pyEDM Version 1.8.1.1   May 2, 2021

pyEDM is a Python package interface to the cppEDM C++ library of empirical dynamic modeling (EDM) algorithms. It returns Pandas DataFrame objects, or Python dictionaries of Pandas DataFrames.

Table of Contents

University of California at San Diego
Scripps Institution 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

There are two methods to install pyEDM:
1) Python Package Index and pip, which is only supported for certain OSX and Windows platforms
2) Download, build and install package.

Python Package Index (PyPI)

Certain Mac OSX and Windows platforms are supported with prebuilt binary distributions and can be installed using the Python pip module. The module is located at pypi.org/project/pyEDM/.

Installation can be executed as:

```
python -m pip install pyEDM --trusted-host pypi.org --trusted-host
files.pythonhosted.org pyEDM
```

Manual Compilation

Unfortunately, we do not have the resources to provide pre-built binary distributions for all computer platforms. In this case the user is required to first build the cppEDM library on their machine, and then install the Python package using pip. On OSX and Linux this requires gcc, on Windows, mingw and Microsoft Visual Studio Compiler (MSVC) which can be obtained from Build Tools for Visual Studio 2019. Only the Windows SDK is needed. Note that the LAPACK library is required to build cppEDM.

OSX and Linux

1) Download pyEDM: git clone https://github.com/SugiharaLab/pyEDM

2) Build cppEDM library:

```
cd pyEDM/cppEDM/src
make
```

3) Build and install package:

```
cd ../..
python -m pip install . --user --trusted-host pypi.org
```

Windows

We do not have resources to maintain windows build support. These suggestions may be useful.

Requires mingw installation. Requires gfortran libraries.
1) Download pyEDM: git clone https://github.com/SugiharaLab/pyEDM
2) Build cppEDM library: `cd pyEDM\cppEDM\src; make`
3) Adjust paths to find gfortran and openblas libraries (`pyEDM/pyEDM/etc/windows/libopenblas.a`).
   You may need to rename libEDM.a to EDM.lib, and openblas.a to openblas.lib.
4) Build and install package in pyEDM\: `python -m pip install . --user --trusted-host pypi.org`

Usage

```
>>> import pyEDM
>>> pyEDM.Examples()
```

See the Examples section below.

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.**

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
`D`	`int`	`0`	Multiview state-space dimension
`E`	`int`	`0`	Embedding dimension
`Tp`	`int`	`0 or 1`	Prediction interval
`knn`	`int`	`0`	Number nearest neighbors
`tau`	`int`	`-1`	Embedding delay
`theta`	`float`	`0`	SMap localisation
`exclusionRadius`	`int`	`0`	Prediction vector exclusion radius
`columns`	`string`	`""`	Column names or indices for prediction
`target`	`string`	`""`	Target library column name or index
`embedded`	`bool`	`false`	Is data an embedding?
`const_pred`	`bool`	`false`	Include non projected forecast data
`verbose`	`bool`	`false`	Echo messages
`smapFile`	`string`	`""`	SMap coefficient output file
`solver`	`sklearn.linear_model`	`None`	SMap solver
`multiview`	`int`	`0`	Number of ensembles, 0 = sqrt(N)
`trainLib`	`bool`	`true`	Multiview use lib as training library
`excludeTarget`	`bool`	`true`	Multiview exclude target from combos
`libSizes_str`	`string`	`""`	CCM library sizes
`sample`	`int`	`0`	CCM number of random samples
`random`	`bool`	`true`	CCM use random samples?
`replacement`	`bool`	`false`	CCM sample with replacement?
`includeData`	`bool`	`false`	CCM include all projections in return
`seed`	`unsigned`	`0`	CCM RNG seed, 0 = random seed
`method`	`string`	`"ebisusaki"`	SurrogateData method
`alpha`	`float`	`range / 5`	SurrogateData seasonal noise std dev
`smooth`	`float`	`0.8`	SurrogateData seasonal spline smooth

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       = -1,
                  columns   = "",
                  verbose   = False )
```

```
//------------------------------------------------------------------
//
//------------------------------------------------------------------
DataFrame MakeBlock ( dataFrame,
                      E             = 0,
                      tau           = -1,
                      columnNames   = "",
                      deletePartial = False )
```

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.

Parameters

`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, and `embedded` is `false`, it is set equal to `E+1`. If `embedded` is `true, knn` is set equal to the number of `columns + 1`.

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

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. The first column "C0" is the bias term, following coefficients are ∂ columns[i] / ∂ target.

Parameters

`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 a multivariate data set is used (number of `columns > 1`) it must use `embedded = true` with `E` equal to the number of `columns`. This prevents the function from internally time-delay embedding the multiple columns to dimension E. If the internal time-delay embedding is performed, then state-space columns will not correspond to the intended dimensions in the matrix inversion, coefficient assignment, and prediction. In the multivariate case, the user should first prepare the embedding (using `Embed()` for time-delay embedding if desired), then pass this embedding to `SMap` with appropriately specified `columns`, E, and `embedded = true`.

If `knn` is not specified, it is set equal to the library size. If `knn` is specified, it must be greater than `E+1`.

The default solver for the SMap coefficient matrix is the LAPACK SVD function `dgelss()`. This can be replaced with a user-instantianted class object from the python `sklearn.linear_model: Linear Models`. Supported solvers include: `LinearRegression, Ridge, Lasso, ElasticNet, RidgeCV, LassoCV, ElasticNetCV`. See the `pyEDM/tests/smapSolverTest.py` script for examples.

```
//---------------------------------------------------------------------
//
//---------------------------------------------------------------------
dict SMap( pathIn          = "./",
           dataFile        = "",
           dataFrame       = None,
           pathOut         = "./",
           predictFile     = "",
           lib             = "",
           pred            = "",
           E               = 0,
           Tp              = 1,
           knn             = 0,
           tau             = -1,
           theta           = 0,
           exclusionRadius = 0,
           columns         = "",
           target          = "",
           smapFile        = "",
           derivatives     = "",   // Not implemented
           solver          = None,
           embedded        = False,
           verbose         = False,
           const_pred      = False,
           showPlot        = False )
```

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.

If `includeData` is False, 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. If `includeData` is True, a `dict` is returned with the `LibMeans` `DataFrame` , and a `DataFrames` of prediction statistics for all predictions in the ensembles.

Parameters

`The libSizes` parameter is a string of whitespace or comma separated library sizes. If the string has 3 values, and, if the third value is less than the second value, then the three values are interpreted as a sequence generator specifying "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 prediction vector is used in the Simplex prediction at each library subset size.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame or dict CCM( pathIn          = "./",
                       dataFile        = "",
                       dataFrame       = None,
                       pathOut         = "./",
                       predictFile     = "",
                       E               = 0,
                       Tp              = 0,
                       knn             = 0,
                       tau             = -1,
                       exclusionRadius = 0,
                       columns         = "",
                       target          = "",
                       libSizes        = "",
                       sample          = 0,
                       random          = True,
                       replacement     = False,
                       seed            = 0,     // seed=0: use RNG
                       includeData     = False,
                       verbose         = False,
                       showPlot        = False );
```

Multiview

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

`Multiview()` returns a `dict`:

```
dict { View        : 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 `View` `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.

Parameters

D represents the number of variables to combine for each assessment, if not specified, it is the number of columns.

E is the embedding dimension of each variable. If E = 1, no time delay embedding is done.

`multiview` is the number of top-ranked D-dimensional predictions to "average" for the final prediction. Corresponds to parameter k in Ye & Sugihara with default k = sqrt(C) where C is the number of combinations C(n,D) available from the embedding columns taken D at-a-time.

`trainLib` specifies whether projections used to rank the column combinations are done in-sample (pred = lib, the default), or, using the lib and pred specified as input options (`trainLib false).`

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

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

`numThreads` defines the number of worker threads.

```
//---------------------------------------------------------------
//
//---------------------------------------------------------------
dict Multiview( pathIn          = "./",
                dataFile        = "",
                dataFrame       = None,
                pathOut         = "./",
                predictFile     = "",
                lib             = "",
                pred            = "",
                D               = 0,
                E               = 1,
                Tp              = 1,
                knn             = 0,
                tau             = -1,
                columns         = "",
                target          = "",
                multiview       = 0,
                exclusionRadius = 0,
                trainLib        = True,
                excludeTarget   = False,
                verbose         = False,
                numThreads      = 4,
                showPlot        = False )
```

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

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame EmbedDimension( pathIn      = "./",
                          dataFile    = "",
                          dataFrame   = None,
                          pathOut     = "./",
                          predictFile = "",
                          lib         = "",
                          pred        = "",
                          maxE        = 10,
                          Tp          = 1,
                          tau         = -1,
                          columns     = "",
                          target      = "",
                          embedded    = False,
                          verbose     = False,
                          numThreads  = 4,
                          showPlot    = False )
```

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: `numThreads` 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      = "./",
                           dataFile    = "",
                           dataFrame   = None,
                           pathOut     = "./",
                           predictFile = "",
                           lib         = "",
                           pred        = "",
                           maxTp       = 10,
                           E           = 0,
                           tau         = -1,
                           columns     = "",
                           target      = "",
                           embedded    = False,
                           verbose     = False,
                           numThreads  = 4,
                           showPlot    = False );
```

PredictNonlinear

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

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

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

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame PredictNonlinear( pathIn      = "./",
                            dataFile    = "",
                            dataFrame   = None,
                            pathOut     = "./",
                            predictFile = "",
                            lib         = "",
                            pred        = "",
                            theta       = "",
                            E           = 0,
                            knn         = 0,
                            Tp          = 1,
                            tau         = -1,
                            columns     = "",
                            target      = "",
                            embedded    = False,
                            verbose     = False,
                            numThreads  = 4,
                            showPlot    = False );
```

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 )
```

SurrogateDate

Generate surrogate data using one of three methods.

1) random_shuffle :
    Sample the data with a uniform distribution.

2) ebisuzaki :
    Journal of Climate. A Method to Estimate the Statistical Significance of a Correlation When the Data Are Serially Correlated.
    https://doi.org/10.1175/1520-0442(1997)010<2147:AMTETS>2.0.CO;2

    Presumes data are serially correlated with low pass coherence. It is: "resampling in the frequency domain. This procedure will not preserve the distribution of values but rather the power spectrum (periodogram). The advantage of preserving the power spectrum is that resampled series retains the same autocorrelation as the original series."

3) seasonal :
    Presume a smoothing spline represents the seasonal trend. The smooth parameter can range from 0 to 1. See scipy.interpolate.UnivariateSpline parameter `s`.

    Each surrogate is a summation of the trend, resampled residuals, and possibly additive Gaussian noise. Default noise has a standard deviation (alpha) that is the data range / 5.

Note: It is presumed the first column of the dataFrame is a time vector. It is set as the first column of the returned DataFrame.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame Surrogatedate ( dataFrame     = None,
                          column        = None,
                          method        = 'ebisuzaki',
                          numSurrogates = 10,
                          alpha         = None,
                          smooth        = 0.8,
                          outputFile    = None )
```

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

```
from pyEDM import *

df = EmbedDimension( dataFrame = sampleData["TentMap"],
                     lib = "1 100", pred = "201 500",
                     columns = "TentMap", showPlot = True )

df = PredictInterval( dataFrame = sampleData["TentMap"],
                      lib = "1 100", pred = "201 500",  E = 2,
                      columns = "TentMap", showPlot = True )

df = PredictNonlinear( dataFrame = sampleData["TentMapNoise"],
                       lib = "1 100", pred = "201 500", E = 2,
                       columns = "TentMap", showPlot = True )

df = Simplex( dataFrame = sampleData["block_3sp"],
              lib = "1 99", pred = "100 198", E = 3,
              columns = "x_t y_t z_t", target = "x_t",
              embedded = True, showPlot = True )

df = Simplex( dataFrame = sampleData["block_3sp"],
              lib = "1 99", pred = "100 195", E = 3,
              columns = "x_t", target = "x_t", showPlot = True )

M = Multiview( dataFrame = sampleData["block_3sp"],
               lib = "1 100", pred = "101 198", E = 3,
               columns = "x_t y_t z_t", target = "x_t", showPlot = True )

S = SMap( dataFrame = sampleData["circle"],
          lib = "1 100", pred = "101 198", E = 2, theta = 4,
          columns = "x y", target = "x", embedded = True, showPlot = True )

df = CCM( dataFrame = sampleData["sardine_anchovy_sst"],
          E = 3, Tp = 0, columns = "anchovy", target = "np_sst",
          libSizes = "5 75 5", sample = 100, showPlot = 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.