# cppEDM Version 0.0.2  March 15, 2019

cppEDM is a C++ implementation of empirical dynamic modeling (EDM) algorithms.  It is designed as an application programming interface (API) with functions stored in the libEDM.a library.

## Table of Contents


Introduction....2
Installation....2
Class Objects....3
  DataFrame....3
  Parameters....5
Application Programming Interface (API)....6
  Embed....6
  Simplex....7
  SMap....8
  CCM....10
  EmbedDimension....12
  PredictInterval....13
  PredictNonlinear....14
  ComputeError....15
Application Notes....16
Example Application....17
Code Notes....18
References....18



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

Joseph Park, Cameron Smith

# Introduction

cppEDM is a C++ implementation of empirical dynamic modeling (EDM) algorithms. Core algorithms are listed in table 1. It is primarily a functional programming implementation with application programming interface (API) functions accepting parameters and returning data objects. EDM functions are accessed from a user-compiled library created from C++ source files and a unix-like compiler supporting the C++11 standard. cppEDM shares many high-level design attributes with the pyEDM package.

| 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 * Not yet implemented | | 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 |
| `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 correlation, MAE, RMSE | |

# Installation

cppEDM is available at github.com/SugiharaLab/cppEDM.

The libEDM.a library can be built by running "`make`" in the cppEDM/src/ directory. This copies libEDM.a into the cppEDM/lib/ directory, where it can be linked to user applications.

cppEDM requires a C++11 standard compiler, and the Eigen C++ template library (eigen.tuxfamily.org/).

Once libEDM.a is built, there are a series of test applications in the cppEDM/tests/ directory. The applications can be built with the "`make`" command, and then executed at the command line.

# Class Objects

Two C++ class objects are used for data access and parameter coordination, the `DataFrame` and `Parameters` classes, described below.

## DataFrame

The `DataFrame` class is the fundamental data object of cppEDM. It stores data in a contiguous block of memory using the C++ `valarray` type in a row-major format.

A `DataFrame` can be initialised with data from a csv file by calling the `DataFrame` constructor with `path` and `fileName` parameters. 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.**

The `WriteData(path, file)` class method can be called explicitly to write data to a csv format file. If the `DataFrame` does not have column names, then column names are created as V1, V2...

Primary `DataFrame` access functions are listed in table 3.

| DataFrame Method | Parameters | Type | Purpose |
|---|---|---|---|
| ( row, column ) | size_t row<br>size_t column | double or int | Access data element |
| DataFrame(path, file) | string path<br>string fileName | DataFrame<double> | Create DataFrame from csv file |
| WriteData(path, file) | string outputFilePath<br>string outputFileName | | Write DataFrame to file |
| Elements() | | valarray | Access data valarray |
| NColumns() | | size_t | Get number of columns |
| NRows() | | size_t | Get number of rows |
| size() | | size_t | Get number of elements |
| ColumnNames() | | vector< string > | Access column names |
| ColumnNameToIndex() | | map<string, size_t> | Access column name to index map |
| MaxRowPrint() | | size_t | Access maximum number of rows to ostream |
| Column( col ) | size_t col | valarray | Get data vector at column |
| Row( row ) | size_t row | valarray | Get data vector at row |
| VectorColumnName(column) | string column | valarray | Get data vector at column with name |
| DataFrameFromColumnIndex ( columns ) | vector<size_t> columns | DataFrame<double> | Get DataFrame subset from column indices |
| DataFrameFromColumnNames ( columns ) | vector<string> columns | DataFrame<double> | Get DataFrame subset from column names |
| WriteRow(row, array) | size_t row<br>std::valarray<T> array | | Write valarray to row |
| WriteColumn(col, array) | size_t col<br>valarray<T> array | | Write valarray to column |

## Parameters

The `Parameters` class is used to store and access API function parameters in a unified object. Generally this is an internal object that does not need to be instantiated, accessed or dynamically modified. API parameter names and purpose are listed in table 4.

| Parameter | Type | Default | Purpose |
|---|---|---|---|
| `pathIn` | `string` | `"./"` | Input data file path |
| `dataFile` | `string` | `""` | Data file name |
| `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 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.

```
//----------------------------------------------------------------
// Overload 1: Explicit data file path/name
//----------------------------------------------------------------
DataFrame< double > Embed ( std::string path     = "",
                            std::string dataFile = "",
                            int         E        = 0,
                            int         tau      = 0,
                            std::string columns  = "",
                            bool        verbose  = false );



//----------------------------------------------------------------
// Overload 2: DataFrame provided
//----------------------------------------------------------------
DataFrame< double > Embed ( DataFrame< double > dataFrame,
                            int                 E       = 0,
                            int                 tau     = 0,
                            std::string         columns = "",
                            bool                verbose = false );



//----------------------------------------------------------------
// Called from Embed to create the time-delay embedding
//----------------------------------------------------------------
DataFrame< double > MakeBlock ( DataFrame< double >      dataFrame,
                                int                      E,
                                int                      tau,
                                std::vector<std::string> columnNames,
                                bool                     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`.

```
//----------------------------------------------------------------
// Overload 1: Explicit data file path/name
//----------------------------------------------------------------
DataFrame<double> Simplex( std::string pathIn       = "./data/",
                           std::string dataFile     = "",
                           std::string pathOut      = "./",
                           std::string predictFile  = "",
                           std::string lib          = "1  10",
                           std::string pred         = "11 20",
                           int         E            = 0,
                           int         Tp           = 1,
                           int         knn          = 0,
                           int         tau          = 1,
                           std::string columns      = "",
                           std::string target       = "",
                           bool        embedded     = false,
                           bool        verbose      = true );



//----------------------------------------------------------------
// Overload 2: DataFrame provided
//----------------------------------------------------------------
DataFrame<double> Simplex( DataFrame< double >,
                           std::string pathOut      = "./",
                           std::string predictFile  = "",
                           std::string lib          = "1  10",
                           std::string pred         = "11 20",
                           int         E            = 0,
                           int         Tp           = 1,
                           int         knn          = 0,
                           int         tau          = 1,
                           std::string columns      = "",
                           std::string target       = "",
                           bool        embedded     = false,
                           bool        verbose      = true );
```

## SMap

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

`SMap()` returns a `SMapValues` structure:

```
struct SMapValues {
    DataFrame< double > predictions;
    DataFrame< double > coefficients;
};
```

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`.

```
//----------------------------------------------------------------
// Overload 1: Explicit data file path/name
//----------------------------------------------------------------
SMapValues SMap( std::string pathIn          = "./data/",
                 std::string dataFile        = "",
                 std::string pathOut         = "./",
                 std::string predictFile     = "",
                 std::string lib             = "1  10",
                 std::string pred            = "11 20",
                 int         E               = 0,
                 int         Tp              = 1,
                 int         knn             = 0,
                 int         tau             = 1,
                 double      theta           = 0,
                 std::string columns         = "",
                 std::string target          = "",
                 std::string smapFile        = "",
                 std::string jacobians       = "",   // Not implemented
                 bool        embedded        = false,
                 bool        verbose         = true );
```

```
//----------------------------------------------------------------
// Overload 2: DataFrame provided
//----------------------------------------------------------------
SMapValues SMap( DataFrame< double >,
                 std::string pathOut         = "./",
                 std::string predictFile     = "",
                 std::string lib             = "1  10",
                 std::string pred            = "11 20",
                 int         E               = 0,
                 int         Tp              = 1,
                 int         knn             = 0,
                 int         tau             = 1,
                 double      theta           = 0,
                 std::string columns         = "",
                 std::string target          = "",
                 std::string smapFile        = "",
                 std::string jacobians       = "",    // Not implemented
                 bool        embedded        = false,
                 bool        verbose         = true );
```

# CCM

Convergent cross mapping via Simplex of the first element in `columns` against `target`. 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`, N = `sample` 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.

```
//----------------------------------------------------------------
// Overload 1: Explicit data file path/name
//----------------------------------------------------------------
DataFrame<double> CCM( std::string pathIn       = "./data/",
                       std::string dataFile     = "",
                       std::string pathOut      = "./",
                       std::string predictFile  = "",
                       int         E            = 0,
                       int         Tp           = 0,
                       int         knn          = 0,
                       int         tau          = 1,
                       std::string columns      = "",
                       std::string target       = "",
                       std::string libSizes     = "",
                       int         sample       = 0,
                       bool        random       = true,
                       unsigned    seed         = 0,     // seed=0: use RNG
                       bool        embedded     = false,
                       bool        verbose      = true );
```

```
//-------------------------------------------------------------------
// Overload 2: DataFrame provided
//-------------------------------------------------------------------
DataFrame<double> CCM( DataFrame< double >,
                       std::string pathOut      = "./",
                       std::string predictFile  = "",
                       int         E            = 0,
                       int         Tp           = 0,
                       int         knn          = 0,
                       int         tau          = 1,
                       std::string columns      = "",
                       std::string target       = "",
                       std::string libSizes     = "",
                       int         sample       = 0,
                       bool        random       = true,
                       unsigned    seed         = 0,     // seed=0: use RNG
                       bool        embedded     = false,
                       bool        verbose      = true );
```

## 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.

```
//----------------------------------------------------------------
// Overload 1: Explicit data file path/name
//----------------------------------------------------------------
DataFrame<double> EmbedDimension( std::string pathIn      = "./data/",
                                  std::string dataFile    = "",
                                  std::string pathOut     = "./",
                                  std::string predictFile = "",
                                  std::string lib         = "1  10",
                                  std::string pred        = "11 20",
                                  int         Tp          = 1,
                                  int         tau         = 1,
                                  std::string columns     = "",
                                  std::string target      = "",
                                  bool        embedded    = false,
                                  bool        verbose     = true,
                                  unsigned    nThreads    = 4 );


//----------------------------------------------------------------
// Overload 2: DataFrame provided
//----------------------------------------------------------------
DataFrame<double> EmbedDimension( DataFrame< double >,
                                  std::string pathOut     = "./",
                                  std::string predictFile = "",
                                  std::string lib         = "1  10",
                                  std::string pred        = "11 20",
                                  int         Tp          = 1,
                                  int         tau         = 1,
                                  std::string columns     = "",
                                  std::string target      = "",
                                  bool        embedded    = false,
                                  bool        verbose     = true,
                                  unsigned    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.

```
//----------------------------------------------------------------
// Overload 1: Explicit data file path/name
//----------------------------------------------------------------
DataFrame<double> PredictInterval( std::string pathIn      = "./data/",
                                   std::string dataFile    = "",
                                   std::string pathOut     = "./",
                                   std::string predictFile = "",
                                   std::string lib         = "1  10",
                                   std::string pred        = "11 20",
                                   int         E           = 0,
                                   int         tau         = 1,
                                   std::string columns     = "",
                                   std::string target      = "",
                                   bool        embedded    = false,
                                   bool        verbose     = true,
                                   unsigned    nThreads    = 4 );



//----------------------------------------------------------------
// Overload 2: DataFrame provided
//----------------------------------------------------------------
DataFrame<double> PredictInterval( DataFrame< double >,
                                   std::string pathOut     = "./",
                                   std::string predictFile = "",
                                   std::string lib         = "1  10",
                                   std::string pred        = "11 20",
                                   int         E           = 0,
                                   int         tau         = 1,
                                   std::string columns     = "",
                                   std::string target      = "",
                                   bool        embedded    = false,
                                   bool        verbose     = true,
                                   unsigned    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.

```
//----------------------------------------------------------------
// Overload 1: Explicit data file path/name
//----------------------------------------------------------------
DataFrame<double> PredictNonlinear( std::string pathIn      = "./data/",
                                    std::string dataFile    = "",
                                    std::string pathOut     = "./",
                                    std::string predictFile = "",
                                    std::string lib         = "1  10",
                                    std::string pred        = "11 20",
                                    int         E           = 0,
                                    int         Tp          = 1,
                                    int         tau         = 1,
                                    std::string columns     = "",
                                    std::string target      = "",
                                    bool        embedded    = false,
                                    bool        verbose     = true,
                                    unsigned    nThreads    = 4 );


//----------------------------------------------------------------
// Overload 2: DataFrame provided
//----------------------------------------------------------------
DataFrame<double> PredictNonlinear( DataFrame< double >,
                                    std::string pathOut     = "./",
                                    std::string predictFile = "",
                                    std::string lib         = "1  10",
                                    std::string pred        = "11 20",
                                    int         E           = 0,
                                    int         Tp          = 1,
                                    int         tau         = 1,
                                    std::string columns     = "",
                                    std::string target      = "",
                                    bool        embedded    = false,
                                    bool        verbose     = true,
                                    unsigned    nThreads    = 4 );
```

## ComputeError

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

`ComputeError()` returns a `VectorError` struct:

```
struct VectorError {
    double rho;
    double RMSE;
    double MAE;
};

//--------------------------------------------------------------------
//--------------------------------------------------------------------
VectorError ComputeError( std::valarray< double > obsIn,
                          std::valarray< double > 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.

## Example Application

This application is assumed to be located in the etc/ directory. Otherwise, adust the -I and -L compiler flags and the Simplex path argument accordingly. The file etc/Test.cc shows sample invocations for API functions.

```
// g++ TestApp.cc -o TestApp -std=c++11 -g -I../src -L../lib -lstdc++ -lEDM

#include "Common.h"

//---------------------------------------------------------------
int main( int argc, char *argv[] ) {

    try {
        //-------------------------------------------------------
        // embedded=false : Simplex embeds data file columns to E=3
        //-------------------------------------------------------
        DataFrame<double> dataFrame =
            Simplex( "../data/", "block_3sp.csv",
                     "./", "Block3sp_E3.csv",
                     "1 100", "101 195", 3, 1, 0, 1,
                     "x_t y_t z_t", "x_t", false, true );

        dataFrame.MaxRowPrint() = 12; // Set number of rows to print
        std::cout << dataFrame;

        VectorError ve = ComputeError(
            dataFrame.VectorColumnName( "Observations" ),
            dataFrame.VectorColumnName( "Predictions"  ) );

        std::cout << "rho " << ve.rho << "  RMSE " << ve.RMSE
                  << "  MAE " << ve.MAE << std::endl << std::endl;

    }

    catch ( const std::exception& e ) {
      std::cout << "Exception caught in main:\n";
      std::cout << e.what() << std::endl;
      return -1;
    }
    catch (...) {
      std::cout << "Unknown exception caught in main.\n";
      return -1;
    }

    std::cout << "Normal termination.\n";

    return 0;
}
```

## Code Notes

1) The OSX XCode compiler/linker seems to be incompatible with the C++11 standard implementation allowing template classes to be distributed into declarations (.h) and implementation (.cc). To support OSX, DataFrame.h contains both declarations and implementations. See: etc/libstdc++_Notes.txt.

2) The code relies heavily on class and data containers without explicit heap allocation. This facilitates garbage collection. However, using copy-on-return for large data objects is likely a performance issue. If the code encounters massive data objects/large problems, this may warrant investigation.

3) Eigen template library. The recommended SVD solver is the BDCSVD that scales to large problems. However, the Eigen documentation states:

> This algorithm is unlikely to provide accurate results when compiled with unsafe math optimizations. For instance, this concerns Intel's compiler (ICC), which performs such optimization by default unless you compile with the -fp-model precise option. Likewise, the -ffast-math option of GCC or clang will significantly degrade accuracy.

See: eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra

However, when running the SMap circle.csv test, this solver (with default parameters) produces occasional divergence. We have therefore defaulted to the JacobiSVD solver.

4) Eigen template library. Eigen allows replacement of it's internal template library routines with direct calls to BLAS/LAPACK libraries. See: https://eigen.tuxfamily.org/dox/TopicUsingBlasLapack.html This may offer performance and stability advantages.

## References


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.

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

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

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