

## cppEDM Version 1.3.4 April 15, 2020

cppEDM is a C++ implementation of empirical dynamic modeling (EDM) algorithms. It is designed as an application programming interface (API) to functions 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.....	11
Multiview.....	13
EmbedDimension.....	15
PredictInterval.....	16
PredictNonlinear.....	17
ComputeError.....	18
Application Notes.....	19
Example Application.....	20
Code Notes.....	21
References.....	21

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 devEDM Python package. cppEDM has Python and R interfaces with the pyEDM and EDM packages.

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
EmbedDimension()	Evaluate prediction skill vs. embedding dimension	$E = [1, 10]$
PredictInterval()	Evaluate prediction skill vs. forecast interval	$T_p = [1, 10]$
PredictNonlinear()	Evaluate prediction skill vs. SMap nonlinear localisation	$\theta = 0.01, 0.1, 0.3, 0.5, 0.75, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, 9$
ComputeError()	Pearson correlation, MAE, RMSE	

## Installation

cppEDM is available at [github.com/SugiharaLab/cppEDM](https://github.com/SugiharaLab/cppEDM).

cppEDM requires a C++11 standard compiler, and the LAPACK library. 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.

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 executed at the command line. API examples can also be found in cppEDM/etc/Test.cc.

## 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. 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 assumed that the first column of the csv file is a vector of times or time indices for each observation (row). All subsequent columns are expected to be numeric. However, a `DataFrame` can be created with the `noTime = true` parameter to read in numeric data with no time in the first column. Such data frames should not be passed to prediction functions.

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 size_t column	double or int	Access data element
DataFrame(path, file)	string path string fileName	DataFrame<double>	Create DataFrame from csv file
WriteData(path, file)	string outputFilePath 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
ColumnMajorData()		valarray	Elements() in column major format.
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 std::valarray<T> array		Write valarray to row
WriteColumn(col, array)	size_t col 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
<code>pathIn</code>	string	<code>"./"</code>	Input data file path
<code>dataFile</code>	string	<code>""</code>	Data file name
<code>pathOut</code>	string	<code>"./"</code>	Output file path
<code>predictFile</code>	string	<code>""</code>	Prediction output file
<code>lib</code>	string	<code>""</code>	library start : stop row indices
<code>pred</code>	string	<code>""</code>	prediction start : stop row indices
<code>E</code>	int	<code>0</code>	Data dimension
<code>TP</code>	int	<code>0 or 1</code>	Prediction interval (rows)
<code>knn</code>	int	<code>0</code>	Number nearest neighbors
<code>tau</code>	int	<code>-1</code>	Embedding delay (rows)
<code>theta</code>	double	<code>0</code>	SMap localisation
<code>exclusionRadius</code>	int	<code>0</code>	Prediction vector exclusion row radius
<code>columns</code>	string	<code>""</code>	Column names or indices for prediction
<code>target</code>	string	<code>""</code>	Target library column name or index
<code>embedded</code>	bool	<code>false</code>	Is data an embedding?
<code>const_pred</code>	bool	<code>false</code>	Include non-projected forecast data
<code>verbose</code>	bool	<code>false</code>	Echo messages
<code>smapFile</code>	string	<code>""</code>	SMap coefficient output file
<code>multiview</code>	int	<code>0</code>	Number of ensembles, $0 = \sqrt{N}$
<code>libSizes_str</code>	string	<code>""</code>	CCM library sizes
<code>sample</code>	int	<code>0</code>	CCM number of random samples
<code>random</code>	bool	<code>true</code>	CCM use random samples?
<code>replacement</code>	bool	<code>false</code>	CCM sample with replacement?
<code>seed</code>	unsigned	<code>0</code>	RNG seed, $0 = \text{random seed}$

# 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  $\tau \cdot (E - 1)$  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     = -1,  
                           std::string columns = "",  
                           bool            verbose = false );  
  
//-----  
// Overload 2: DataFrame provided  
//-----  
DataFrame< double > Embed ( DataFrame< double > dataframe,  
                           int             E      = 0,  
                           int             tau     = -1,  
                           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.

### 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.

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

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



## SMap

```
//-----  
// 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            = "",  
                  std::string pred           = "",  
                  int E                     = 0,  
                  int Tp                    = 1,  
                  int knn                   = 0,  
                  int tau                   = -1,  
                  double theta              = 0,  
                  int exclusionRadius        = 0,  
                  std::string columns        = "",  
                  std::string target         = "",  
                  std::string smapFile       = "",  
                  std::string derivatives    = "", // Not implemented  
                  bool embedded              = false,  
                  bool const_pred            = false,  
                  bool verbose               = true );
```

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

## SMap

```
//-----  
// Overload 3: Data path/file with external solver object, init to default SVD  
//-----  
SMapValues SMap( std::string pathIn          = "./data/",  
                  std::string dataFile       = "",  
                  std::string pathOut        = "./",  
                  std::string predictFile    = "",  
                  std::string lib            = "",  
                  std::string pred           = "",  
                  int      E                 = 0,  
                  int      Tp                = 1,  
                  int      knn               = 0,  
                  int      tau               = -1,  
                  double   theta             = 0,  
                  int      exclusionRadius    = 0,  
                  std::string columns         = "",  
                  std::string target          = "",  
                  std::string smapFile        = "",  
                  std::string derivatives     = "", // Not implemented  
                  std::valarray<double> (*solver)(DataFrame < double >,  
                                                    std::valarray < double >) = &SVD,  
                  bool      embedded          = false,  
                  bool      const_predict     = false,  
                  bool      verbose          = true );  
  
//-----  
// Overload 4: DataFrame with external solver object, init to default SVD  
//-----  
SMapValues SMap( DataFrame< double > &dataFrameIn,  
                  std::string pathOut        = "./",  
                  std::string predictFile    = "",  
                  std::string lib            = "",  
                  std::string pred           = "",  
                  int      E                 = 0,  
                  int      Tp                = 1,  
                  int      knn               = 0,  
                  int      tau               = -1,  
                  double   theta             = 0,  
                  int      exclusionRadius    = 0,  
                  std::string columns         = "",  
                  std::string target          = "",  
                  std::string smapFile        = "",  
                  std::string derivatives     = "", // Not implemented  
                  std::valarray<double> (*solver)(DataFrame < double >,  
                                                    std::valarray < double >) = &SVD,  
                  bool      embedded          = false,  
                  bool      const_predict     = false,  
                  bool      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.

CCM() returns a CCMValues structure:

```
struct CCMValues {
    DataFrame< double > AllLibStats; // unified mean libsize, rho, RMSE, MAE
    CrossMapValues      CrossMap1;
    CrossMapValues      CrossMap2;
};
struct CrossMapValues {
    DataFrame< double > LibStats; // mean libsize, rho, RMSE, MAE
    DataFrame< double > PredictStats; // each predict libsize, rho, RMSE, MAE
    std::forward_list< DataFrame< double > > Predictions;
};
```

CCMValues::LibStats 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 the includeData parameter is true, then the CrossMapValues structures CrossMap1 and CrossMap2 in CCMValues are populated with the DataFrame PredictStats and list of DataFrames Predictions. The LibStats is used for internal storage, and is empty upon return.

## Parameters

The libSizes parameter 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 randomly selected from the subset of each library size. If replacement is true, observation are selected with replacement. 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
//-----
CCMValues 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,
               bool         replacement = false,
               unsigned     seed       = 0,      // seed=0: use RNG
               bool         includeData = false,
               bool         verbose    = true );

//-----
// Overload 2: DataFrame reference provided
//-----
CCMValues CCM( DataFrame< double > &dataFrameIn,
               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,
               bool         replacement = false,
               unsigned     seed       = 0,      // seed=0: use RNG
               bool         includeData = false,
               bool         verbose    = true );

```

## Multiview

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

Multiview() returns a MultiviewValues structure:

```
struct MultiviewValues {  
    DataFrame< double > Combo_rho;  
    DataFrame< double > Predictions;  
    std::vector< std::string > Combo_rho_table;  
};
```

The Combo\_rho DataFrame has E+3 columns. The first E columns are the the column indices in the input data DataFrame embedding (not the input DataFrame) and are applied to multivariate Simplex predictions. The last three columns are "rho", "MAE", "RMSE" corresponding to the prediction Pearson correlation, maximum absolute error and root mean square error.

Combo\_rho\_table is a list of strings that include the multiview embedding column names in addition to the column indices and prediction statistics.

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.

## 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.

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.

```

//-----
// Overload 1: Explicit data file path/name
//-----
MultiviewValues Multiview( std::string pathIn          = "./",
                           std::string dataFile        = "",
                           std::string pathOut         = "./",
                           std::string predictFile     = "",
                           std::string lib             = "",
                           std::string pred            = "",
                           int D                      = 0,
                           int E                      = 1,
                           int Tp                     = 1,
                           int knn                    = 0,
                           int tau                    = -1,
                           std::string columns         = "",
                           std::string target          = "",
                           int multiview              = 0,
                           int exclusionRadius        = 0,
                           bool verbose               = false,
                           unsigned nThreads          = 4 );

//-----
// Overload 2: DataFrame provided
//-----
MultiviewValues Multiview( DataFrame< double >,
                           std::string pathOut        = "./",
                           std::string predictFile    = "",
                           std::string lib            = "",
                           std::string pred           = "",
                           int D                     = 0,
                           int E                     = 1,
                           int Tp                    = 1,
                           int knn                   = 0,
                           int tau                   = -1,
                           std::string columns        = "",
                           std::string target         = "",
                           int multiview              = 0,
                           int exclusionRadius        = 0,
                           bool verbose               = false,
                           unsigned nThreads          = 4 );

```

## EmbedDimension

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

```
//-----  
// 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          = "",  
                                  std::string pred         = "",  
                                  int maxE                = 10,  
                                  int Tp                 = 1,  
                                  int tau                 = -1,  
                                  std::string columns      = "",  
                                  std::string target       = "",  
                                  bool embedded           = false,  
                                  bool verbose            = true,  
                                  unsigned nThreads       = 4 );  
  
//-----  
// Overload 2: DataFrame reference provided  
//-----  
DataFrame<double> EmbedDimension( DataFrame< double > &dataFrameIn,  
                                  std::string pathOut      = "./",  
                                  std::string predictFile  = "",  
                                  std::string lib          = "",  
                                  std::string pred         = "",  
                                  int maxE                = 10,  
                                  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 maxTp (default 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 maxTp.

```
//-----  
// 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          = "",  
                                   std::string pred         = "",  
                                   int maxTp               = 10,  
                                   int E                  = 0,  
                                   int tau                 = -1,  
                                   std::string columns     = "",  
                                   std::string target      = "",  
                                   bool embedded           = false,  
                                   bool verbose            = true,  
                                   unsigned nThreads       = 4 );
```

```
//-----  
// Overload 2: DataFrame reference provided  
//-----  
DataFrame<double> PredictInterval( DataFrame< double > &dataFrameIn,  
                                   std::string pathOut      = "./",  
                                   std::string predictFile  = "",  
                                   std::string lib          = "",  
                                   std::string pred         = "",  
                                   int maxTp               = 10,  
                                   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 parameters  $\theta$  specified in the string theta. Default values of theta are "0.01 0.1 0.3 0.5 0.75 1 1.5 2 3 4 5 6 7 8 9". Default knn (0) sets knn to size of the library. Smaller values can be specified.

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  $\theta$  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         = "",  
                                     std::string pred        = "",  
                                     std::string theta       = "",  
                                     int E                  = 0,  
                                     int Tp                 = 1,  
                                     int knn                = 0,  
                                     int tau                = -1,  
                                     std::string columns    = "",  
                                     std::string target     = "",  
                                     bool embedded          = false,  
                                     bool verbose           = true,  
                                     unsigned nThreads      = 4 );  
  
//-----  
// Overload 2: DataFrame reference provided  
//-----  
DataFrame<double> PredictNonlinear( DataFrame< double > &dataFrameIn,  
                                     std::string pathOut     = "./",  
                                     std::string predictFile = "",  
                                     std::string lib         = "",  
                                     std::string pred        = "",  
                                     std::string theta       = "",  
                                     int E                  = 0,  
                                     int Tp                 = 1,  
                                     int knn                = 0,  
                                     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 assumed that the first column of the csv file is a vector of times or time indices for each observation (row). All subsequent columns are expected to be numeric. However, a `DataFrame` can be created with the `noTime = true` parameter to read in numeric data with no time in the first column. Such data frames should not be passed to prediction functions.

`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 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.

See `etc/Notes` for details regarding the building an application (`etc/Test.cc`) on Windows.

See `etc/Notes` for an explanation of differences between `Simplex()` and `CCM()` default embeddings, and the effect this has on predictions.

## Example Application

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

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

```
#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/",           // pathIn
                    "block_3sp.csv",       // dataFile
                    "./",                  // pathOut
                    "Block3sp_E3.csv",     // predictFile
                    "1 100",               // lib
                    "101 195",             // pred
                    3,                     // E
                    1,                     // Tp
                    0,                     // knn
                    -1,                    // tau
                    0,                     // exclusionRadius
                    "x_t y_t z_t",         // columns
                    "x_t",                 // target
                    false,                 // embedded
                    false,                 // const_predict
                    true );                 // verbose

        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) SMap uses the LAPACK (<http://www.netlib.org/lapack/explore-html/index.html>) routine `dgeels()` to solve the local linear maps by singular value decomposition.

## 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.