Echo messages
`smapFile`	`string`	`""`	SMap coefficient output file
`libSizes_str`	`string`	`""`	CCM library sizes
`sample`	`int`	`0`	CCM number of random samples
`random`	`bool`	`true`	CCM use random samples?
`seed`	`unsigned`	`0`	RNG seed, 0 = random seed

Application Programming Interface (API)

Embed

Create a data block of Takens (1981) time-delay embedding from each of the columns in the csv file or dataFrame. The `columns` parameter can be a list of column names, or a list of column indices. If `columns` is a list of indices, then column names are created as V1, V2...

Note: The returned DataFrame will have `tau*(E-1)` fewer rows than the input data from the removal of partial vectors as a result of the embedding.

Note: The returned DataFrame will not have the time column.

```
//------------------------------------------------------------------
//
//------------------------------------------------------------------
DataFrame Embed ( path      = "",
                  dataFile  = "",
                  dataFrame = None,
                  E         = 0,
                  tau       = 0,
                  columns   = "",
                  verbose   = false )


//------------------------------------------------------------------
//
//------------------------------------------------------------------
DataFrame MakeBlock ( dataFrame,
                      E,
                      tau,
                      columnNames,
                      verbose )
```

Simplex

Simplex projection of the input data file or DataFrame. The returned `DataFrame` has 3 columns "Time", "Observations", "Predictions". nan values are inserted where there is no observation or prediction. See the Parameters table for parameter definitions.

`lib` and `pred` specify [start stop] row indices of the input data for the library and predictions.

If `embedded` is `false` the data columns are embedded to dimension `E` with delay `tau`. If `embedded` is `true` the data columns are assumed to be a multivariable data block.

If `knn` is not specified, it is set equal to `E+1`.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame Simplex( pathIn       = "./data/",
                   dataFile     = "",
                   dataFrame    = None,
                   pathOut      = "./",
                   predictFile  = "",
                   lib          = "",
                   pred         = "",
                   E            = 0,
                   Tp           = 1,
                   knn          = 0,
                   tau          = 1,
                   columns      = "",
                   target       = "",
                   embedded     = false,
                   verbose      = true )
```

SMap

SMap projection of the input data file or DataFrame. See the Parameters table for parameter definitions.

`SMap()` returns a `dict` with two DataFrames:

```
dict { predictions  : DataFrame,
       coefficients : DataFrame
}
```

The `predictions` `DataFrame` has 3 columns "Time", "Observations", "Predictions". nan values are inserted where there is no observation or prediction. If `predictFile` is provided the `predictions` will be written to it in csv format.

The `coefficients` `DataFrame` will have `E+2` columns. The first column is the "Time" vector, the remaining `E+1` columns are the SMap SVD fit coefficients.

`lib` and `pred` specify [start, stop] row indices of the input data for the library and predictions.

If `embedded` is `false` the data columns are embedded to dimension `E` with delay `tau`. If `embedded` is `true` the data columns are assumed to be a multivariable data block. If `smapFile` is provided the `coefficients` will be written to it in csv format.

If `knn` is not specified, it is set equal to the library size. If `knn` is specified, it must be greater than `E`.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
dict SMap( pathIn          = "./data/",
           dataFile        = "",
           dataFrame       = None,
           pathOut         = "./",
           predictFile     = "",
           lib             = "",
           pred            = "",
           E               = 0,
           Tp              = 1,
           knn             = 0,
           tau             = 1,
           theta           = 0,
           columns         = "",
           target          = "",
           smapFile        = "",
           jacobians       = "",   // Not implemented
           embedded        = false,
           verbose         = true )
```

CCM

Convergent cross mapping via Simplex of the first vector specified in `columns` against `target`. The data cannot be multivariable, the first vector in `columns` is time-delay embedded to dimension `E`. See the Parameters table for parameter definitions.

The returned `DataFrame` has 3 columns. The first column is "LibSize", the second and third columns are Pearson correlation coefficients for "column : target" and "target : column" cross mapping.

`libSizes` specifies a string with "start stop increment" row values, i.e. "10 80 10" will evaluate library sizes from 10 to 80 in increments of 10.

If `random` is `true`, `sample` observations are radomly selected from the subset of each library size. If `seed=0`, then a random seed is generated for the random number generator. Otherwise, `seed` is used to initialise the random number generator.

If `random` is false, `sample` is ignored and contiguous library rows up to the current library size are used.

Note: Cross mappings are performed between `column : target`, and `target : column`. The default is to do this in separate threads. Threading can be disabled in the makefile by removing `-DCCM_THREADED`.

Note: The entire library size is used in the Simplex prediction at each library subset size.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame CCM( pathIn       = "./data/",
               dataFile     = "",
               dataFrame    = None,
               pathOut      = "./",
               predictFile  = "",
               E            = 0,
               Tp           = 0,
               knn          = 0,
               tau          = 1,
               columns      = "",
               target       = "",
               libSizes     = "",
               sample       = 0,
               random       = true,
               seed         = 0,     // seed=0: use RNG
               verbose      = true );
```

Multiview

Multiview embedding and forecasting of the input data file or DataFrame. See the Parameters table for parameter definitions.

`Multiview()` returns a `dict`:

```
dict { Combo_rho   : DataFrame,
       Predictions : DataFrame
}
```

The `Predictions` `DataFrame` has 3 columns "Time", "Observations", "Predictions". nan values are inserted where there is no observation or prediction. If `predictFile` is provided the `Predictions` will be written to it in csv format.

The `Combo_rho` `DataFrame` will have `E+3` columns. The first `E` columns are the the column indices in the input data `DataFrame` that are embedded and applied to Simplex prediction. The last three columns are "rho", "MAE", "RMSE" corresponding to the prediction Pearson correlation, maximum absolute error and root mean square error.

`lib` and `pred` specify [start, stop] row indices of the input data for the library and predictions.

If `multiview` is not specified it is set to sqrt(C) where C is the number of E-dimensional combinations out of all available data vectors.

If `knn` is not specified, it is set equal to `E+1`.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
dict Multiview( pathIn      = "./",
                dataFile    = "",
                dataFrame   = None,
                pathOut     = "./",
                predictFile = "",
                lib         = "",
                pred        = "",
                E           = 0,
                Tp          = 1,
                knn         = 0,
                tau         = 1,
                columns     = "",
                target      = "",
                multiview   = 0,
                verbose     = false,
                nThreads    = 4 )
```

EmbedDimension

Evaluate Simplex prediction skill for embedding dimensions from 1 to 10. The returned `DataFrame` has columns "E" and "rho". See the Parameters table for parameter definitions.

Note: `nThreads` defines the number of worker threads for the 10 embeddings. The maximum number of threads is 10.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame EmbedDimension( pathIn      = "./data/",
                          dataFile    = "",
                          dataFrame   = None,
                          pathOut     = "./",
                          predictFile = "",
                          lib         = "",
                          pred        = "",
                          Tp          = 1,
                          tau         = 1,
                          columns     = "",
                          target      = "",
                          embedded    = false,
                          verbose     = true,
                          nThreads    = 4 )
```

PredictInterval

Evaluate Simplex prediction skill for forecast intervals from 1 to 10. The returned `DataFrame` has columns "Tp" and "rho". See the Parameters table for parameter definitions.

Note: `nThreads` defines the number of worker threads for the 10 prediction interval forecasts. The maximum number of threads is 10.

```
//----------------------------------------------------------------
// Overload 1: Explicit data file path/name
//----------------------------------------------------------------
DataFrame PredictInterval( pathIn      = "./data/",
                           dataFile    = "",
                           dataFrame   = None,
                           pathOut     = "./",
                           predictFile = "",
                           lib         = "",
                           pred        = "",
                           E           = 0,
                           tau         = 1,
                           columns     = "",
                           target      = "",
                           embedded    = false,
                           verbose     = true,
                           nThreads    = 4 );
```

PredictNonlinear

Evaluate SMap prediction skill for localisation parameter θ from 0.01 to 9. The returned `DataFrame` has columns "theta" and "rho". See the Parameters table for parameter definitions.

Note: `nThreads` defines the number of worker threads for the 15 θ value forecasts.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame PredictNonlinear( pathIn      = "./data/",
                            dataFile    = "",
                            dataFrame   = None,
                            pathOut     = "./",
                            predictFile = "",
                            lib         = "",
                            pred        = "",
                            E           = 0,
                            Tp          = 1,
                            tau         = 1,
                            columns     = "",
                            target      = "",
                            embedded    = false,
                            verbose     = true,
                            nThreads    = 4 );
```

ComputeError

Compute Pearson correlation coefficient, maximum absolute error (MAE) and root mean square error (RMSE) between two vectors.

`ComputeError()` returns a dict:

```
dict { rho  :  double,
       RMSE :  double,
       MAE  :  double
}

//------------------------------------------------------------------
//------------------------------------------------------------------
dict ComputeError( obsIn, predIn )
```

Application Notes

All data input files are assumed to be in csv format. The files are assumed to have a single line header with column names. If column names are not detected in the header line, then column names are created as V1, V2... **It is required that the first column be a vector of times or time indices.**

`SMap()` should be called with `DataFrame` that have columns explicity corresponding to dimensions E. This means that if a multivariate data set is used, it should Not be called with an embedding from `Embed()` since `Embed()` will add lagged coordinates for each variable. These extra columns will then not correspond to the intended dimensions in the matrix inversion and prediction reconstruction. In this case, use the `embedded` parameter set to `true` so that the columns selected correspond to the proper dimension.

Examples

```
df = EmbedDimension( "./data/", "TentMap_rEDM.csv", None, "./", "",
                     "1 100", "201 500", 1, 1,
                     "TentMap", "", False, False, 4 )

df = PredictInterval( "./data/", "TentMap_rEDM.csv", None, "./", "",
                      "1 100", "201 500", 2, 1,
                      "TentMap", "", False, False, 4 )

df = PredictNonlinear( "./data/", "TentMapNoise_rEDM.csv", None, "./", "",
                       "1 100", "201 500", 2, 1, 1,
                       "TentMap", "", False, False, 4 )

df = Simplex( "./data/", "block_3sp.csv", None, "./", "",
              "1 99", "100 198", 3, 1, 0, 1,
              "x_t y_t z_t", "x_t", True, True, True )

df = Simplex( "./data/", "block_3sp.csv", None, "./", "",
              "1 99", "100 195", 3, 1, 0, 1,
              "x_t", "x_t", False, True, True )

M = Multiview( "./data/", "block_3sp.csv", None, "./", "",
               "1 100", "101 198", 3, 1, 0, 1,
               "x_t y_t z_t", "x_t", 0, False, 4, True )

S = SMap( "./data/", "circle.csv", None, "./", "",
          "1 100", "101 198", 2, 1, 0, 1, 4,
          "x y", "x", "", "", True, True, True )

df = CCM( "./data/", "sardine_anchovy_sst.csv", None, "./", "",
          3, 0, 0, 1, "anchovy", "np_sst",
          "10 80 10", 100, True, 0, True, True )
```

References

Dixon, P. A., M. Milicich, and G. Sugihara, 1999. Episodic fluctuations in larval supply. Science 283:1528–1530.

Sugihara G. and May R. 1990. Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. Nature, 344:734–741.

Sugihara G. 1994. Nonlinear forecasting for the classification of natural time series. Philosophical Transactions: Physical Sciences and Engineering, 348 (1688) : 477–495.

Sugihara G., May R., Ye H., Hsieh C., Deyle E., Fogarty M., Munch S., 2012. Detecting Causality in Complex Ecosystems. Science 338:496-500.

Takens, F. Detecting strange attractors in turbulence. Lect. Notes Math. 898, 366–381 (1981).

Ye H., and G. Sugihara, 2016. Information leverage in interconnected ecosystems: Overcoming the curse of dimensionality. Science 353:922–925.