Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

rEDM: an R package for Empirical Dynamic Modeling and Convergent cross mapping

Hao Ye
Scripps Institution of Oceanography

Adam T. Clark
University of Minnesota

Ethan R. Deyle
Scripps Institution of Oceanography

George Sugihara
Scripps Institution of Oceanography

Abstract

Empirical dynamic modeling (EDM) is an emerging non-parametric framework for modeling nonlinear dynamic systems. EDM is based on the mathematical theory of reconstructing attractor manifolds from time series data (Takens 1981). The **rEDM** package collects several EDM methods, including simplex projection (Sugihara and May 1990), S-map (Sugihara 1994), multivariate embeddings (Dixon, Milicich, and Sugihara 1999), convergent cross mapping (Sugihara, May, Ye, Hsieh, Deyle, Fogarty, and Munch 2012), and multiview embedding (Ye and Sugihara 2016). Here, we introduce the basic underlying theory, and describe the functionality of the **rEDM**, using examples from both model simulations and real data.

Keywords: empirical dynamics, convergent cross mapping, causation, R.

1. Introduction

Many scientific fields rely on models as simplifications of reality. These models are used for various purposes (e.g. testing hypotheses regarding mechanisms or processes, explaining past observations, predicting future outcomes), but are typically based on hypothesized parametric equations. In many cases, however, using models with explicit equations is impractical because the exact mechanisms are unknown or too complex to be characterized with existing datasets. Empirical models, that infer patterns and associations from the data (instead of using hypothesized equations), represent an alternative and highly flexible approach. Here, we review the theoretical background for the emerging framework of empirical dynamic modeling (EDM) and the functionality of the **rEDM** package, which are intended for the study nonlinear dynamic systems that can prove problematic for traditional modeling approaches.

The basic goal underlying EDM is to reconstruct the behavior of dynamic systems using time

series data. This approach is based on mathematical theory developed initially by Takens (1981), and expanded on by others (Sauer, Yorke, and Casdagli 1991; Casdagli, Eubank, Farmer, and Gibson 1991; Deyle and Sugihara 2011). Because these methods operate with minimal assumptions, they are particularly suitable for studying systems that exhibit non-equilibrium dynamics and nonlinear state-dependent behavior (i.e. where interactions change over time and as a function of the system state).

1.1. Time Series as Observations of a Dynamic System

The essential concept is that time series can be viewed as projections of the behavior of a dynamic system. Here, only a few modest assumptions are required. First, that the system state can be described as a point in a high-dimensional space. The axes of this space can be thought of as fundamental state variables; in an ecosystem, these variables might correspond to population abundances, resources, or environmental conditions. Second, that the system state changes through time following a set of deterministic rules. In other words, the behavior of the system is not completely stochastic.

Consequently, it is possible to project the system state onto one of the coordinate axes to obtain the value of the corresponding state variable. Thus, sequential projections over time will form a time series for that variable. For example, in Figure 1, the states of the canonical Lorenz Attractor (Lorenz 1963) are projected to the x-axis, creating a time series for variable x.

Although different time series observed from the system can represent different state variables, in general, each time series is a function of the system state and may potentially convolve several different state variables.

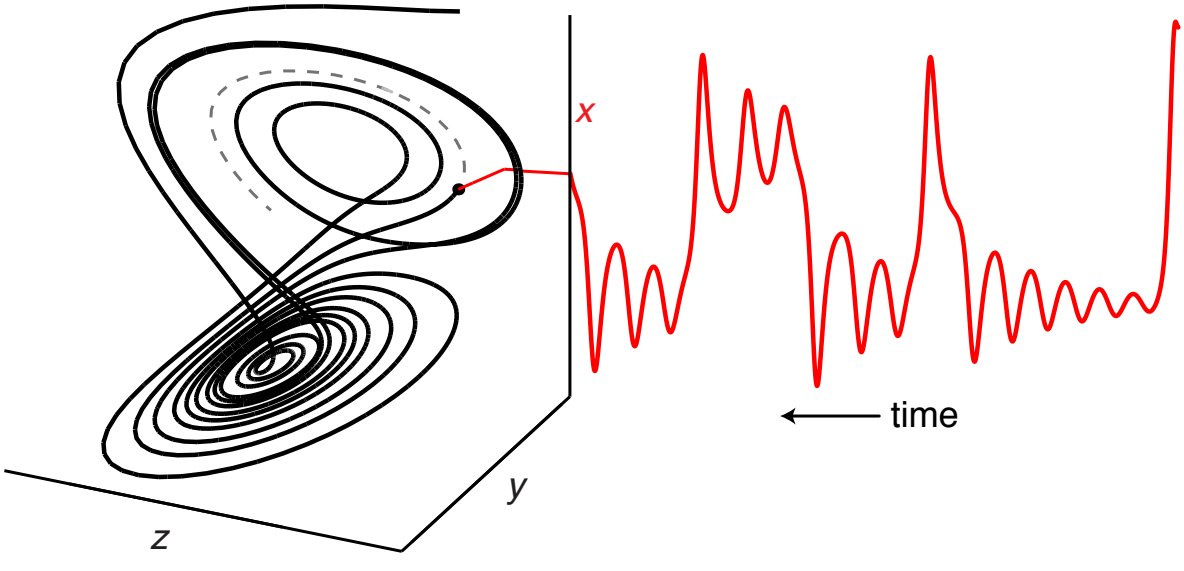


Figure 1: **Time Series Projection from Lorenz Attractor.**
Projecting the motion of the canonical Lorenz attractor onto the x-axis yields a time series (red) for variable x.

1.2. Attractor Reconstruction / Takens' Theorem

The goal of EDM is to reconstruct the system dynamics from time series data. As seen above, a time series can be thought of as sequential projections of the motion on an attractor; in other words, information about the behavior is encoded in the temporal ordering of the time series. Takens' Theorem (Takens 1981) states that mathematically valid and equivalent reconstructions of the attractor can be created using lags of just a single time series, by substituting those lags for unknown or unobserved variables. In other words, instead of representing the system state using a complete set of state variables, we can instead use a lagged-coordinate embedding:

$$\vec{x}_t = \langle x_t, x_{t-\tau}, \ldots, x_{t-(E-1)\tau} \rangle$$

If sufficient lags are used, the reconstruction preserves essential mathematical properties of the original system: reconstructed states will map one-to-one to actual system states, and nearby points in the reconstruction will correspond to similar system states. Figure 2 shows a reconstruction of the Lorenz attractor (from Figure 1) where the reconstructed system state is comprised of 3 lags of variable x. Here, the visual similarity between the reconstruction and the original Lorenz Attractor is quite clear.

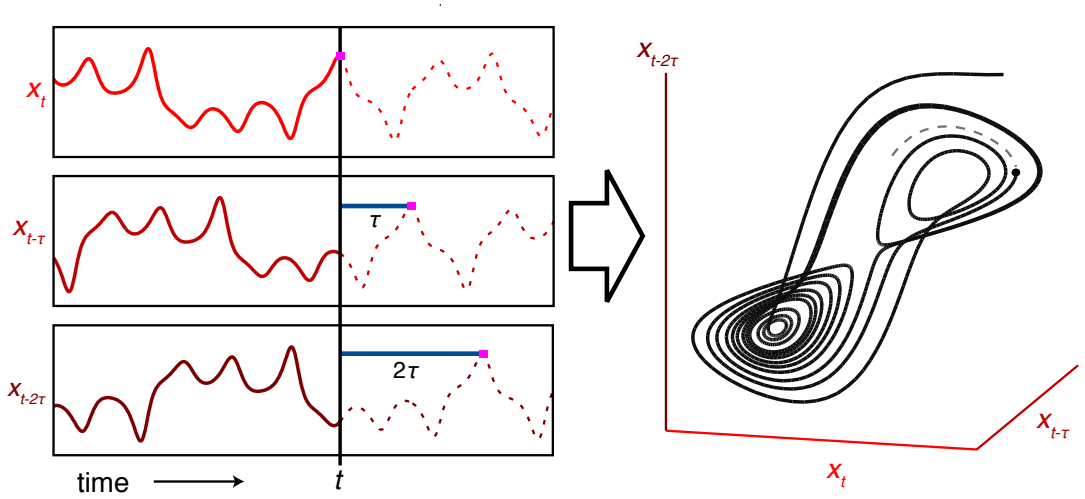


Figure 2: **Attractor Reconstruction.**
Successive lags (with time step τ) of the time series x_t are plotted as separate coordinates to form a reconstructed "shadow" attractor, which appears similar to the original attractor in figure 1

As a consequence of the fact that these mathematical properties of the original system can be recovered from a single time series, there are multiple applications. For example, empirical models can be used for forecasting (Sugihara and May 1990), to understand nonlinear behavior (Sugihara 1994), or to uncover mechanism (Dixon *et al.* 1999). Moreover, recent work describes how EDM can be used to identify causal interactions, by testing whether two time series are observed from the same system (Sugihara *et al.* 2012). In the next section, we demonstrate how the **rEDM** software package can be used to accomplish these various tasks.

2. Demonstration of EDM

2.1. Nearest Neighbor Forecasting using Simplex Projection

As mentioned previously, the reconstruction will map one-to-one to the original attractor manifold if enough lags are used (i.e. if the reconstruction has a sufficiently large embedding dimension). If the embedding dimension is too small, then reconstructed states will overlap and appear to be the same even though they actually correspond to different states. These "singularities" will result in poor forecast performance, because the system behavior cannot be uniquely determined in the reconstruction. Thus, we can use prediction skill as an indicator for identifying the optimal embedding dimension. In the following example, we demonstrate how this can be accomplished using a nearest neighbor forecastaing method, Simplex Projection (Sugihara and May 1990), implemented in **rEDM** as the function `simplex`.

Example

In this example, we use simulated time series from the classical tent map that exhibits chaotic behavior. The tent map is a discrete time dynamic system, where a sequence, x_t, on the interval $[0, 1]$ is iterated according to:

$$x_{t+1} = \begin{cases} 2x_t & x_t < \frac{1}{2} \\ 2(1 - x_t) & x_t \geq \frac{1}{2} \end{cases}$$

In **rEDM**, a sample time series of the first-differenced values can be found in dataset `tentmap_del`.

```
> library(rEDM)
> data(tentmap_del)
> str(tentmap_del)

 num [1:999] -0.0992 -0.6013 0.7998 -0.7944 0.798 ...
```

We can see that the data consists of just a single vector, containing the raw first-differences values of x_t. Because the `simplex` function can accept a single vector as the input time series, no further processing of the data is required.

```
> ts <- tentmap_del
> lib <- c(1, 100)
> pred <- c(201, 500)
```

We begin by initializing the `lib` and `pred` variables. These determine which portions of the data will be used to create the reconstruction, and which portions of the data the reconstruction will be used to make forecasts on. In other words, defining the "training" and "test" subsets of the data. Here, the first 100 points (rows 1 to 100) in the time series constitute the "library", and 300 points (rows 201 to 500) are the "prediction set" for which the model will produce forecasts.

The remaining parameters will be left at their default values (see section 4.2 for details). For the `simplex` function, this means that the embedding dimension, E, will range from 1 to 10.

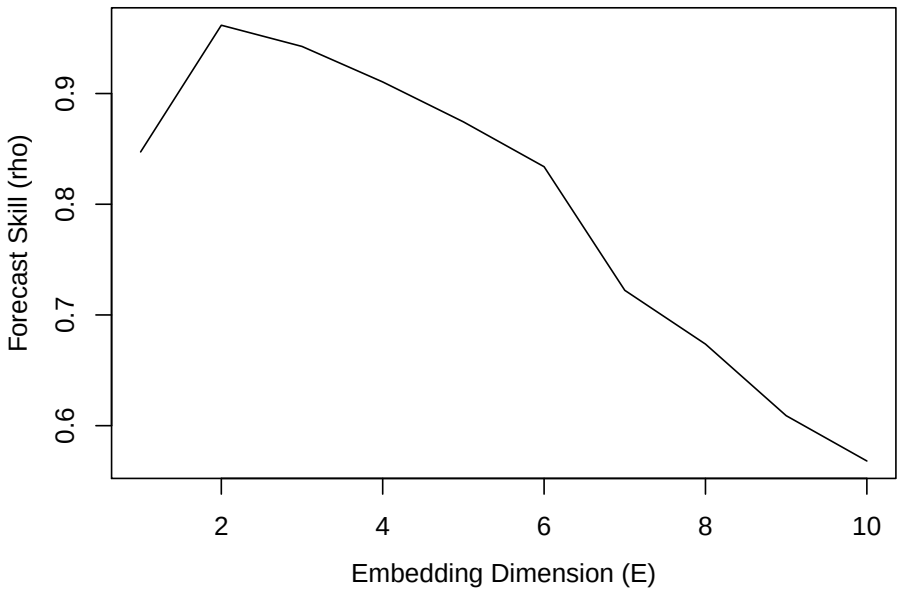
```
> simplex_output <- simplex(ts, lib, pred)
> str(simplex_output)
```

```
'data.frame':        10 obs. of  16 variables:
 $ E                   : int  1 2 3 4 5 6 7 8 9 10
 $ tau                 : num  1 1 1 1 1 1 1 1 1 1
 $ tp                  : num  1 1 1 1 1 1 1 1 1 1
 $ nn                  : num  2 3 4 5 6 7 8 9 10 11
 $ num_pred            : num  299 298 297 296 295 294 293 292 291 290
 $ rho                 : num  0.847 0.962 0.943 0.91 0.874 ...
 $ mae                 : num  0.207 0.104 0.138 0.19 0.235 ...
 $ rmse                : num  0.392 0.189 0.235 0.291 0.334 ...
 $ perc                : num  0.853 0.906 0.899 0.885 0.824 ...
 $ p_val               : num  2.67e-102 9.18e-251 2.13e-200 1.54e-151 2.59e-118 ...
 $ const_pred_num_pred: num  299 298 297 296 295 294 293 292 291 290
 $ const_pred_rho      : num  -0.668 -0.671 -0.671 -0.673 -0.671 ...
 $ const_pred_mae      : num  1.02 1.02 1.02 1.02 1.01 ...
 $ const_pred_rmse     : num  1.25 1.25 1.26 1.26 1.25 ...
 $ const_pred_perc     : num  0.341 0.339 0.337 0.338 0.339 ...
 $ const_p_val         : num  1 1 1 1 1 1 1 1 1 1
```

The returned object from `simplex` is a simple data.frame with columns for each of the model parameters and forecast statistics, and rows for each separate model (i.e. different parameter combinations). For `simplex`, the model parameters are `E`, embedding dimension; `tau`, time lag between successive dimensions; `tp`, time to prediction; and `nn`, number of nearest neighbors (see section 4.2 for a detailed description). The forecast statistics are `num_pred`, the number of predictions made; `rho`, Pearson's correlation coefficient between predictions and observations; `mae`, mean absolute error of predictions; `rmse`, root mean squared error of predictions; `perc`, the percent of predictions that are the same sign as observations; and `p_val`, the p-value for `rho` being significantly greater than 0, using Fisher's transformation (Fisher 1915). The last 6 columns give those same forecast statistics, but for a naïve constant predictor (where the forecast is simply the current value) over the same set of predictions.

In this case, there are 10 separate models (one for each value of `E`), so we can simply plot `E` against `rho` (the correlation between observed and predicted values) to determine the optimal embedding dimension (i.e. the number of dimensions for which the reconstructed attractor is best unfolded, producing the highest forecast skill).

```
> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0)) # set up margins for plotting
> plot(simplex_output$E, simplex_output$rho, type = "l",
+      xlab = "Embedding Dimension (E)", ylab = "Forecast Skill (rho)")
```



Here, we observe that forecast skill peaks at $E = 2$, indicating that the dynamics of our data are unfolded best in 2 dimensions. Note that this optimal value for `E` may not be the same as the actual dimensionality of the corresponding dynamic systems, since predictability will be limited by observational error, process noise, and time series length.

2.2. Prediction Decay

An important property of many natural systems is that nearby trajectories eventually diverge over time (i.e. "deterministic chaos" or the so-called "butterfly effect"). In essence, this means that short-term prediction is possible, because the near future is well-constrained, but in the long-term, states of the system are essentially random and not predictable. We can demonstrate this effect by examining how prediction skill changes as the `tp` parameter is increased; recall that this parameter describes the forecast horizon for how far ahead into the future forecasts are made.

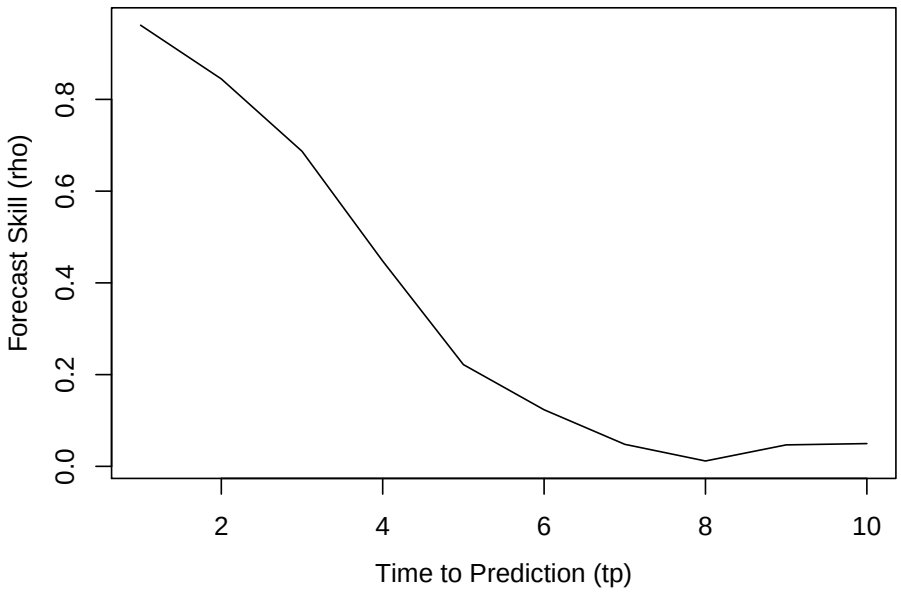
Example

Using the same tent map time series, we supply a range for the `tp` parameter in the `simplex` function, but fix the embedding dimension to the value determined above ($E = 2$):

```
> simplex_output <- simplex(ts, lib, pred, E = 2, tp = 1:10)
```

As above, the returned object is a data.frame, and the prediction decay can be studied by plotting forecast skill (`rho`) against the time to prediction (`tp`).

```
> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0)) # set up margins for plotting
> plot(simplex_output$tp, simplex_output$rho, type = "l",
+      xlab = "Time to Prediction (tp)", ylab = "Forecast Skill (rho)")
```



As expected (because the parameters chosen for the tent map fall in the region for chaotic behavior), the clear decline in forecast skill (`rho` $\rightarrow 0$) indicates that the system is chaotic.

2.3. Identifying Nonlinearity

One concern is that many time series may show predictability even if they are purely stochastic, because they behave similarly to autocorrelated red noise. Fortunately, we can distinguish between red noise and nonlinear deterministic behavior by using S-maps as described in Sugihara (1994).

In contrast to the nearest-neighbor interpolation of simplex projection, the S-map forecasting method (Sugihara 1994) fits local linear maps to describe the dynamics. In addition to the standard set of parameters for a lagged-coordinate reconstruction (as in `simplex`), S-maps also contain a nonlinear tuning parameter, θ, that determines the degree to which points are weighted when fitting the local linear map. For example, when $\theta = 0$, all points are equally weighted, such that the local linear map is identical for different points in the reconstructed state-space. As such, the S-map will be identical to a global linear map (i.e. an autoregressive model). When values of θ are greater than 0, nearby points in the state space receive larger weights, and the local linear map can vary in state-space to accommodate nonlinear behavior.

Consequently, if the time series are sampled from autoregressive red noise, then the linear model ($\theta = 0$) should produce better forecasts, because the global linear map (which will, in effect, be fitted to more data points) will reduce the effects of observation error compared to local linear maps. In contrast, if forecast skill increases for $\theta > 0$, then the results are suggestive of nonlinear dynamics, because better forecasts are achieved when the local linear map can change depending on the location in state-space: it is a better description of state-dependent behavior.

Example

The S-map method is implemented as the function `s_map` in the **rEDM** package. Much like

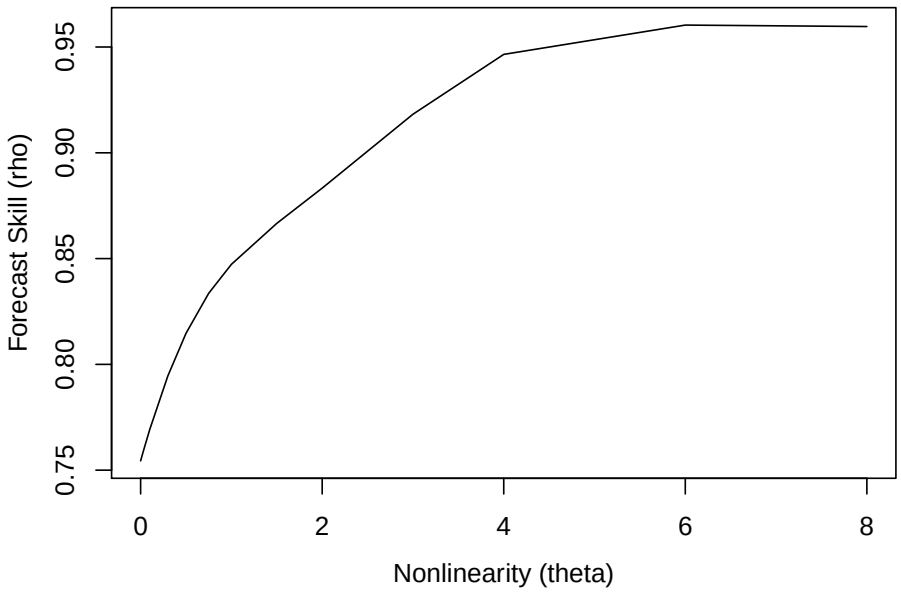
in the previous use of `simplex`, we can leave many of the parameters at default settings (see section 4.2 for details). Here, by default the θ parameter (`theta`) will range from 0 to 8, enabling us to test for nonlinearity. Here again, we will use the tent map time series, and set $E = 2$ based on the results from simplex projection.

Note that the default value for `num_neighbors` is `0`. Typically, when using `s_map` to test for nonlinear behavior, we want to use all points in the reconstruction when constructing the local linear map, and allow the `theta` parameter to control the weighting assigned to individual points. This is in contrast to simplex projection, where only the nearest neighbors are used. Here the function recognizes that all values < 1 are nonsensical and all the points will be used instead.

```
> smap_output <- s_map(ts, lib, pred, E = 2)
```

Again, the results are a simple data.frame with columns for each of the model parameters and forecast statistics, with rows for each run of the model. In this case, there is one run for each value of `theta`, so we can simply plot `theta` against `rho`:

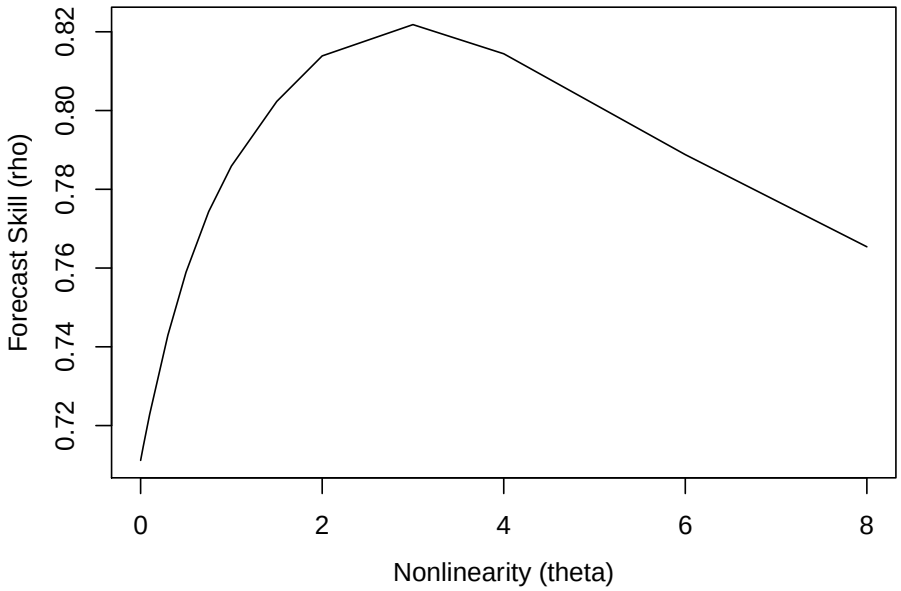
```
> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0)) # set up margins for plotting
> plot(smap_output$theta, smap_output$rho, type = "l",
+      xlab = "Nonlinearity (theta)", ylab = "Forecast Skill (rho)")
```



Here, we can see that forecast skill substantially improves as `theta` increases, indicating the presence of nonlinear dynamics. Typically, we would expect forecast skill to decrease at high values of `theta`, because as `theta`, the local linear map can become overfitted to the nearest points. However, because the example data are observed without any error, even a very localized linear map can continue to produce good forecasts. By simulating the addition of a small amount of observational error, a more typical `rho` vs. `theta` plot can be achieved:

```
> ts <- ts + rnorm(length(ts), sd = sd(ts) * 0.2)
> smap_output <- s_map(ts, lib, pred, E = 2)
```

```
> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0)) # set up margins for plotting
> plot(smap_output$theta, smap_output$rho, type = "l",
+      xlab = "Nonlinearity (theta)", ylab = "Forecast Skill (rho)")
```



2.4. Generalized Takens' Theorem

Instead of creating an attractor by taking lags of a single time series, it is possible to combine lags from different time series, if they are all observed from the same system (Sauer *et al.* 1991; Deyle and Sugihara 2011). The practical reality of applying EDM to systems with finite data, noisy observations, and stochastic influences means that such "multivariate" reconstructions can often be a better description of the true dynamics than "univariate" reconstructions

In **rEDM**, the `block_lnlp` function generalizes the `simplex` and `s_map` functions, allowing generic reconstructions to be used with either the simplex projection or S-map methods The main data input for `block_lnlp` is a matrix or data.frame of the time series observations, where each column is a separate time series with rows representing "simultaneous" observations. In addition to the standard parameters for `simplex` or `s_map`, `block_lnlp` contains parameters to specify which column is to be forecast (`target_column`) as well as which columns to use to construct the attractor (`columns`). For both parameters, either a numerical index or the column name can be given. Note that if lagged coordinates are intended to be used, they need to be manually created as separate columns in the matrix or data.frame.

Example

We begin by loading an example dataset of time series and lags from a coupled 3-species model system. Here, the `block_3sp` variable is a 10-column data.frame with 1 column for time, and 3 columns for each of the variables (unlagged, $t-1$, and $t-2$ lags). Note that the lagged columns begin with `NA` values because there are no observations of the variables for times $t < 1$. In **rEDM**, vectors that include `NA` values are excluded if they would be used for

computation, but predictions will still be made if only the observed values are missing (see section 4.1 for more details).

```
> data(block_3sp)
> str(block_3sp)
```

```
'data.frame':        200 obs. of  10 variables:
 $ time : int  1 2 3 4 5 6 7 8 9 10 ...
 $ x_t  : num  -0.742 1.245 -1.918 -0.962 1.332 ...
 $ x_t-1: num  NA -0.742 1.245 -1.918 -0.962 ...
 $ x_t-2: num  NA NA -0.742 1.245 -1.918 ...
 $ y_t  : num  -1.268 1.489 -0.113 -1.107 2.385 ...
 $ y_t-1: num  NA -1.268 1.489 -0.113 -1.107 ...
 $ y_t-2: num  NA NA -1.268 1.489 -0.113 ...
 $ z_t  : num  -1.864 -0.482 1.535 -1.493 -1.119 ...
 $ z_t-1: num  NA -1.864 -0.482 1.535 -1.493 ...
 $ z_t-2: num  NA NA -1.864 -0.482 1.535 ...
```

In order to correctly index into columns, `block_lnlp` has an option to indicate that the first column is actually a time index. When `first_column_time` is set to `TRUE`, a value of `1` for `target_column` will refer to the first *data* column in the data.frame, and skip the "time" column (the parameter `columns` is similarly indexed). If column names are used, then the `first_column_time` parameter is ignored.

```
> lib <- c(1, NROW(block_3sp))
> pred <- c(1, NROW(block_3sp))
> columns <- c(1, 2, 4) # c("x_t", "x_t-1", "y_t")
> target <- 1 # "x_t"
> block_lnlp_output <- block_lnlp(block_3sp, lib = lib, pred = pred,
+                                 columns = c(1, 2, 4), target_column = 1,
+                                 stats_only = FALSE, first_column_time = TRUE)
```

Note that the default value for the `tp` parameter is `1`, indicating that predictions will be made for 1 time step into the future (i.e. the subsequent row of the input data). In some cases, the data may already be processed such that the predictions are already aligned in the same row, but a different column, in which case the `tp` parameter should be set to `0`.

```
> str(block_lnlp_output)
```

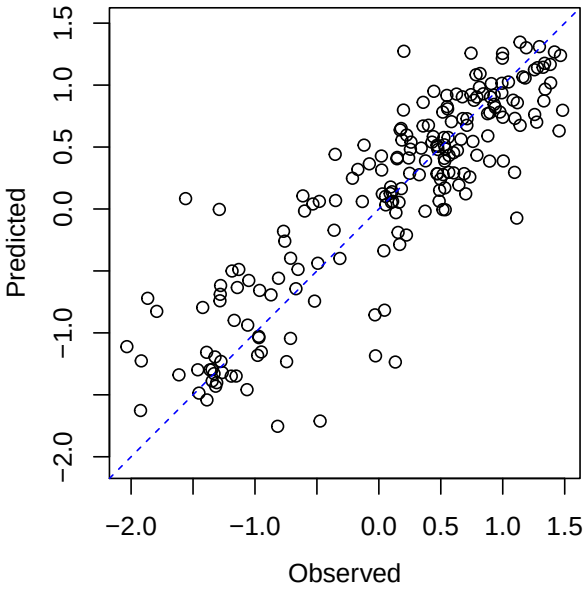
```
List of 1
 $ :List of 4
  ..$ params      :'data.frame':        1 obs. of  3 variables:
  .. ..$ embedding: int 1
  .. ..$ tp       : num 1
  .. ..$ nn       : num 4
  ..$ embedding   : chr "1, 2, 4"
```

```
..$ model_output:'data.frame':        200 obs. of  4 variables:
.. ..$ time    : num [1:200] 2 3 4 5 6 7 8 9 10 11 ...
.. ..$ obs     : num [1:200] 1.245 -1.918 -0.962 1.332 -0.817 ...
.. ..$ pred    : num [1:200] NaN -1.226 -0.657 0.872 -1.754 ...
.. ..$ pred_var: num [1:200] NaN 0.328 0.522 0.236 0.1 ...
..$ stats       :'data.frame':        1 obs. of  12 variables:
.. ..$ num_pred           : num 198
.. ..$ rho                : num 0.872
.. ..$ mae                : num 0.32
.. ..$ rmse               : num 0.434
.. ..$ perc               : num 0.889
.. ..$ p_val              : num 9.98e-79
.. ..$ const_pred_num_pred: num 198
.. ..$ const_pred_rho     : num -0.539
.. ..$ const_pred_mae     : num 1.31
.. ..$ const_pred_rmse    : num 1.55
.. ..$ const_pred_perc    : num 0.394
.. ..$ const_p_val        : num 1
```

By setting `stats_only` to `FALSE`, the output is a list with the full model output. Because only 1 model (1 combination of input parameters) was run, the output is a list with 1 element. This element is actually another list with 4 named elements: `params`, a data.frame of the input parameters; `embedding`, a vector of the columns used to construct the model; `model_output`, a data.frame of the full set of observed and predicted values; and `stats`, a data.frame of the forecast statistics.

To compare the observed and predicted values, we must therefore index correctly into the output. We can then plot the observed and predicted values and see how well the model did relative to the expected 1:1 line.

```
> observed <- block_lnlp_output[[1]]$model_output$obs
> predicted <- block_lnlp_output[[1]]$model_output$pred
> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0), pty = "s") # set up margins for plotting
> plot_range <- range(c(observed, predicted), na.rm = TRUE)
> plot(observed, predicted, xlim = plot_range, ylim = plot_range,
+      xlab = "Observed", ylab = "Predicted")
> abline(a = 0, b = 1, lty = 2, col = "blue")
```



2.5. Causality Inference and Cross Mapping

One of the corollaries to the Generalized Takens' Theorem is that it should be possible to cross predict or cross map between variables that are observed from the same system. Consider two variables, x and y that interact in a dynamic system. Then the univariate reconstructions based on either x or y alone should uniquely identify the system state and thus the corresponding value of the other variable.

In the case of unidirectional causality, e.g. x causes y, the causal variable (x) leaves a signature on the affected variable (y). Consequently, the reconstructed states based on y can be used to cross predict the values of x (because the reconstruction based on y must be complete, it must include information about the value of x). Note that this cross prediction is in the **opposite** direction of the causal effect. At the same time, cross prediction from x to y will fail, because the time series of x behaves independently of y, so a univariate reconstruction using only lags of x is necessarily incomplete.

Although x has incomplete information for predicting y, it does affect the values of y, and therefore will likely to have nonzero predictive skill. However, this cross mapping will be limited to the statistical association between x and y and will generally not improve as longer time series are used for reconstruction. In contrast, the cross prediction of x from y will generally improve. This convergence is therefore a crucial property for inferring causality. For practical reasons, the sensitivity of detecting causality this way is improved if, instead of predicting the future value of another variable, we estimate the concurrent value of another variable. We refer to this modified method as cross mapping, because we are not "predicting" the future.

For a more detailed description of using cross mapping to infer causation, see Sugihara *et al.* (2012).

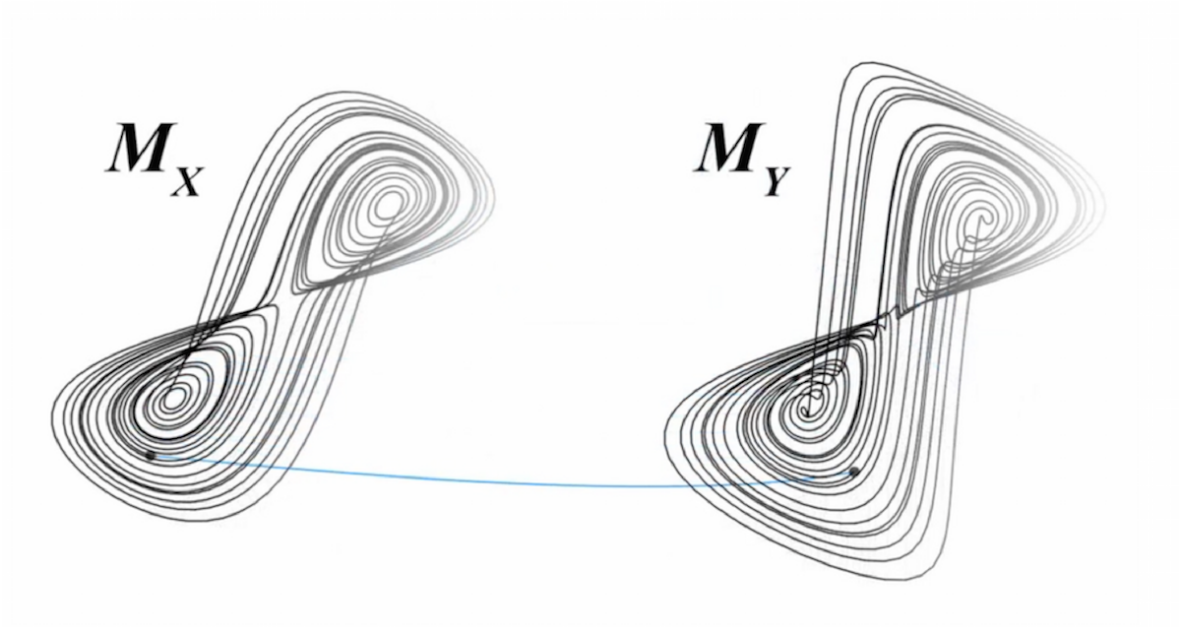


Figure 3: **cross mapping Between Univariate Reconstructions of the Lorenz Attractor**
Univariate reconstructions of the canonical Lorenz Attractor based on x and y (M_X and M_Y, respectively) map one-to-one to each other.

2.6. Convergent Cross Mapping (CCM)

In **rEDM**, convergent cross mapping is implemented as the `ccm` function, which provides an easy way to compute cross map skill for different subsamples of the time series. In the following example, we reproduce the analysis from Sugihara *et al.* (2012), identifying causality between anchovy landings in California and Newport Pier sea-surface temperature. For this example, a previously identified value of `3` for the embedding dimension will be used.

To quantify convergence, we compute the cross map skill over many random subsamples of the time series. The `lib_sizes` parameter specifies the "library" size of the subsamples that we use, and the `num_samples` parameter specifies the number of subsamples to be generated at each library size. The `random_libs` and `replace` parameters specify how the subsamples will be generated. Here, setting both to `TRUE` means that random sampling with replacement will be used.

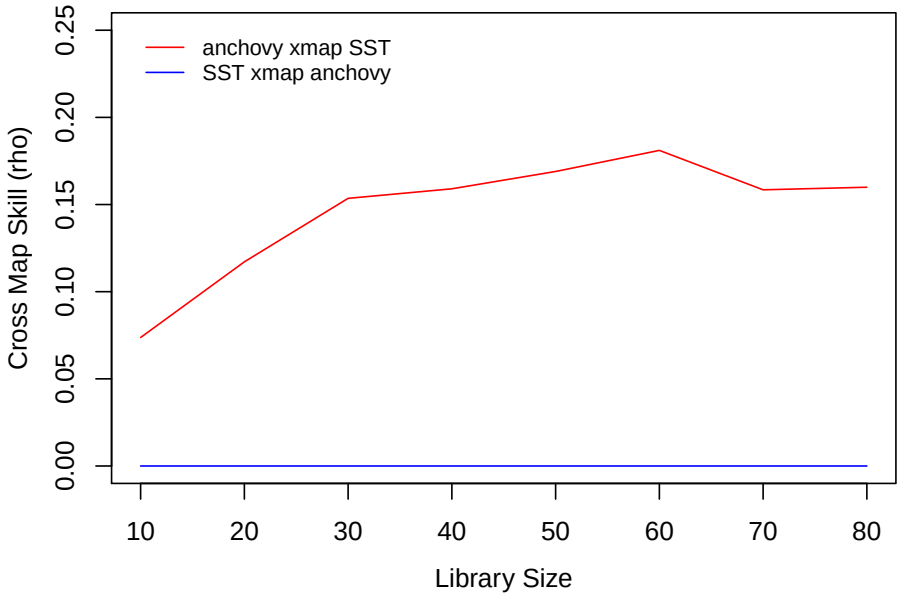
```
> data(sardine_anchovy_sst)
> anchovy_xmap_sst <- ccm(sardine_anchovy_sst, E = 3,
+                         lib_column = "anchovy", target_column = "np_sst",
+                         lib_sizes = seq(10, 80, by = 10), num_samples = 100,
+                         random_libs = TRUE, replace = TRUE)
> sst_xmap_anchovy <- ccm(sardine_anchovy_sst, E = 3,
+                         lib_column = "np_sst", target_column = "anchovy",
+                         lib_sizes = seq(10, 80, by = 10), num_samples = 100,
```

```
+                          random_libs = TRUE, replace = TRUE)
> str(anchovy_xmap_sst)
```

```
'data.frame':        800 obs. of  11 variables:
 $ E            : num  3 3 3 3 3 3 3 3 3 3 ...
 $ tau          : num  1 1 1 1 1 1 1 1 1 1 ...
 $ tp           : num  0 0 0 0 0 0 0 0 0 0 ...
 $ num_neighbors: num  4 4 4 4 4 4 4 4 4 4 ...
 $ lib_column   : Factor w/ 1 level "anchovy": 1 1 1 1 1 1 1 1 1 1 ...
 $ target_column: Factor w/ 1 level "np_sst": 1 1 1 1 1 1 1 1 1 1 ...
 $ lib_size     : num  10 10 10 10 10 10 10 10 10 10 ...
 $ num_pred     : num  76 76 76 76 76 76 76 76 76 76 ...
 $ rho          : num  0.0901 0.3498 0.1828 0.1921 -0.0182 ...
 $ mae          : num  0.984 0.798 0.829 1.023 0.928 ...
 $ rmse         : num  1.227 0.989 1.05 1.254 1.113 ...
```

The output from CCM is a data.frame with statistics for each model run (in thise case, 100 models at each of 8 library sizes). To interpret the results, we aggregate the cross map performance at each library size using the `ccm_means` function, which computes a mean value for the forecast statistics. Because average cross map skill less than 0 means there is no prediction skill, (predictions should not be anticorrelated with observations), we filter out negative values when plotting.

```
> a_xmap_t_means <- ccm_means(anchovy_xmap_sst)
> t_xmap_a_means <- ccm_means(sst_xmap_anchovy)
> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0)) # set up margins for plotting
> y1 <- pmax(0, a_xmap_t_means$rho)
> y2 <- pmax(0, t_xmap_a_means$rho)
> plot(a_xmap_t_means$lib_size, y1, type = "l", col = "red",
+      xlab = "Library Size", ylab = "Cross Map Skill (rho)", ylim = c(0, 0.25))
> lines(t_xmap_a_means$lib_size, y2, col = "blue")
> legend(x = "topleft", legend = c("anchovy xmap SST", "SST xmap anchovy"),
+        col = c("red", "blue"), lwd = 1, bty = "n", inset = 0.02, cex = 0.8)
```



3. Real Data Examples

3.1. Community Productivity and Invasibility

The data presented here are part of Experiment 120 (E120), the "Big Biodiversity" experiment at Cedar Creek LTER (Tilman, Knops, Wedin, Reich, Ritchie, and Siemann 1997), and the full data and metadata relating to the experiments that we discuss are available at `https://www.cedarcreek.umn.edu/research/data`. This experiment is the longest running randomized test for the effects of plant diversity on ecosystem functions. Plots were established in 1994 and planted with 1, 2, 4, 8, or 16 species, and have since then been sampled annually for above-ground plant biomass. The most well-known result from the experiment is that the number of planted species has strong positive influences on above-ground biomass production. However, because the diversity treatments are fixed (i.e. not allowed to vary dynamically), they do not lend themselves to EDM.

Instead, we focus on a different set of published results from the experiment: interactions between primary productivity, soil nitrate, and invasion rates. These show that increased biomass is associated with decreases in soil nitrate levels and decreases in invasion success (Fargione and Tilman 2005). A posited mechanism for this is soil nitrate: increased primary productivity leads to decreased soil nitrate, which in turn reduces resources available to invaders. In order to increase sample size for these analyses, we combine data from the treatments with 4-8 planted species, which follow similar dynamics, and analyze the different plots together as spatial replicates (Hsieh, Anderson, and Sugihara 2008).

The columns in the dataset `e120_biodiversity` are as follows: `Exp` indicates the experiment code, `Year` and `Month` are the sampling time, while `Plot`, `Field`, and `FieldPlot` describe plot identity. `NumSp` and `SpNum` show the planted and realized species diversity, respectively. `AbvBioAnnProd` shows annual aboveground productivity of planted species, in g/m^2. `noh020tot` shows soil nitrate levels in the top 20 cm of soil, measured in $\mu g/kg$ soil.

`invrichness` shows species richness of unplanted species in the plot. `SummerPrecip.mm.` shows precipitation annual from May to August measured in mm.

3.2. Preparing the Data

E120 includes data from multiple plots, meaning that we first need to collapse it into a single composite time series. We begin by normalizing each time series to mean zero and standard deviation one. This facilitates mixing multiple spatial replicates in a single analyses in EDM.

```
> data(e120_biodiversity)
> normalize <- function(x, ...) {(x - mean(x, ...))/sd(x, ...)}
> # separate time column from data
> vars <- c("AbvBioAnnProd", "noh020tot", "invrichness", "SummerPrecip.mm.")
> composite_ts <- e120_biodiversity[, vars]
> # normalize each time series within a plot
> data_by_plot <- split(composite_ts, e120_biodiversity$Plot)
> normalized_data <- lapply(data_by_plot, function(df) sapply(df, normalize))
> composite_ts <- cbind(Year = e120_biodiversity$Year,
+                       data.frame(do.call(rbind, normalized_data)))
```

To prevent lagged vectors from being constructed that span separate plots, we need to create an appropriate index variable that identifies different segments.

```
> # Make composite library
> segments_end <- cumsum(sapply(data_by_plot, NROW))
> segments_begin <- c(1, segments_end[-length(segments_end)] + 1)
> segments <- cbind(segments_begin, segments_end)
> # Choose random segments for prediction
> set.seed(2312)
> rndlib <- sample(1:NROW(segments), floor(NROW(segments) * 0.75))
> composite_lib <- segments[rndlib, ]
> composite_pred <- segments[-rndlib, ]
```

Because the time series for precipitation does not vary among replicates, we also need to construct separate variables for analyzing precipitation dynamics:

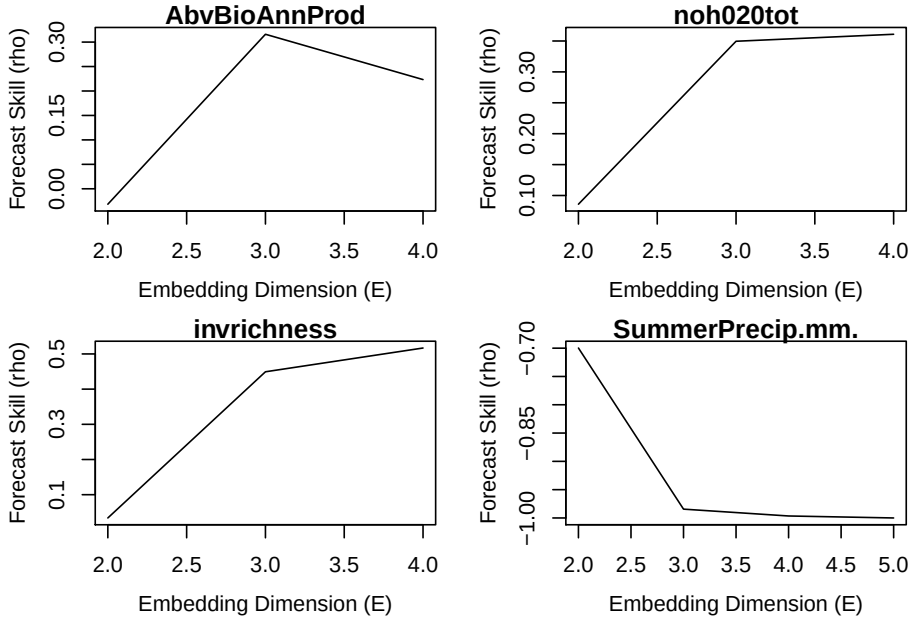
```
> precip_ts <- unique(e120_biodiversity[, c("Year", "SummerPrecip.mm.")])
> precip_ts <- precip_ts[order(precip_ts$Year), ]
```

Applying Simplex and S-map Algorithms

We can then use the **rEDM** functions as normal for each of our time series. First, we apply simplex projection:

```
> simplex_out <- lapply(names(composite_ts)[2:4], function(var) {
+     simplex(composite_ts[, c("Year", var)], E = 2:4,
```

```
+             lib = composite_lib, pred = composite_pred)
+ })
> simplex_out[[length(simplex_out)+1]] <- simplex(precip_ts, E = 2:5)
> names(simplex_out) <- names(composite_ts)[-1]
> par(mar = c(4, 4, 1, 1), mfrow = c(2, 2), mgp = c(2.5, 1, 0)) # set up margins for plott
> out<-lapply(names(simplex_out), function(var) {
+     plot(simplex_out[[var]]$E, simplex_out[[var]]$rho, type = "l",
+          xlab = "Embedding Dimension (E)", ylab = "Forecast Skill (rho)",
+          main = var)
+ })
```



These results give us the best embedding dimension for each of our projections:

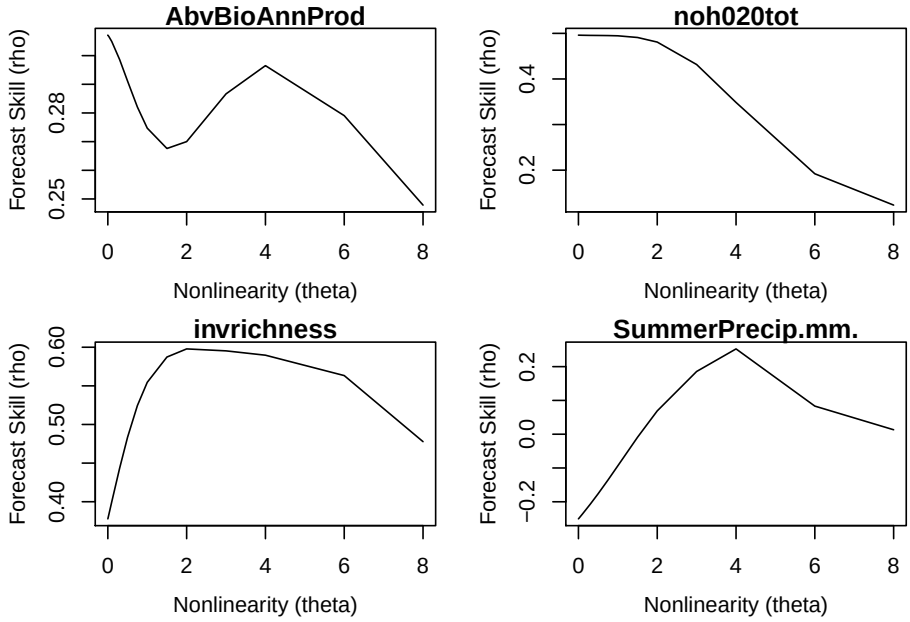
```
> best_E <- sapply(simplex_out, function(df) {df$E[which.max(df$rho)]})
> best_E

   AbvBioAnnProd        noh020tot      invrichness SummerPrecip.mm.
               3                4                4                2
```

Note that for two variables (nitrogen and invasive richness), the best embedding dimension is also the maximum that we test. These results suggest that the dynamics might be high-dimensional, and that collecting longer time series might enable better predictions.

Using these embedding dimensions, we can now apply S-maps to identify nonlinearity:

```
> smap_out <- lapply(names(composite_ts)[2:4], function(var) {
+     s_map(composite_ts[, c("Year", var)], E = best_E[var],
+             lib = composite_lib, pred = composite_pred)
+ })
> smap_out[[length(smap_out)+1]] <- s_map(precip_ts, E = best_E[length(smap_out)+1])
```

```
> names(smap_out) <- names(simplex_out)
> par(mar = c(4, 4, 1, 1), mfrow = c(2, 2), mgp = c(2.5, 1, 0)) # set up margins for plott
> lapply(names(smap_out), function(var) {
+     plot(smap_out[[var]]$theta, smap_out[[var]]$rho, type = "l",
+          xlab = "Nonlinearity (theta)", ylab = "Forecast Skill (rho)",
+          main = var)
+ })
```



Note that species richness, and precipitation time series suggest nonlinear dynamics in the data (because of the initial increase in forecast skill for non-zero `theta`, followed by a sharp drop-off). However, the strict fall-off in `rho` for non-zero `theta` in the case of `noh020tot` and `AbvBioAnnProd` suggest that soil nitrate levels and total biomass may have only linear dynamics,

Multivariate Models

Out next step is to use information from several different variables to make better forecasts. We can do this using `block_lnlp`, but first need to manually constructed lagged vectors for each variable. This requires a bit of care in coding, as we need to ensure that lagged components come only from observations within a single field and transect.

```
> data_by_plot <- split(composite_ts, e120_biodiversity$Plot)
> block_data <- do.call(rbind, lapply(data_by_plot, function(df) {
+     n <- NROW(df)
+     temp <- data.frame(Year = df$Year)
+     temp$AB_tm <- df$AbvBioAnnProd
+     temp$AB_tm1 <- c(NA, temp$AB_tm[-n])
+     temp$AB_tm2 <- c(NA, temp$AB_tm1[-n])
+     temp$AB_tm3 <- c(NA, temp$AB_tm2[-n])
```

```
+
+     temp$NO_tm <- df$noh020tot
+     temp$NO_tm1 <- c(NA, temp$NO_tm[-n])
+     temp$NO_tm2 <- c(NA, temp$NO_tm1[-n])
+     temp$NO_tm3 <- c(NA, temp$NO_tm2[-n])
+
+     temp$IV_tm <- df$invrichness
+     temp$IV_tm1 <- c(NA, temp$IV_tm[-n])
+     temp$IV_tm2 <- c(NA, temp$IV_tm1[-n])
+     temp$IV_tm3 <- c(NA, temp$IV_tm2[-n])
+
+     temp$PR_tm <- df$SummerPrecip.mm
+     temp$PR_tm1 <- c(NA, temp$PR_tm[-n])
+     temp$PR_tm2 <- c(NA, temp$PR_tm1[-n])
+     temp$PR_tm3 <- c(NA, temp$PR_tm2[-n])
+
+     return(temp)
+ }))
> head(block_data[, 1:5], 20)

     Year      AB_tm     AB_tm1     AB_tm2     AB_tm3
3.1  1996 -1.0626351         NA         NA         NA
3.2  1997 -0.5456990 -1.0626351         NA         NA
3.3  1998  1.6338642 -0.5456990 -1.0626351         NA
3.4  1999 -0.4869863  1.6338642 -0.5456990 -1.0626351
3.5  2001 -0.2983658 -0.4869863  1.6338642 -0.5456990
3.6  2002  0.7598219 -0.2983658 -0.4869863  1.6338642
12.1 1996 -1.0139507         NA         NA         NA
12.2 1997 -0.9855735 -1.0139507         NA         NA
12.3 1998  1.4960850 -0.9855735 -1.0139507         NA
12.4 1999  0.3898716  1.4960850 -0.9855735 -1.0139507
12.5 2001 -0.4926840  0.3898716  1.4960850 -0.9855735
12.6 2002  0.6062517 -0.4926840  0.3898716  1.4960850
15.1 1996 -1.4989147         NA         NA         NA
15.2 1997 -0.6867934 -1.4989147         NA         NA
15.3 1998  1.5499186 -0.6867934 -1.4989147         NA
15.4 1999  0.3648442  1.5499186 -0.6867934 -1.4989147
15.5 2000  0.6089230  0.3648442  1.5499186 -0.6867934
15.6 2001 -0.5679305  0.6089230  0.3648442  1.5499186
15.7 2002  0.2299527 -0.5679305  0.6089230  0.3648442
22.1 1996 -1.3230421         NA         NA         NA
```

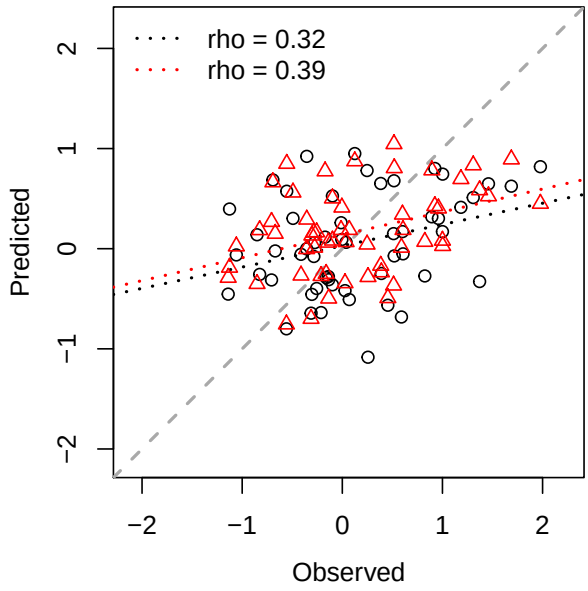
Now, we can run the `block_lnlp` on the composite, multivariate time series. First, we run the algorithm to predict primary productivity dynamics, based on its own lagged dynamics. Next, we add additional information about precipitation:

```
> AB_columns <- c("AB_tm", "AB_tm1", "AB_tm2")
> AB_output <- block_lnlp(block_data, lib = composite_lib, pred = composite_pred,
```

```
+                          columns = AB_columns, target_column = 1,
+                          stats_only = FALSE, first_column_time = TRUE)
> ABNO_columns <- c("AB_tm", "AB_tm1", "AB_tm2", "NO_tm", "NO_tm1", "NO_tm2")
> ABNO_output <- block_lnlp(block_data, lib = composite_lib, pred = composite_pred,
+                           columns = ABNO_columns, target_column = 1,
+                           stats_only = FALSE, first_column_time = TRUE)
```

Note that each additional variable slightly improves the predictive skill of the model.

```
> observed_AB <- AB_output[[1]]$model_output$obs
> predicted_AB <- AB_output[[1]]$model_output$pred
> observed_ABNO <- ABNO_output[[1]]$model_output$obs
> predicted_ABNO <- ABNO_output[[1]]$model_output$pred
> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0), pty = "s") # set up margins for plotting
> plot_range <- range(c(observed_AB, predicted_AB), na.rm = TRUE)
> plot(observed_AB, predicted_AB, xlim = plot_range, ylim = plot_range,
+      xlab = "Observed", ylab = "Predicted")
> abline(a = 0, b = 1, lty = 2, col = "darkgrey", lwd = 2)
> abline(lm(predicted_AB ~ observed_AB), col = "black", lty = 3, lwd = 2)
> points(observed_ABNO, predicted_ABNO, pch = 2, col = "red")
> abline(lm(predicted_ABNO ~ observed_ABNO), col = "red", lty = 3, lwd = 2)
> legend("topleft", legend = c(paste("rho =", round(AB_output[[1]]$stats$rho, 2)),
+                              paste("rho =", round(ABNO_output[[1]]$stats$rho, 2))),
+        lty = 3, lwd = 2, col = c("black", "red"), bty = "n")
```

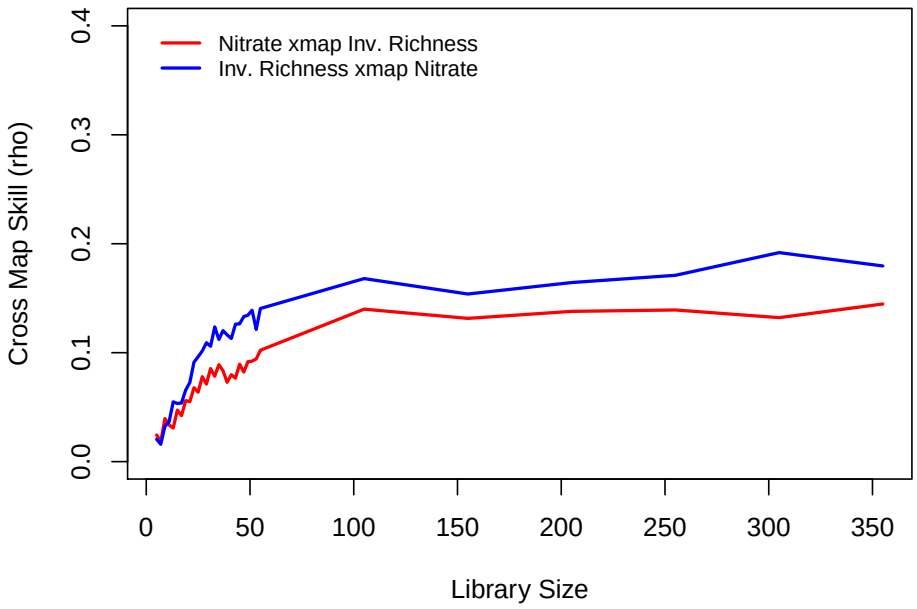


Convergent Cross Mapping

Finally, we can apply CCM to our data to test for pairwise causal links among variables. For each individual test, we use the embedding dimension corresponding to the "best" embedding

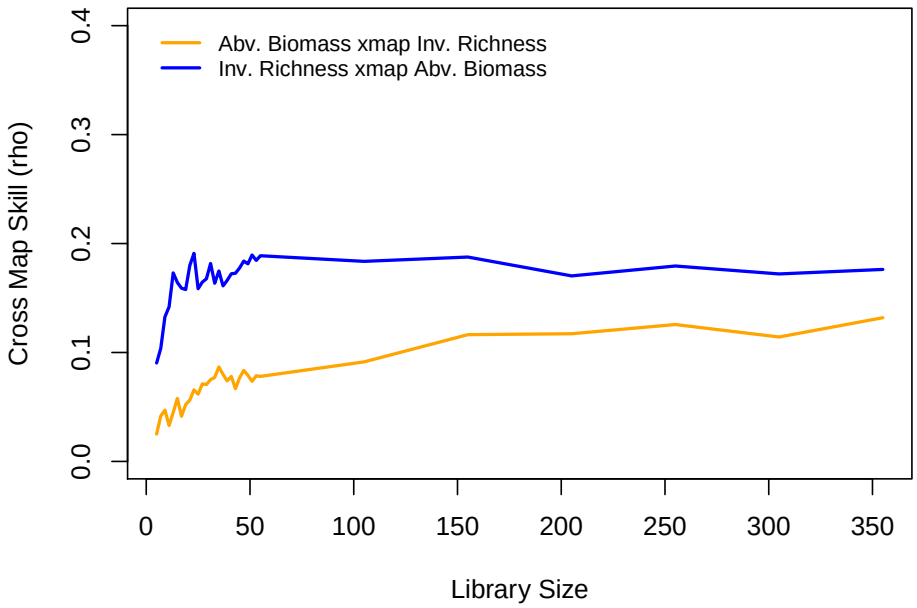
dimension for the variable that we are trying to predict (i.e. the putative causal process).

```
> my_lib_sizes <- c(seq(5, 55, by = 2), seq(55, 400, by = 50))
> no_xmap_inv <- ccm(composite_ts, lib = segments, pred = segments,
+                    lib_column = "noh020tot", target_column = "invrichness",
+                    E = best_E["noh020tot"], lib_sizes = my_lib_sizes, silent = TRUE)
> inv_xmap_no <- ccm(composite_ts, lib = segments, pred = segments,
+                    lib_column = "invrichness", target_column = "noh020tot",
+                    E = best_E["invrichness"], lib_sizes = my_lib_sizes, silent = TRUE)
> n_xmap_i_means <- ccm_means(no_xmap_inv)
> i_xmap_n_means <- ccm_means(inv_xmap_no)
> par(mar = c(4, 4, 1, 1)) # set up margins for plotting
> plot(n_xmap_i_means$lib_size, pmax(0, n_xmap_i_means$rho), type = "l",
+      xlab = "Library Size", ylab = "Cross Map Skill (rho)",
+      col = "red", ylim = c(0, 0.4), lwd = 2)
> lines(i_xmap_n_means$lib_size, pmax(0, i_xmap_n_means$rho),
+       col = "blue", lwd = 2)
> legend(x = "topleft", col = c("red", "blue"), lwd = 2,
+        legend = c("Nitrate xmap Inv. Richness", "Inv. Richness xmap Nitrate"),
+        inset = 0.02, bty = "n", cex = 0.8)
```




```
> ab_xmap_inv <- ccm(composite_ts, lib = segments, pred = segments,
+                    lib_column = "AbvBioAnnProd", target_column = "invrichness",
+                    E = best_E["AbvBioAnnProd"], lib_sizes = my_lib_sizes, silent = TRUE)
> inv_xmap_ab <- ccm(composite_ts, lib = segments, pred = segments,
+                    lib_column = "invrichness", target_column = "AbvBioAnnProd",
+                    E = best_E["invrichness"], lib_sizes = my_lib_sizes, silent = TRUE)
> a_xmap_i_means <- ccm_means(ab_xmap_inv)
```

```
> i_xmap_a_means <- ccm_means(inv_xmap_ab)
> par(mar = c(4, 4, 1, 1)) # set up margins for plotting
> plot(a_xmap_i_means$lib_size, pmax(0, a_xmap_i_means$rho), type = "l",
+      xlab = "Library Size", ylab = "Cross Map Skill (rho)",
+      col = "orange", ylim = c(0, 0.4), lwd = 2)
> lines(i_xmap_a_means$lib_size, pmax(0, i_xmap_a_means$rho),
+       col = "blue", lwd = 2)
> legend(x = "topleft", col = c("orange", "blue"),
+        legend = c("Abv. Biomass xmap Inv. Richness",
+                   "Inv. Richness xmap Abv. Biomass"),
+        lwd = 2, inset = 0.02, bty = "n", cex = 0.8)
```



In each case, results suggest that invasive richness is driven by other variables more strongly than it influences them in return. For soil nitrate, this makes sense, as invading species in these plots are quickly weeded out and should not have much time to influence local environmental conditions, whereas the establishment success of invading species should depend strongly on local nitrogen availability.

Our analyses for biomass and invasion may indicate either bidirectional forcing, or synchrony. Though there is a stronger rise in ρ with library length for "Inv. Richness xmap Abv. Biomass" (i.e., "Biomass affects Invasive Richness"), there also does not appear to be a significant difference between ρ at short library lengths and ρ at large library lengths. Nevertheless, bidirectional causal forcing, with stronger effects of planted biomass on invasion than vice versa, would make sense, as weeding treatments should insure that there should only be moderate effects of invading species on plot-level biomass (e.g. by decreasing biomass of planted species through light competition), while effects of plot-level planted biomass on invader success should be much stronger (e.g. through competition for space or soil resources).

3.3. Apple-Blossom Thrips

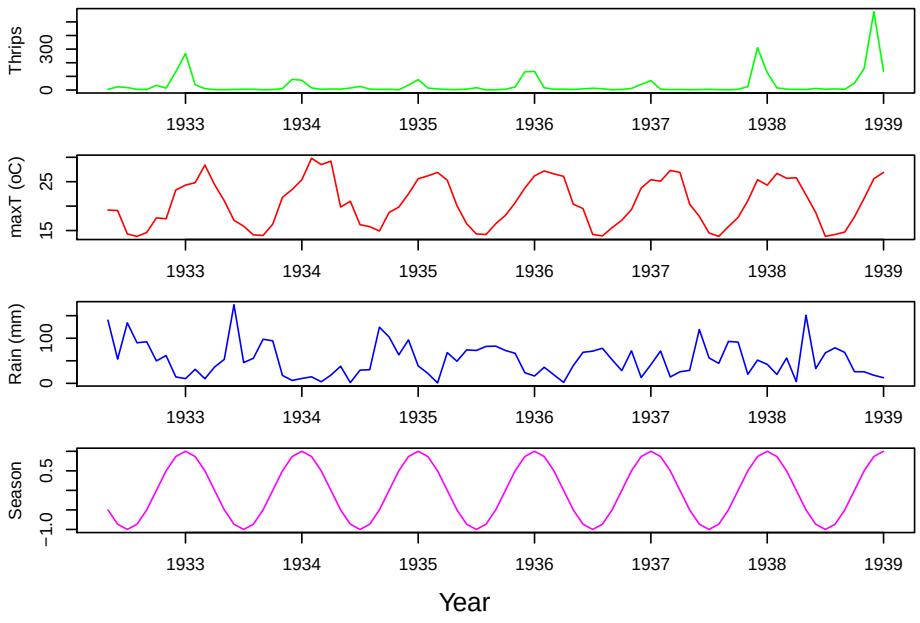
In this next example, we use EDM to re-examine the classic apple-blossom thrips (*Thrips imaginis*) time series from the Waite Institute in Australia (Davidson and Andrewartha 1948a,b). Seasonal outbreaks of *Thrips imaginis* were observed to vary greatly in magnitude from year to year, but large outbreaks tended to coincide across large spatial domains. This led to the hypothesis that regional-scale climatic factors are responsible for controlling the size of the seasonal outbreaks (what might now be called the Moran effect (Moran 1953)).

```
> data(thrips_block)
> colnames(thrips_block)

[1] "Year"           "Month"          "Thrips_imaginis" "maxT_degC"
[5] "Rain_mm"        "Season"
```

The first data column `Thrips_imaginis` contains counts of *Thrips imaginis* obtained from the Global Population Dynamics Database (GPDD) (NERC Centre for Population Biology 2010). `maxT_degC` is the mean maximum daily temperature (°C) taken over each month and `Rain_mm` is the monthly rainfall (mm), both from the Waite Institute. The final column `Season` is a simple annual sinusoid that peaks in December (the Austral summer) and acts as an indicator of season.

First, we plot the data.

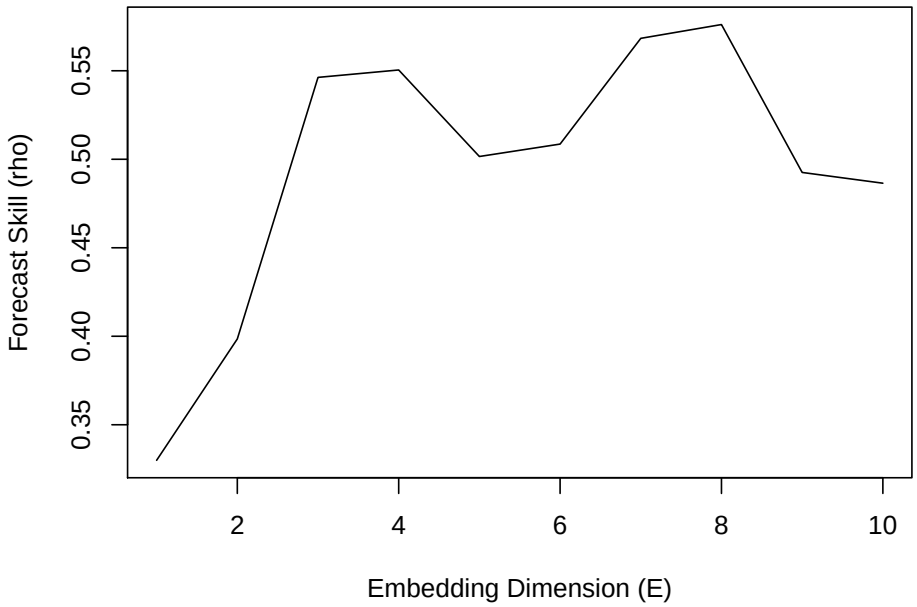
```
> par(mar = c(2, 4, 1, 1), oma = c(2, 0, 0, 0), mfrow = c(4, 1), mgp = c(2.5, 1, 0)) # set
> time_dec <- thrips_block$Year + (thrips_block$Month) / 12
> plot(time_dec, thrips_block$Thrips_imaginis, type = "l", col = "green", ylab = "Thrips",
> plot(time_dec, thrips_block$maxT_degC, type = "l", col = "red", ylab = "maxT (oC)", xlab
> plot(time_dec, thrips_block$Rain_mm, type = "l", col = "blue", ylab = "Rain (mm)", xlab
> plot(time_dec, thrips_block$Season, type = "l", col = "magenta", ylab = "Season")
> mtext("Year", side = 1, outer = TRUE, line = 1)
```



Note that all the time-series variables, particularly the mean maximum daily temperature, show marked seasonality.

Univariate Analysis

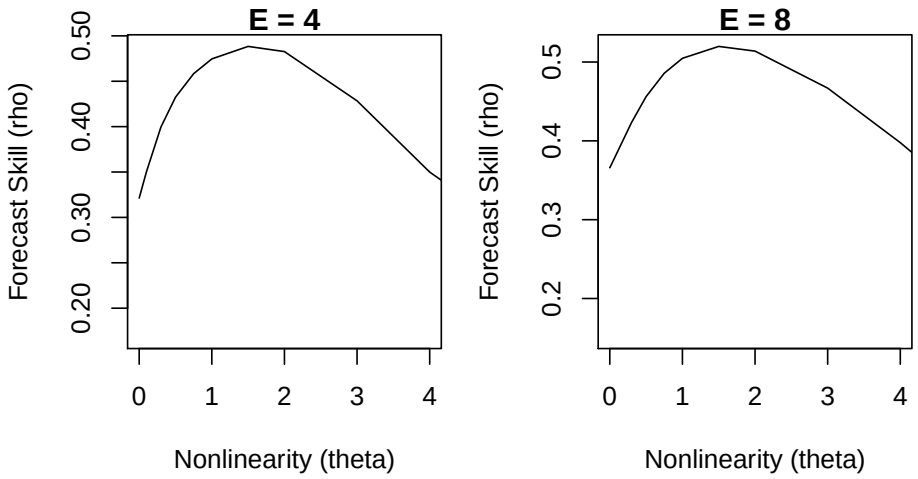
```
> ts <- thrips_block$Thrips_imaginis
> lib <- c(1, length(ts))
> pred <- c(1, length(ts))
> simplex_output <- simplex(ts, lib, pred, tau = 1)

> par(mar = c(4, 4, 1, 1)) # set up margins for plotting
> plot(simplex_output$E, simplex_output$rho, type = "l",
+      xlab = "Embedding Dimension (E)", ylab = "Forecast Skill (rho)")
```



While there is an initial peak in forecast skill for simplex projection at `E = 4`, the global maximum is at `E = 8`. This suggests that both `E = 4` and `E = 8` are practical embedding dimensions.

To test for nonlinearity, we examine both `E = 4` and `E = 8` to verify that the S-map result is robust to the choice of embedding dimension.

```
> smap_output <- list()
> smap_output[[1]] <- s_map(ts, lib, pred, E = 4)
> smap_output[[2]] <- s_map(ts, lib, pred, E = 8)

> par(mar = c(4, 4, 1, 1), mfrow = c(1, 2)) # set up margins for plotting
> plot(smap_output[[1]]$theta, smap_output[[1]]$rho, type = "l", xlim = c(0, 4),
+      xlab = "Nonlinearity (theta)", ylab = "Forecast Skill (rho)", main = "E = 4")
> plot(smap_output[[2]]$theta, smap_output[[2]]$rho, type = "l", xlim = c(0, 4),
+      xlab = "Nonlinearity (theta)", ylab = "Forecast Skill (rho)", main = "E = 8")
```



The S-map results demonstrate clear nonlinearity in the *Thrips* time series, as nonlinear models ($\theta > 0$) produce better predictions than the linear model ($\theta = 0$). This suggests that, despite the strong seasonal dynamics, abundances of *Thrips* do not simply track the environment passively, but have intrinsic dynamics that can be recovered using nonlinear methods. To look more closely at the issue of seasonal drivers, however, we turn to convergent cross mapping (CCM).

Seasonal Drivers

Recall that there is a two-part criterion for CCM to be a rigorous test of causality: (1) The cross map prediction skill is statistically significant when using the full time series as the library. (2) Cross map prediction demonstrates convergence, i.e. prediction skill increases as more of the time series is used for the library and the reconstructed attractor becomes more dense.

For an initial summary, we first compute the cross map skill (measured with Pearson's ρ) for each variable pair, using the full time series as the library.

```
> vars <- colnames(thrips_block[3:6])
> n <- NROW(thrips_block)
> ccm_matrix <- array(NA, dim = c(length(vars), length(vars)),
+                     dimnames = list(vars, vars))
> for(ccm_from in vars)
+ {
+     for(ccm_to in vars[vars != ccm_from])
+     {
+         out_temp <- ccm(thrips_block, E = 8,
+                         lib_column = ccm_from, target_column = ccm_to,
+                         lib_sizes = n, replace = FALSE, silent = TRUE)
+         ccm_matrix[ccm_from, ccm_to] <- out_temp$rho
+     }
+ }
```

For comparison we also compute the lagged cross-correlation, allowing lags of up to ± 6 months.

```
> corr_matrix <- array(NA, dim = c(length(vars), length(vars)),
+                      dimnames = list(vars, vars))
> for(ccm_from in vars)
+ {
+     for(ccm_to in vars[vars != ccm_from])
+     {
+         cf_temp <- ccf(thrips_block[,ccm_from], thrips_block[,ccm_to],
+                        type = "correlation", lag.max = 6, plot = FALSE)$acf
+         corr_matrix[ccm_from, ccm_to] <- max(abs(cf_temp))
+     }
+ }
```

We compare the two matrices.

```
> head(ccm_matrix) # Matrix of CCM prediction skill
```

```
                Thrips_imaginis maxT_degC   Rain_mm    Season
Thrips_imaginis              NA 0.9239205 0.5136489 0.9551902
maxT_degC             0.6046406        NA 0.4629704 0.9918832
Rain_mm               0.4277785 0.8210977        NA 0.7780148
Season                0.5619095 0.9625766 0.3944837        NA
```

```
> head(corr_matrix) # Matrix of Lagged Correlation
```

```
                Thrips_imaginis maxT_degC   Rain_mm    Season
Thrips_imaginis              NA 0.4489876 0.2668395 0.4488334
maxT_degC             0.4489876        NA 0.5949077 0.9452625
Rain_mm               0.2668395 0.5949077        NA 0.5332935
Season                0.4488334 0.9452625 0.5332935        NA
```

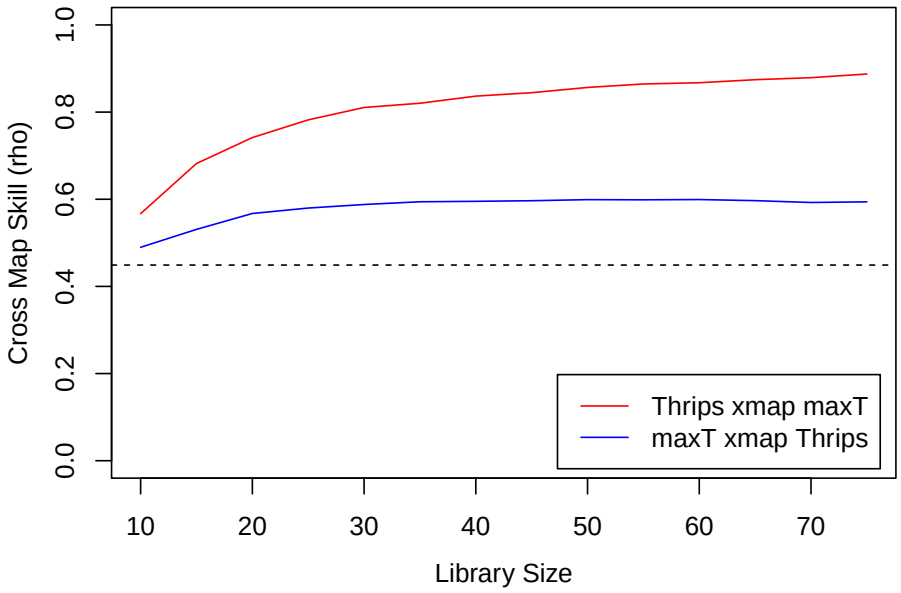
Notice that the lagged correlation between temperature and the seasonal indicator is extremely high. Thus, it is not surprising that the cross map results indicate that the seasonal variable can almost perfectly recover the temperature, $\rho = 0.95$. This makes interpretation more complicated, because we have to consider the possibility that cross mapping is simply identifying the shared seasonality between two time series. In other words, we may be able to cross map between temperature and any variable with a seasonal cycle, even if there is no actual causal mechanism.

Convergent Cross Mapping

With this in mind, we examine convergence in cross map predictability, i.e. we compute ρ as a function of library size `L`. We first look at *Thrips* and temperature.

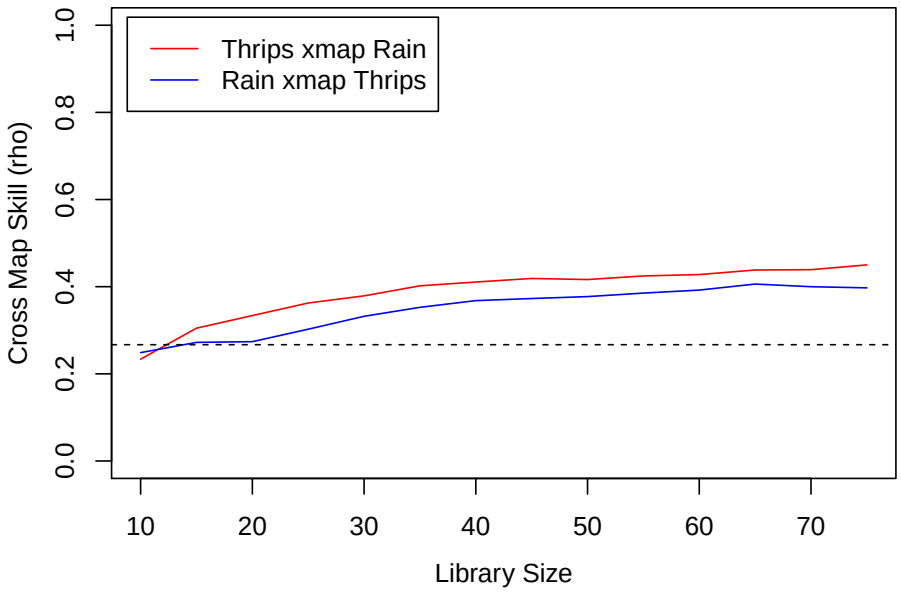
```
> thrips_xmap_maxT <- ccm(thrips_block, E = 8, random_libs = TRUE,
+                         lib_column = "Thrips_imaginis", target_column = "maxT_degC",
+                         lib_sizes = seq(10, 75, by = 5), num_samples = 300)
> maxT_xmap_thrips <- ccm(thrips_block, E = 8, random_libs = TRUE,
```

```
+                         lib_column = "maxT_degC", target_column = "Thrips_imaginis",
+                         lib_sizes = seq(10, 75, by = 5), num_samples = 300)
> ccm_out <- list(ccm_means(thrips_xmap_maxT), ccm_means(maxT_xmap_thrips))

> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0)) # set up margins for plotting
> plot(ccm_out[[1]]$lib_size, pmax(0, ccm_out[[1]]$rho), type = "l", col = "red",
+      xlab = "Library Size", ylab = "Cross Map Skill (rho)", ylim = c(0, 1))
> lines(ccm_out[[2]]$lib_size, pmax(0, ccm_out[[2]]$rho), col = "blue")
> abline(h = corr_matrix['Thrips_imaginis', 'maxT_degC'], col = "black", lty = 2)
> legend(x = "bottomright", legend = c("Thrips xmap maxT", "maxT xmap Thrips"),
+        col = c("red", "blue"), lwd = 1, inset = 0.02)
```



The magnitude of the cross-correlation between *Thrips* and temperature is shown as a black dashed line for comparison. We repeat the analysis for rainfall.

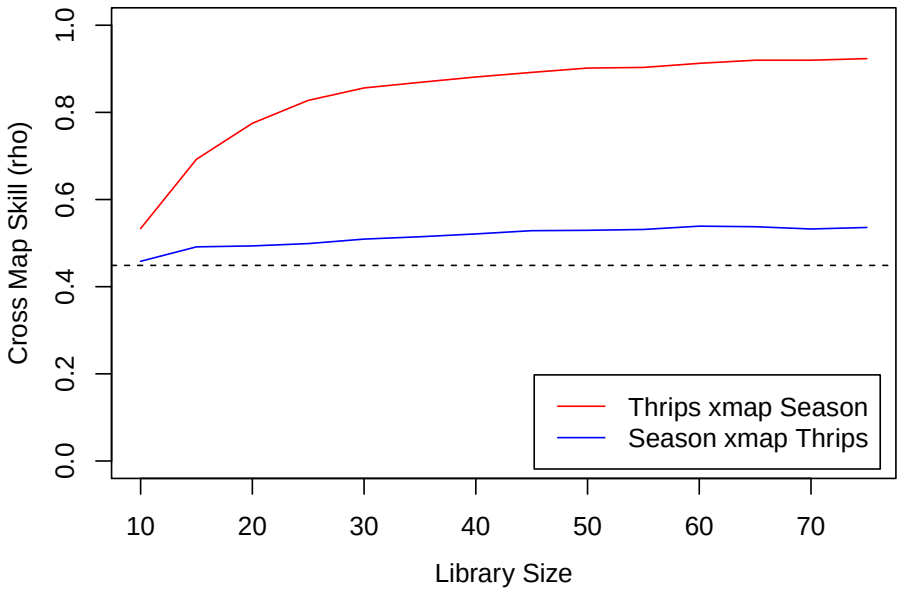
```
> thrips_xmap_Rain <- ccm(thrips_block, E = 8, random_libs = TRUE,
+                         lib_column = "Thrips_imaginis", target_column = "Rain_mm",
+                         lib_sizes = seq(10, 75, by = 5), num_samples = 300)
> Rain_xmap_thrips <- ccm(thrips_block, E = 8, random_libs = TRUE,
+                         lib_column = "Rain_mm", target_column = "Thrips_imaginis",
+                         lib_sizes = seq(10, 75, by = 5), num_samples = 300)
> ccm_out <- list(ccm_means(thrips_xmap_Rain), ccm_means(Rain_xmap_thrips))

> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0)) # set up margins for plotting
> plot(ccm_out[[1]]$lib_size, pmax(0, ccm_out[[1]]$rho), type = "l", col = "red",
+      xlab = "Library Size", ylab = "Cross Map Skill (rho)", ylim = c(0, 1))
> lines(ccm_out[[2]]$lib_size, pmax(0, ccm_out[[2]]$rho), col = "blue")
> abline(h = corr_matrix['Thrips_imaginis','Rain_mm'], col = 'black', lty = 2)
> legend(x = "topleft", legend = c("Thrips xmap Rain", "Rain xmap Thrips"),
+  col = c("red", "blue"), lwd = 1, inset = 0.02)
```



And finally for the deterministic seasonal cycle.

```
> thrips_xmap_Season <- ccm(thrips_block, E = 8, random_libs = TRUE,
+                           lib_column = "Thrips_imaginis", target_column = "Season",
+                           lib_sizes = seq(10, 75, by = 5), num_samples = 300)
> Season_xmap_thrips <- ccm(thrips_block, E = 8, random_libs = TRUE,
+                           lib_column = "Season", target_column = "Thrips_imaginis",
+                           lib_sizes = seq(10, 75, by = 5), num_samples = 300)
> ccm_out <- list(ccm_means(thrips_xmap_Season), ccm_means(Season_xmap_thrips))
```

```
> par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0)) # set up margins for plotting
> plot(ccm_out[[1]]$lib_size, pmax(0, ccm_out[[1]]$rho), type = "l", col = "red",
+      xlab = "Library Size", ylab = "Cross Map Skill (rho)", ylim = c(0, 1))
> lines(ccm_out[[2]]$lib_size, pmax(0, ccm_out[[2]]$rho), col = "blue")
> abline(h = corr_matrix['Thrips_imaginis', 'Season'], col = 'black', lty = 2)
> legend(x = "bottomright", col = c("red", "blue"), lwd = 1, inset = 0.02,
+        legend = c("Thrips xmap Season", "Season xmap Thrips"))
```



Importantly, the results show clear evidence of convergence for `Thrips` cross mapping the climactic variables, with the ρ at maximum `L` greatly exceeding the simple linear correlation.

However, we are still left with the conundrum that temperature and to a lesser extent, rainfall, are easily predicted from the seasonal cycle, and so we cannot immediately ignore the possibility that the cross map results are an artifact of shared seasonal forcing.

To reframe, we wish to reject the null hypothesis that the level of cross mapping we obtain for `maxT_degC` and `Rain_mm` can be solely explained by shared seasonality. This hypothesis is readily tested using randomization tests based on surrogate data. The idea here is to generate surrogate time series with the same level of shared seasonality. Cross mapping between the real time series and these surrogates thus generates a null distribution for ρ, against which the actual cross map ρ value can be compared.

Seasonal Surrogate Test

```
> num_surr <- 199
> surr_maxT <- make_surrogate_data(thrips_block$maxT_degC, method = "seasonal",
+                                  T_period = 12, num_surr = num_surr)
> surr_Rain <- make_surrogate_data(thrips_block$Rain_mm, method = "seasonal",
+                                  T_period = 12, num_surr = num_surr)
> rho_surr <- data.frame(maxT = numeric(num_surr), Rain = numeric(num_surr))
> xmap_func <- function(ts) {
+     ccm(cbind(thrips_block$Thrips_imaginis, ts), E = 8, lib_column = 1,
+         target_column = 2, lib_sizes = NROW(thrips_block), replace = FALSE)$rho
+ }
> rho_surr$maxT <- apply(surr_maxT, 2, xmap_func)
> rho_surr$Rain <- apply(surr_Rain, 2, xmap_func)
```

We now have a null distribution, and can easily estimate the p value for rejecting the null hypothesis of mutual seasonality.

```
> (sum(ccm_matrix['Thrips_imaginis','Rain_mm'] < rho_surr$Rain) + 1) /
+     (length(rho_surr$Rain) + 1)

[1] 0.07

> (sum(ccm_matrix['Thrips_imaginis','maxT_degC'] < rho_surr$maxT) + 1) /
+     (length(rho_surr$maxT) + 1)

[1] 0.155
```

In both cases, the cross map ρs we measure for the real time series are better than the median expectation under the null hypothesis. For rainfall, the effect is marginally significant, based on the common threshold of `p < 0.10`. Howover, the high correlation between the maximum daily temperature averaged over a month and the seasonal cycle makes it harder to establish significance for the effect of temperature. We note that the original *Thrips* data collections were at a much higher frequency than those available through the GPDD, and that `maxT_degC` shows much larger departures from the seasonal cycle on shorter time-scales. With more highly resolved data, it may well be possible to establish significance.

4. Technical Notes

4.1. Data Input

The **rEDM** functions are designed to accept data in common R data formats, namely vectors, matrices, and data.frames. Depending on the specific function, one or the other data type is preferred. Please see the documentation associated with individual functions for more details.

Missing data can be input using either `NA` or `NAN`. The program will automatically ignore such missing values as appropriate. For instance, simplex projection will not select nearest neighbors if any of the state vector coordinates is missing or if the corresponding target value is missing.

Note that when there is no observed target value, it is still possible to predict from a given state vector, if it has no missing values. Thus, it is possible to use the software to forecast ahead from an observed state into an unbserved future. This can be done simply by substituting `NA` or `NAN` for unknown future values. However, be aware that the performance metrics will be computed so as to ignore such predictions (since there are no observed values to compare against). Thus, the statistics (e.g., ρ, MAE, RMSE) may be computed based on fewer predictions than those actually made by the software.

4.2. General Parameters

Many of the functions in **rEDM** are designed around the same prediction engine, and so share many of the following parameters. Please see the documentation associated with individual functions to verify which parameters are applicable as well as the default values (which can change from function to function)

- `lib`: a 2-column matrix (or 2-element vector) where each row specifies the portions of the time series to use for attractor reconstruction (i.e., the set of vectors that can be selected as nearest neighbors).
  - e.g., `(1, n)` specifies that the first n *rows* (from 1 to n) of data are a contiguous time series block, each point of which can be used to construct state vectors
  - by default, lib uses the entire input as a single contiguous segment
- `pred`: the same format as `lib`, but specifes the portions of the time series to make predictions for
- `norm_type`: the distance metric to use for calculations
  - `"L2 norm"` (default) is the standard Euclidean distance, where the distance between a vector $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ and $\vec{y} = \langle y_1, y_2, \ldots, y_n \rangle$ is computed as $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^x + \cdots + (x_n - y_n)^2}$.
  - `"L1 norm"` is the Manhattan norm (also known as taxicab distance), where the distance between a vector $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ and $\vec{y} = \langle y_1, y_2, \ldots, y_n \rangle$ is computed as $|x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n|$.
  - `"P norm"` is the LP norm, using the `P` parameter as the exponent, where the distance between a vector $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ and $\vec{y} = \langle y_1, y_2, \ldots, y_n \rangle$ is computed as $\left((x_1 - y_1)^P + (x_2 - y_2)^P + \cdots + (x_n - y_n)^P\right)^{(1/P)}$
- `P`: the exponent for the `"P norm"`
- `E`: the embedding dimension to use for attractor reconstruction
- `tau`: the lag to use for attractor reconstruction; by default, tau is set to `1`
- `tp`: time to prediction (how many steps ahead to make forecasts).
- `num_neighbors`: the number of neighbors to use. Any of `"e+1"`, `"E+1"`, `"e + 1"`, and `"E + 1"` will all peg this parameter to be equivalent to the `E+1` for each run. If set to any value less than 1, all possible neighbors will be used.
- `theta`: the nonlinear tuning parameter (for use with S-maps) that adjusts how distance is factored into computation of the local linear map. `theta = 0` corresponds to a globally linear map, while values greater than 0 correspond to nonlinear models where the local linear map changes as a function of state-space.
- `stats_only`: specifies whether the output should just contain statistics of the predictions (`TRUE` by default), or also contain all the predictions that were made
- `exclusion_radius`: sets the threshold whereby all vectors with time indices too close to the predictee will be excluded from being considered nearest neighbors. For example, `exclusion_radius = 1` means that vectors must have an associated time index more than 1 away from potential nearest neighbors, By default, `exclusion_radius = NULL`, turning this filtering off.

- `epsilon`: sets the threshold whereby all vectors with distance too far away from the predictee will be excluded from being considered nearest neighbors. For example, `epsilon = 2` means that vectors must have be within a distance of 2 from potential nearest neighbors. By default, `epsilon = NULL`, turning this filtering off.

- `silent`: specifies whether to suppress warning messages from being printed to the R console (`FALSE` by default).

- `save_smap_coefficients`: specifies whether to include a table of s-map coefficients with the output (`FALSE` by default). Note that setting `save_smap_coefficients = TRUE` forces the full output and will override `stats_only = TRUE`.

5. Acknowledgements

This work is supported by DoD-Strategic Environment Research and Development Program RC-2509, Lenfest Ocean Program #00028335, National Science Foundation Grant No. DEB-1020372, NSF-NOAA Comparative Analysis of Marine Ecosystem Organization (CAMEO) program Grant NA08OAR4320894/CAMEO, National Science Foundation Graduate Research Fellowships, Environmental Protection Agency Science to Achieve Results Fellowship, the Sugihara Family Trust, the Deutsche Bank-Jameson Complexity Studies Fund, and the McQuown Chair in Natural Science.

rEDM is the latest incarnation of EDM code. Past versions have include contributions from George Sugihara, Alan Trombla, Richard Penner, Victor Wong, Martin Casdagli, Mohsen Azarbayejani, Ava Pierce, Jennifer Trezzo, and Hao Ye.

References

Casdagli M, Eubank S, Farmer J, Gibson J (1991). "State space reconstruction in the presence of noise." *Physica D: Nonlinear Phenomena*, **51**(1-3), 52–98.

Davidson J, Andrewartha HG (1948a). "Annual trends in a natural population of *Thrips imaginis* (Thysanoptera)." *Journal of Animal Ecology*, **17**, 193–199.

Davidson J, Andrewartha HG (1948b). "The influence of rainfall, evaporation and atmospheric temperature on fluctuations in the size of a natural population of *Thrips imaginis* (Thysanoptera)." *Journal of Animal Ecology*, **17**, 200–222.

Deyle ER, Sugihara G (2011). "Generalized theorems for nonlinear state space reconstruction." *PLoS ONE*, **6**, e18295.

Dixon PA, Milicich M, Sugihara G (1999). "Episodic fluctuations in larval supply." *Science*, **283**, 1528–1530.

Fargione JE, Tilman D (2005). "Diversity decreases invasion via both sampling and complementarity effects." *Ecology Letters*, **8**, 604–611.

Fisher RA (1915). "Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population." *Biometrika*, **10**(4), 507–521.

Hsieh CH, Anderson C, Sugihara G (2008). "Extending nonlinear analysis to short Ecological time series." *The American Naturalist*, **171**(1), 71–80.

Lorenz EN (1963). "Deterministic nonperiodic flow." *Journal of the Atmospheric Sciences*, **20**(2), 130–141.

Moran PAP (1953). "The statistical analysis of the Canadian Lynx cycle II. synchronization and meteorology." *Australian Journal of Zoology*, pp. 291–298.

NERC Centre for Population Biology IC (2010). "The Global Population Dynamics Database Version 2."

Sauer T, Yorke JA, Casdagli M (1991). "Embedology." *Journal of Statistical Physics*, **65**(3-4), 579–616.

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 CH, Deyle E, Fogarty M, Munch S (2012). "Detecting causality in complex ecosystems." *Science*, **338**, 496–500.

Sugihara G, May RM (1990). "Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series." *Nature*, **344**, 734–741.

Takens F (1981). "Detecting strange attractors in turbulence." *Dynamical Systems and Turbulence, Lecture Notes in Mathematics*, **898**, 366–381.

Tilman D, Knops J, Wedin D, Reich P, Ritchie M, Siemann E (1997). "The Influence of Functional Diversity and Composition on Ecosystem Processes." *Science*, **277**(5330), 1300–1302. http://www.sciencemag.org/content/277/5330/1300.full.pdf.

Ye H, Sugihara G (2016). "Information leverage in interconnected ecosystems: Overcoming the curse of dimensionality." *Science*, **353**(6302), 922–925.

Affiliation:

Hao Ye
Scripps Institution of Oceanography
University of California, San Diego
9500 Gilman Drive, MC 0202
La Jolla, CA USA 92093-9202
E-mail: hye@ucsd.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd