pyEDM Version 2.2.0 January 27, 2025

pyEDM is a Python package of empirical dynamic modeling (EDM) algorithms. Input data are Pandas `DataFrame`, return objects are Pandas `DataFrame`, Python dictionaries of Pandas `DataFrame`, or the class object of the algorithm. pyEDM is hosted on the Python Package Index (PyPI) at pypi/pyEDM. A Jupyter notebook GUI is available at jpyEDM.

Table of Contents

Joseph Park

University of California San Diego
Scripps Institution of Oceanography, Sugihara Lab

Okinawa Institute of Science and Technology
Biological Nonlinear Dynamics Data Science Unit

Introduction

pyEDM is a Python package implementing empirical dynamic modeling (EDM) algorithms. Input and output objects are based on Pandas `DataFrame`. Core algorithms are listed in Table 1. Version 2. is a pure Python implementation based on numpy replacing the C++ kernel used in Version 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. The functions `EMbedDimension(), PredictInterval(), PredictNonlinear()` use multiprocessing for parallelism.

Function	Purpose	Default Parameters
`Embed()`	Timeseries delay dimensional embedding	
`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 localization	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

Python Package Index (PyPI)

pyEDM is available at pypi.org/project/pyEDM/.

Installation: `python -m pip install pyEDM`

Local Install

pyEDM can be installed from the github archive.

1) Download pyEDM: `git clone https://github.com/SugiharaLab/pyEDM`
2) Build and install package in pyEDM directory: `python -m pip install .`

Example Usage

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

See the Examples section below.

Data Requirements

Input data files are required to be a Pandas `DataFrame`.

DataFrames are required to have column names.

It is expected the first column is a vector of times or time indices. This can be disabled with the `noTime = True` parameter.

The data are expected to sequential time series observations. NaN data will be removed if `ignoreNan = True`, the default.

Parameters

API parameter names and purpose are listed in Table 3.

Parameter	Type	Default	Purpose
dataFrame	Pandas DataFrame	None	Input DataFrame
columns	[] or string	""	Column names or indices for library
target	[] or string	""	Target library column name or index
lib	[] or string	""	library start : stop row indices
pred	[] or 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 localization
exclusionRadius	int	0	Prediction vector exclusion radius
embedded	bool	False	Is data an embedding?
ignoreNan	bool	True	Remove nan from lib & target
validLib	[bool]	[]	Conditional Embedding (CE)
noTime	bool	False	Do not require first column to be time
generateSteps	int	0	Generative mode (Simplex, SMap)
generateConcat	bool	False	Merge original and generated data
solver	sklearn.linear_model	None	SMap solver
libSizes	[int] or string	[]	CCM library sizes
sample	int	0	CCM number of random samples
includeData	bool	False	CCM include all statistics in return
seed	unsigned	0	CCM RNG seed, 0 = random seed
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
method	string	"ebisusaki"	SurrogateData method
alpha	float	range / 5	SurrogateData seasonal noise std dev
smooth	float	0.8	SurrogateData seasonal spline smooth
verbose	bool	False	Echo messages

Application Programming Interface (API)

Embed

Create a time-delay embedding from each of the `columns` in the `dataFrame` or csv `dataFile`.

```
Embed ( dataFrame   = None,
        E           = 0,
        tau         = -1,
        columns     = "",
        includeTime = False,
        pathIn      = "./",
        dataFile    = "" )
```

Return: DataFrame with time shifted values dictated by `E` and `tau`.

Note: The returned DataFrame will not have the time column unless `includeTime = True`.

Simplex

Simplex projection of the input DataFrame `columns` to the `target`.

```
Simplex ( dataFrame       = None,
          columns         = "",
          target          = "",
          lib             = "",
          pred            = "",
          E               = 0,
          Tp              = 1,
          knn             = 0,
          tau             = -1,
          exclusionRadius = 0,
          embedded        = False,
          validLib        = [],
          noTime          = False,
          generateSteps   = 0,
          generateConcat  = False,
          verbose         = False,
          showPlot        = False,
          ignoreNan       = True,
          returnObject    = False )
```

Return:

If `returnObject = False`: returns `DataFrame` with columns : "Time", "Observations", "Predictions", “Pred_Variance”.

If `returnObject = True`: the Simplex class object is returned. The attribute `Projection` contains the simplex projection DataFrame.

Parameters

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

If `embedded` is `False` all data columns are embedded to dimension `E` with delay `tau`. If `embedded` is `True` the data columns are assumed to be an embedding.

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`.

`exclusionRadius` defines the number of library rows excluded from the state-space library with respect to a temporal "radius" from the prediction state. If `exclusionRadius = 1`, library state-space points from observation time series that are within ±1 sequential observation row of the prediction state are not included in the library. Note that units of the radius are time series rows, not time values.

`validLib` implements conditional embedding (CE). It is a boolean vector the same length as the number of time series rows. A `False` entry means that the state-space vector derived from the corresponding time series row will not be included in the state-space library.

If `noTime = False`, the first column of the input DataFrame or .csv file must be an index or time column. If `noTime = True` an index time column is created.

If `generateSteps > 0` data are generated from the specified library. The prediction indices (`pred`) are overridden to generate one prediction at the end of the library. Generative mode only works on univariate (`columns == target`) data with `embedded = False`.

If `generateConcat is True` data are generated from the specified library are appended to the input dataFrame Observations.

SMap

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

```
SMap( dataFrame       = None,
      columns         = "",
      target          = "",
      lib             = "",
      pred            = "",
      E               = 0,
      Tp              = 1,
      knn             = 0,
      tau             = -1,
      theta           = 0,
      exclusionRadius = 0,
      solver          = None,
      embedded        = False,
      validLib        = [],
      noTime          = False,
      generateSteps   = 0,
      generateConcat  = False,
      ignoreNan       = True,
      showPlot        = False,
      verbose         = False,
      returnObject    = False )
```

Return:

`dict` with three DataFrames:

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

The `predictions DataFrame` has 3 columns "Time", "Observations", "Predictions", “Pred_Variance”. nan values are inserted where there is no observation or prediction.

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 ∂ target / ∂ columns[i] .

The `singularValues DataFrame` holds SVD singular values if the default LAPACK solver, or scikit.learn solver is used. The scikit.learn LinearRegression does not return a singular value for the intercept (bias) term.

If `returnObject = True`, the SMap class object is returned. The attribute `Projection` contains the simplex projection DataFrame, `Coefficients` and `SingularValues` the corresponding DataFrames.

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 `columns` is a string and column names have whitespace place the column names in a list.

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`.

`exclusionRadius` defines the number of library rows excluded from the state-space library with respect to a temporal "radius" from the prediction state. If `exclusionRadius = 1,` library state-space points from observation time series that are within ±1 sequential observation row of the prediction state are not included in the library. Note that units of the radius are time series rows, not time values.

If `noTime = False`, the first column of the input DataFrame or .csv file must be an index or time column. If `noTime = True` an index or time column is not required.

`ignoreNan` removes nan to avoid nan observations and associated state vectors. If `ignoreNan` is `False` no nan are removed. The user can manually specify library row segments to ignore nan values.

`validLib` implements conditional embedding (CE). It is a boolean vector the same length as the number of time series rows. A `False` entry means that the state-space vector derived from the corresponding time series row will not be included in the state-space library.

The default solver for the SMap coefficient matrix is SciPy scipy.linalg.lstsq. This is a wrapper for the LAPACK function `dgelsd()`.

The solver can be replaced with a user-instantiated class object from the python `sklearn.linear_model: Linear Models`. Supported solvers include: `LinearRegression, SGDRegressor, Ridge, Lasso, LassoLars, OrthogonalMatchingPursuit, ElasticNet, RidgeCV, LassoCV, ElasticNetCV...` See the `pyEDM/tests/smapSolverTest.py` script for examples.

If `generateSteps > 0` data are generated from the specified library. The prediction indices (`pred`) are overridden to generate one prediction at the end of the library. Generative mode only works on univariate (`columns == target`) data with `embedded = False`. If `generateConcat is True` data are generated from the specified library are appended to the input dataFrame Observations.

CCM

Convergent cross mapping of `columns` against `target` via Simplex. Normally, one `column` and one `target` are specified. The `column` time series is time-delay embedded to dimension E, cross mapped with the `target` time series. The target time series is then embedded to E and cross mapped against the column as the "target" time series.

```
CCM( dataFrame        = None,
     columns          = "",
     target           = "",
     libSizes         = [],
     sample           = 0,
     E                = 0,
     Tp               = 0,
     knn              = 0,
     tau              = -1,
     exclusionRadius  = 0,
     seed             = None,
     embedded         = False,
     validLib         = [],
     includeData      = False,
     noTime           = False,
     ignoreNan        = True,
     verbose          = False,
     showPlot         = False,
     returnObject     = False )
```

Return:

DataFrame with `LibSize`, `column:target` and `target:column` CCM correlations.

If `includeData` is True, a `dict` is returned with the `LibMeans` `DataFrame` , and a `DataFrame` with prediction statistics of all predictions in each ensemble.

If there are multiple `columns` and `embedded` is false, each column is time-delay embedded to dimension `E` creating an N-columns * E dimensional "mixed" embedding. If `embedded` is true, no time-delay embedding is done, creating a multivariate embedding of the speficied columns. The same logic applies if multiple `target` are specified for the "reverse" mapping. If `embedded` is false, each target is time-delay embedded to dimension `E` creating an N-target * E dimensional "mixed" embedding cross mapped to the first `column` as the cross map target. If `embedded` is true, no time-delay embedding is done, creating a multivariate embedding of the speficied target(s).

Cross mappings are performed between `column : target`, and, `target : column` in separate multiprocessing instances. If additional multiprocessing is applied, halve the number of processors.

The `CCM_Matrix.py` application in `apps/` uses `ProcessPoolExecutor` to process parallelize CCM. All dataFrame columns are cross mapped to all others. A convergence metric is computed based on linear slope of the `CCM( libSize )`.

Parameters

`The libSizes` parameter is list of library sizes or a string of whitespace or comma separated library sizes. If the list has 3 values, and, if the third value is less than the second value, 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 `noTime = False`, the first column of the input DataFrame or .csv file must be an index or time column. If `noTime = True` an index or time column is not required.

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

Multiview

Multiview embedding and forecasting of DataFrame columns.

```
Multiview( dataFrame       = None,
           columns         = "",
           target          = "",
           lib             = "",
           pred            = "",
           D               = 0,
           E               = 1,
           Tp              = 1,
           knn             = 0,
           tau             = -1,
           multiview       = 0,
           exclusionRadius = 0,
           trainLib        = True,
           excludeTarget   = False,
           noTime          = False,
           ignoreNan       = True,
           verbose         = False,
           numProcess      = 4,
           showPlot        = False,
           returnObject    = False )
```

Return:

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

The `Predictions DataFrame` has 4 columns "Time", "Observations", "Predictions", “Pred_Variance”.

The `View DataFrame` will have 4 columns. The first column are tuples of column names for each multiview. The following 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`.

`numProcess` defines the number of multiprocessing instances.

If `noTime = False`, the first column of the input DataFrame or .csv file must be an index or time column. If `noTime = True` an index or time column is not required.

EmbedDimension

Evaluate Simplex prediction skill for embedding dimensions from 1 to `maxE`.

```
EmbedDimension( dataFrame       = None,
                columns         = "",
                target          = "",
                maxE            = 10,
                lib             = "",
                pred            = "",
                Tp              = 1,
                tau             = -1,
                exclusionRadius = 0,
                embedded        = False,
                validLib        = [],
                noTime          = False,
                ignoreNan       = True,
                verbose         = False,
                numProcess      = 4,
                showPlot        = True )
```

Return: `DataFrame` with columns "E" and "rho".

Note: Python multiprocessing is used to process parallelize over the embedding dimensions from 1 to `maxE`. `numProcess` defines the number of multiprocessing instances.

PredictInterval

Evaluate Simplex prediction skill for forecast intervals from 1 to `maxTp`.

```
PredictInterval( dataFrame       = None,
                 columns         = "",
                 target          = "",
                 lib             = "",
                 pred            = "",
                 maxTp           = 10,
                 E               = 1,
                 tau             = -1,
                 exclusionRadius = 0,
                 embedded        = False,
                 validLib        = [],
                 noTime          = False,
                 ignoreNan       = True,
                 verbose         = False,
                 numProcess      = 4,
                 showPlot        = True)
```

Return: `DataFrame` with columns "Tp" and "rho".

Note: Python multiprocessing is used to process parallelize over the forecast intervals from 1 to `maxTp`. `numProcess` defines the number of multiprocessing instances.

PredictNonlinear

Evaluate SMap prediction skill for localization parameter θ.

```
PredictNonlinear( dataFrame       = None,
                  columns         = "",
                  target          = "",
                  theta           = None,
                  lib             = "",
                  pred            = "",
                  E               = 1,
                  Tp              = 1,
                  knn             = 0,
                  tau             = -1,
                  exclusionRadius = 0,
                  solver          = None,
                  embedded        = False,
                  validLib        = [],
                  noTime          = False,
                  ignoreNan       = True,
                  verbose         = False,
                  numProcess      = 4,
                  showPlot        = True )
```

Return: `DataFrame with` columns "theta" and "rho".

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

Default `theta` values : `[0.01, 0.1, 0.3, 0.5, 0.75, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, 9]`

Note: Python multiprocessing is used to process parallelize over the localization parameter `theta`. `numProcess` defines the number of multiprocessing instances.

ComputeError

Compute Pearson correlation coefficient, maximum absolute error (MAE) and root mean square error (RMSE) between two vectors. A minimum of 5 data points is required. Any `nan` will be removed.

```
ComputeError( obsIn, predIn )
```

Return:

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

SurrogateData

Generate surrogate data using one of three methods.

```
SurrogateData( dataFrame     = None,
               column        = None,
               method        = 'ebisuzaki',
               numSurrogates = 10,
               alpha         = None,
               smooth        = 0.8,
               outputFile    = None )
```

Return:

`DataFrame` with `numSurrogates + 1` columns. First column is time vector from the input dataFrame. Remaining `numSurrogates` columns are the surrogate data with column names appended with `_1, _2…` for each surrogate.

Parameters

`method` :

1) random_shuffle

Sample the data with a uniform distribution.

2) ebisuzaki

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

Presumes data are serially correlated with low pass coherence. The method implements "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.

Application Notes

Input data files are Pandas DataFrame.

> **The DataFrame are required to have column names.**
>
> **It is expected the first column is a vector of times or time indices.** This can be disabled by setting the parameter `noTime = True`.

If `columns` is a string and column names have whitespace place the column names in a list.

`SMap()` should be called with `DataFrame` that have columns explicitly 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 not correspond to the intended dimensions in the matrix inversion and prediction reconstruction rendering the `SMap` coefficients invalid. In this case, use the `embedded` parameter set to `True` so columns correspond to the proper dimension.

Examples

```
from pyEDM import *

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

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

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

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

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

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

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

df = CCM( dataFrame = sampleData["sardine_anchovy_sst"],
          E = 3, Tp = 0, columns = "anchovy", target = "np_sst",
          libSizes = [10, 70, 10], 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.