Tutorial in Empirical Dynamic Modeling and Convergent Cross-Mapping

Hao Ye
Adam Clark
Ethan Deyle

August 9, 2015

Contents

1 Introduction

Empirical dynamic modeling (EDM) is an emerging non-parametric framework for modeling nonlinear dynamic systems. In an ecological context, EDM has numerous applications, including forecasting population abundances, unraveling species interactions, and identifying causal drivers. In contrast to the conventional approach of fitting assumed model equations to data, EDM relies on the fact that ecosystems have dynamics that allows us to reconstruct attractors directly from time series. This approach (with minimal assumptions) makes EDM particularly suitable for studying ecosystems, which exhibit non-equilibrium dynamics (problematic for models that typically assume stable equilibria) and state-dependent behavior (interactions that change with system state). This guide is designed to introduce both the theory behind EDM, as well as providing practical examples for using the rEDM software package.

2 Logistics

In this workshop, we will be exclusively using the statistical language **R**.

There are numerous reasons for choosing a language, but the primary advantages of R are that it is freely available and open-source, and with much added functionality (by way of R's package system) that facilitates data analysis, modeling, and statistics. These traits make it well-suited for Ecology, as evinced by several popular books on R aimed at Ecologists and at least 5 other R-based workshops at this conference.

2.1 Installing R

R can be obtained from the main CRAN website. In nearly all cases, you can simply download the appropriate installer for your operating system and run it to install R on your system. This installation includes 3 main components:

- an interactive R console (where you can enter commands aka Read-Eval-Print-Loop)
- a basic text editor with some R script functionality
- the basic R software packages (`base`, `stats`, etc.)

2.2 Installing RStudio

The R console is usable by itself, but many people find the interface of an Integrated Development Environment (IDE) to be more convenient. RStudio is one of the more popular ones for R, with many added features for project management, version control, etc. Although all of our code examples will run fine in base R, if you are new to R or do not yet have a preferred IDE, we recommend you give RStudio a try.

2.3 R packages and Installing Rcpp

R's basic functionality is extended through packages. These extend the base functionality of R with additional functions, datasets, etc. Many packages are distributed through the Comprehensive R Archive Network (CRAN), though there are other, often more specialized, archives. For example, Bioconductor primarily focuses on high-throughput genomic data, but also distributes other packages. Many packages are also available independently on code-sharing sites, such as GitHub.

R packages can be installed manually by downloading the package file and using the R interface with the `install.packages` function. However, both the R console application and RStudio also have menu options to manage packages. Here, we will first install the **Rcpp** package, which allows R to interface with C++ code, and is required by rEDM, our R package for Empirical Dynamic Modeling.

2.4 Installing rEDM

rEDM is an Rcpp package, and contains both C++ and R code. Because the C++ code needs to be compiled prior to usage, we present several different options for obtaining and installing the rEDM package, depending on your preference.

2.4.1 Binary Version

Most users will want the precompiled binary version. This can be downloaded from Github here. Please be sure to download the latest version (0.2.6 as of August 9, 2015). Clicking on the "view the full file" link should initiate a downlaod of the .tgz or .zip file, which you can save anywhere you like. Note that you do not want to unpack the .tgz or .zip file, since R expects a single package file, rather than a folder.

To install the downloaded package, use the standard R command (`install.packages`) as below, replacing **∗∗∗** with the name of the package file. (You will need to either give the complete path, or put the package file in R's current working directory.)

```
install.packages("***", type = "source", repos = NULL)
```

2.4.2 Source Version

The raw source can be downloaded here. If you are familiar with using Git with Rstudio projects, you may find it easier to clone directly. It can also be installed using functions from the **devtools** package:

```
library(devtools)
install_github("ha0ye/rEDM")
```

Note that this method requires **Git** to be installed, along with a **C++11** compiler, in addition to **Rcpp**. We have successfully tested this method of installation using both **Rtools 3.1** for Windows and **XCode 5.0+** for Macintosh.

3 Empirical Dynamic Modeling

3.1 Time Series as Observations of a Dynamic System

The essential concept behind Empirical Dynamic Modeling (EDM) is that time series can be viewed as projections of the behavior of a dynamic system. This framework requires only a few modest assumptions:

- The system state can be described as a single point in a high-dimensional space. Axes can be thought of as fundamental state variables, such as population abundances, resources, or environmental properties a la the Hutchinson Niche.
- The system state changes through time following some deterministic rules. In other words, the behavior is not completely random.

Projecting the system state to an axis then gives the value of the corresponding state variable, and sequential projections over time produce a time series. Different time series observed from the system can capture different state variables, but are more generally, some function of the system state and may convolve several different state variables.
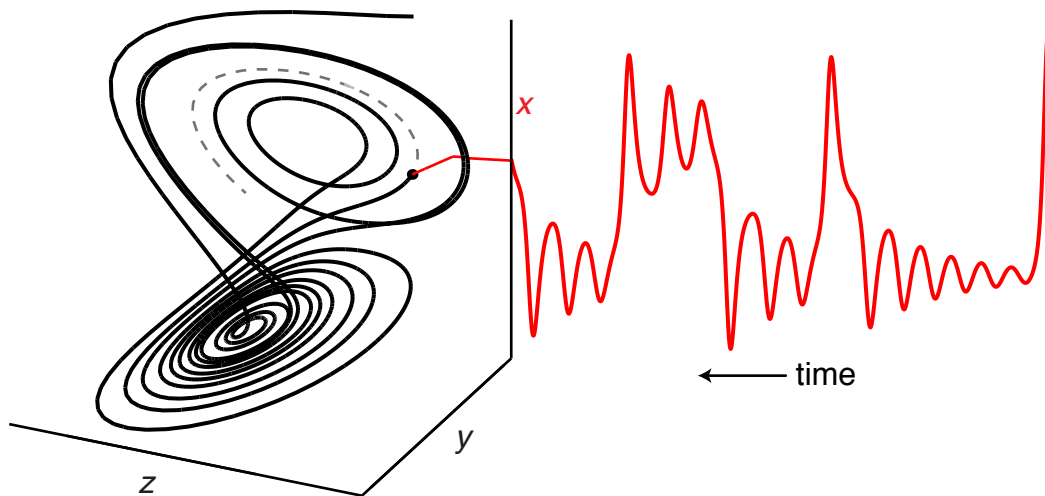
3.2 Attractor Reconstruction / Takens' Theorem

To reproduce this fundamental, geometric view of the system, one might suppose that time series of all the state variables are required. However, Takens' Theorem (Takens 1981) states that a mathematically equivalent reconstruction can be created by substituting lags of a time series for the unknown or unobserved variables.

In other words, instead of representing a system state as being composed of multiple different state variables, we instead use a lagged-coordinate embedding:

$$\vec{x}_t = \langle x_t, x_{t-\tau}, \dots, x_{t-(E-1)\tau} \rangle$$
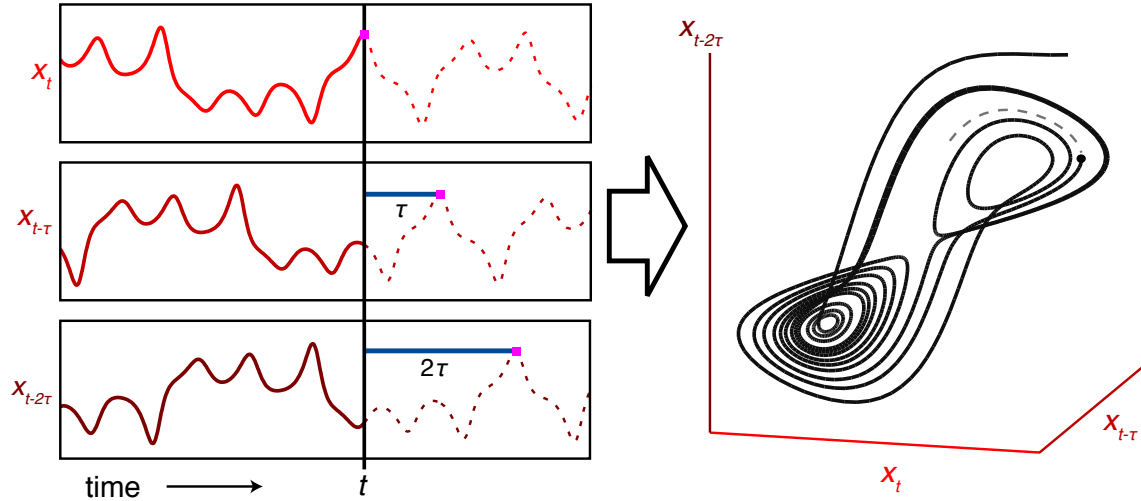
If sufficient lags are used, the reconstruction preserves essential mathematical properties of the original system. For instance, the points will map one-to-one to actual system states, and nearest neighbors in the reconstruction correspond to similar system states and behave similarly in the near future.

3.3 Nearest Neighbor Forecasting using Simplex Projection

One application of the reconstructed attractor is prediction. This can be accomplished using nearest neighbor forecasting methods, because of the similar behavior of nearby points in the reconstruction. One such method is **simplex projection** (Sugihara and May 1990). Simplex Projection is implemented in rEDM as the function `simplex`, and can be used both for prediction or to identify the optimal embedding dimension by quantifying the forecast skill of reconstructions with different dimensionality.

3.3.1 Example

First, we load the data and look at the format. Because this dataset is part of the `rEDM` package, we need to first load the package into R, before we have access to its datasets and functions.

```
library(rEDM)
data(tentmap_1d)
head(tentmap_1d)
```

```
## [1] -0.0992003 -0.6012986  0.7998003 -0.7944096  0.7979992 -0.8195405
```

We can see that the data consists of just a single vector, containing the raw data (first-differences from a tentmap). Because the `simplex` function can accept a single vector as the input time series, we don't need further processing of the data. Furthermore, because the data come from a discrete-time model, we can let many of the parameters be default values (e.g., $\tau = 1$, tp = 1). The default values for the embedding dimension, E, range from 1 to 10, and so the output will allow us to determine which embedding dimension best unfolds the attractor.

We need to specify what portions of the data to use for constructing the simplex projection model, and what portions to use for testing the forecast skill. By default, `simplex` will use leave-one-out cross-validation over the entire time series, but because the data contain no observational noise, and are particularly long, we'd like to be more conservative.

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

This will use the first 100 points (time = 1 to 100) in the time series as the "library" to construct the model, and 300 points (time = 201 to 500) as the "prediction set" to test the model.

Note that if the code detects any overlap in the lib and pred, it will enable leave-one-out cross-validation and return a warning message.

```
ts <- tentmap_1d
simplex_output <- simplex(ts, lib, pred)
```

The results are a simple data.frame with columns for each of the model parameters and forecast statistics, and rows for each run of the model. In this case, there is one run for each value of E, so we can simply plot E against ρ, the correlation between observed and predicted values:

```
par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0))
plot(simplex_output$E, simplex_output$rho, type = "l", xlab = "Embedding Dimension (E)",
    ylab = "Forecast Skill (rho)")
```
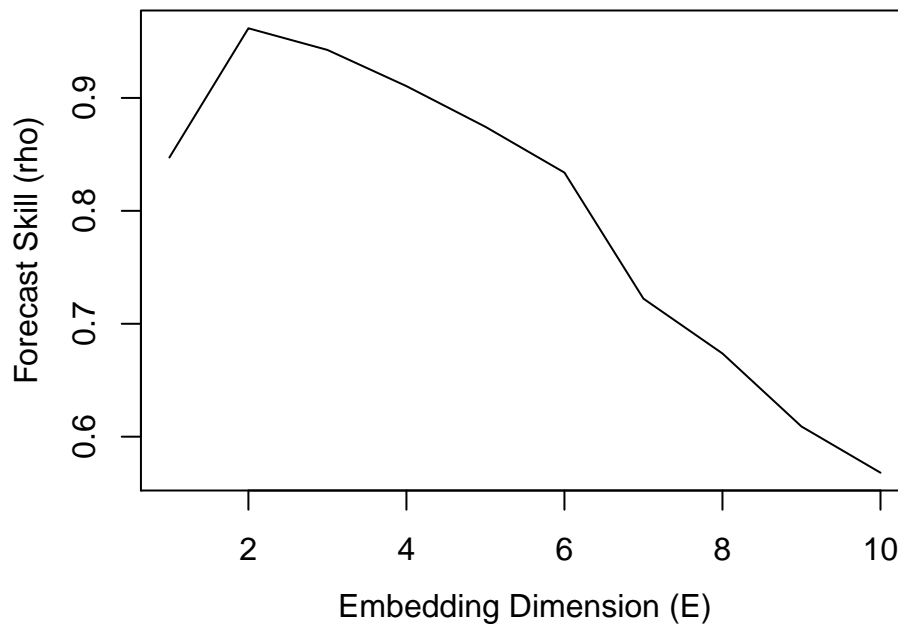

3.4   Prediction Decay

An important property of deterministic chaos is that nearby trajectories eventually diverge over time (the so-called "butterfly effect"). This means that prediction is primarily limited to short-term forecasts, because over the long-term, the system state may be viewed as essentially random. This property also differentiates nonlinear systems from the equilibrium view, where the system can be expected to settle around a stable point.

3.4.1   Example

We can test for this property by adjusting the `tp` parameter in the models, which determines how far into the future the model seeks to predict:

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

Here, we can simply plot tp against ρ to see how forecast accuracy changes as we predict further and further into the future:

```
par(mar = c(4, 4, 1, 1))
plot(simplex_output$tp, simplex_output$rho, type = "l", xlab = "Time to Prediction (tp)",
    ylab = "Forecast Skill (rho)")
```
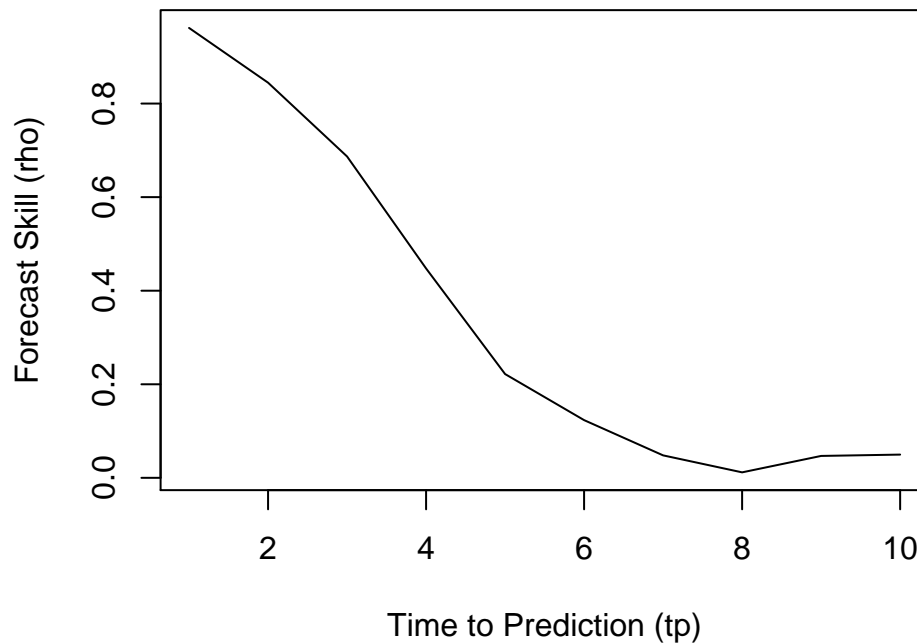

3.5 Identifying Nonlinearity

One concern is that many time series may show predictability even if they are purely stochastic, because they behave similarly to autocorrelated red noise. Luckily, there are additional tests that can be done to distinguish between red noise and deterministic behavior, by quantifying the degree of "nonlinearity" in the data.

Here, we use the **S-map** forecasting method, that is based on fitting local linear maps for prediction instead of the nearest-neighbor interpolation of simplex projection (Sugihara 1994). In addition to the parameters for simplex projection, S-map also contains a nonlinear tuning parameter, θ that affects the weights associated with individual points when fitting the local linear map. When $\theta = 0$, all weights are equal, and the S-map is identical to an autoregressive model; values of θ above 0 give greater weight to nearby points in the state space, thereby accommodating nonlinear behavior by allowing the local linear map to vary in state-space. For autoregressive red noise, the linear model should perform better, because the S-map model can reduce observation error by averaging over many points instead of just the most nearby points.

Thus, varying θ allows us to compare equivalent linear and nonlinear models as a test for nonlinear dynamics (after first using simplex projection to estimate the optimal embedding dimension for a time-series.)

3.5.1 Example

Following from the previous example, we set `E = 2` based on the results from simplex projection. Again, note that we allow many of the parameters take on default values (e.g., $\tau = 1$, tp $= 1$). If we had changed these

for simplex projection, we would want to propagate them here. The default values for the nonlinear tuning parameter, θ, range from 0 to 8, and are suitable for our purposes.

Note also, that the default value for `num_neighbors` is 0. Typically, when using `s_map` to test for nonlinear behavior, we allow all points in the reconstruction to be used, subject only to the weighting based on distance. By using 0 for this parameter (an otherwise nonsensical value), the program will use all nearest neighbors.

```
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, and rows for each run of the model. In this case, there is one run for each value of θ, so we can simply plot θ against ρ:

```
par(mar = c(4, 4, 1, 1), mgp = c(2.5, 1, 0))
plot(smap_output$theta, smap_output$rho, type = "l", xlab = "Nonlinearity (theta)",
    ylab = "Forecast Skill (rho)")
```
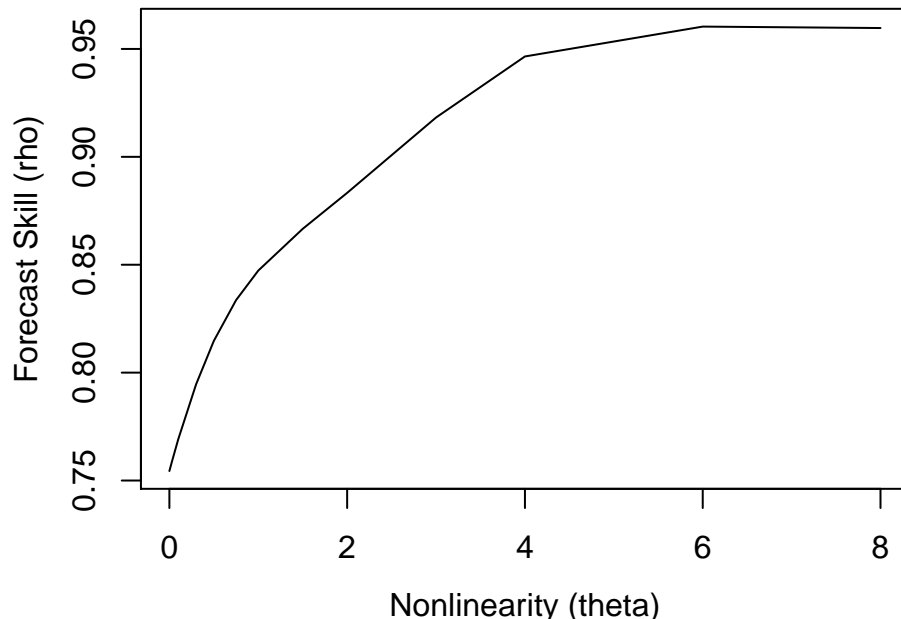

3.6 Generalized Takens' Theorem

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 depiction of the true dynamics than "univariate" counterparts.

In rEDM, the `block_lnlp` function generalizes the `simplex` and `s_map` functions, allowing generic reconstructions to be used with either of the simplex projection or S-map algorithms. 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 and each row represents the variables observed at the same time. In addition to the typical arguments for `simplex` or `s_map`, `block_lnlp` contains arguments to specify which column is to be forecast (`target_column`) as well as which columns to use to construct the attractor (`columns`). In both cases, either a numerical index or the column name can be given. For obvious reasons, column names will work only if the input data has column names.

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.

3.6.1 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).

```
data(block_3sp)
head(block_3sp)
```

```
##   time        x_t       x_t-1       x_t-2         y_t       y_t-1       y_t-2
## 1    1 -0.7418625          NA          NA -1.2681036          NA          NA
## 2    2  1.2448818 -0.7418625          NA  1.4888875 -1.2681036          NA
## 3    3 -1.9176852  1.2448818 -0.7418625 -0.1131881  1.4888875 -1.2681036
## 4    4 -0.9623176 -1.9176852  1.2448818 -1.1067786 -0.1131881  1.4888875
## 5    5  1.3318751 -0.9623176 -1.9176852  2.3850408 -1.1067786 -0.1131881
## 6    6 -0.8170829  1.3318751 -0.9623176 -0.6753463  2.3850408 -1.1067786
##          z_t       z_t-1       z_t-2
## 1 -1.8639802          NA          NA
## 2 -0.4815825 -1.8639802          NA
## 3  1.5352388 -0.4815825 -1.8639802
## 4 -1.4929558  1.5352388 -0.4815825
## 5 -1.1194762 -1.4929558  1.5352388
## 6  0.7466579 -1.1194762 -1.4929558
```

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` now points to the first *data* column in the data.frame, as opposed to the time column (`columns` is similarly indexed).

```
lib <- c(1, NROW(block_3sp))
pred <- c(1, NROW(block_3sp))

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)
```

We can also run the same model by referring to the names of the columns directly.

```
block_lnlp_output <- block_lnlp(block_3sp, lib = lib, pred = pred, columns = c("x_t",
    "x_t-1", "y_t"), target_column = "x_t", stats_only = FALSE, first_column_time = TRUE)
```

Note that we did not specify a value for the `tp` parameter. Here, the default value of `1` means that the program will predict the target variable 1 time step into the future (based on the row-structure of the input data). In some cases, the data may already be processed into a format where one wants to predict a different column that has already been aligned correctly. In that case, one can set `tp = 0` when calling `block_lnlp`.

By setting `stats_only` to `FALSE`, we get back a list with the full model output. Only 1 model was run, so the output is a list with 1 element. To extract the raw predictions, we can go into the `model_output` variable and pull out the observed and predicted values, plotting them to see how well the model fit 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), pty = "s")
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")
```
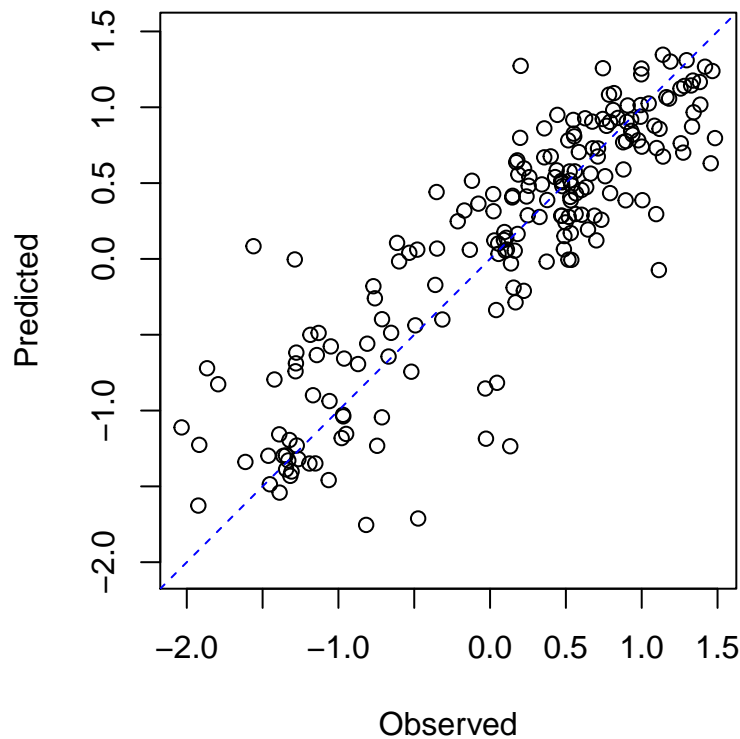

3.7   Causality Inference and Cross Mapping

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 x or y alone should uniquely identify the system state and the corresponding value of the other variable. Thus, it should be possible to use one variable to cross-predict the other.
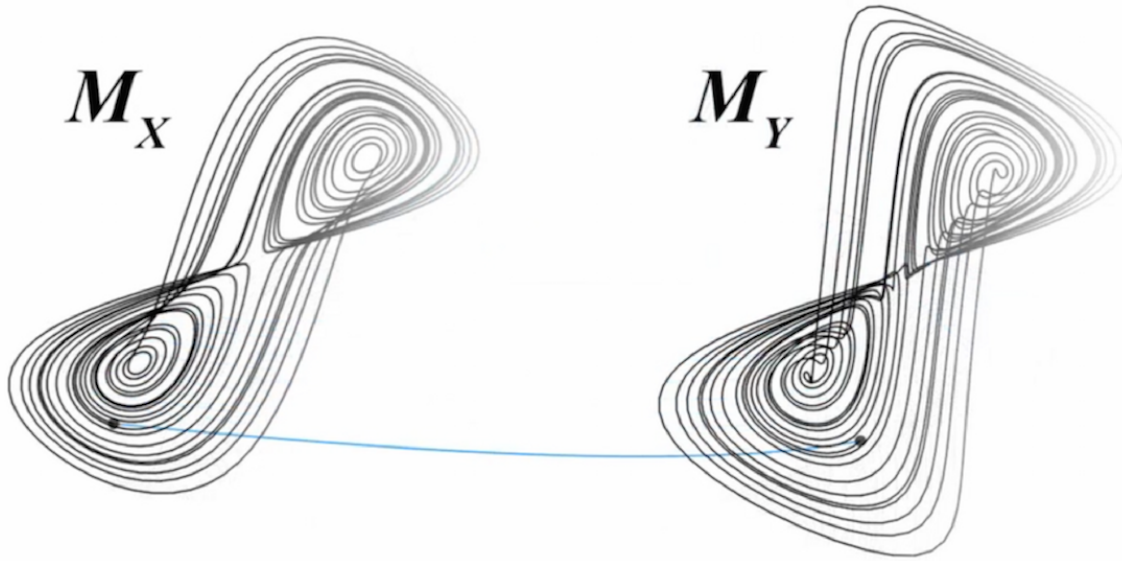
In the case of unidirectional causality, x causes y, then the behavior of the causal variable (x) leaves a signature on the affected variable (y). In such cases, the reconstructed states based on y can be used to cross-predict the values of x (because the reconstruction based on y must be complete, it 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 fail to improve as longer time series are used for reconstruction. In contrast, in the absence of noise, the cross-prediction of x from y will continually improve. This convergence is therefore necessary to infer 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.

3.8   Convergent Cross Mapping (CCM)

In the rEDM package, the `ccm` function presents an easy way to compute cross map skill for different subsamples of the time series, enabling observation of both convergence and uncertainty regarding cross map skill. In the following example, we use CCM to identify causality between anchovy landings in California and Newport Pier sea-surface temperature.

Here, we use a previously identified value of `3` for the embedding dimension. We set `lib_sizes` (the number of library vectors) to vary from `10` to `80` in steps of `10`. Setting `num_samples` to `100` means that `100` different library samples will be generated, by random sampling (`random_libs = TRUE` by default) from the possible vectors with replacement (`replace = TRUE` by default).

```
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), random_libs = FALSE)
```

```
sst_xmap_anchovy <- ccm(sardine_anchovy_sst, E = 3, lib_column = "np_sst", target_column = "anchovy",
    lib_sizes = seq(10, 80, by = 10), random_libs = FALSE)
```

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 = 800 rows). Because we cross map using multiple libraries at each library size, we'd like to aggregate the results and plot the average cross map skill at each library size. Because average cross map skill less than 0 is noninformative, 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))
plot(a_xmap_t_means$lib_size, pmax(0, a_xmap_t_means$rho), type = "l", col = "red",
    xlab = "Library Size", ylab = "Cross Map Skill (rho)", ylim = c(0, 0.4))
lines(t_xmap_a_means$lib_size, pmax(0, t_xmap_a_means$rho), col = "blue")
legend(x = "topleft", legend = c("anchovy xmap SST", "SST xmap anchovy"), col = c("red",
    "blue"), lwd = 1, inset = 0.02, cex = 0.8)
```
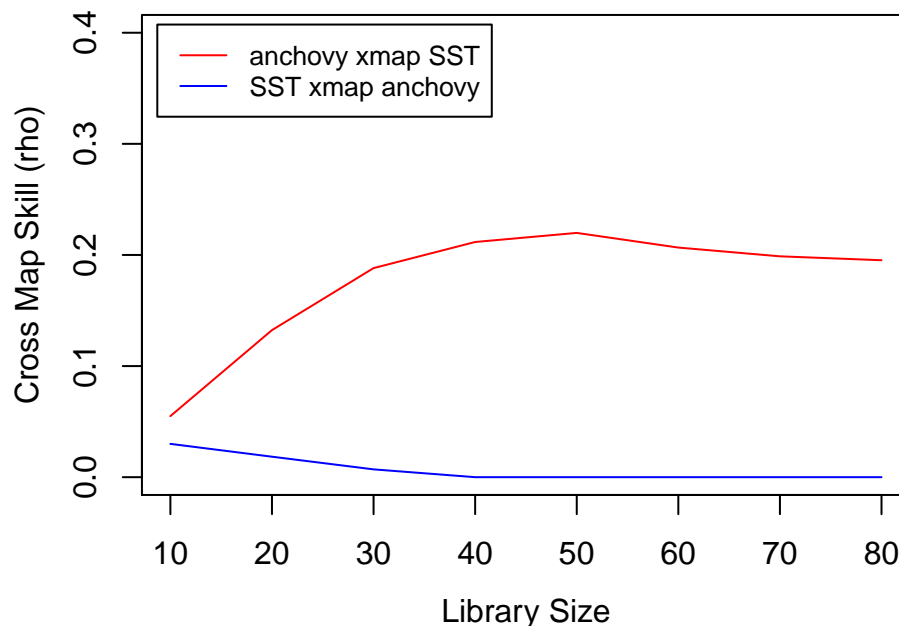

4 Real Data Examples

4.1 Community Productivity and Invasibility

The data presented here are part of Experiment 120, the "Big Biodiversity" experiment at Cedar Creek LTER. 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. Full methods are described in (Tilman et al. 1997). The most well-known result from the experiment is that planted species number strongly, positively influences above-ground biomass production. However, because the diversity treatments are fixed, rather than dynamical variables, they do not lend themselves to state space reconstruction.

Instead, we focus a different set of published results from the experiment: interactions between primary productivity, soil nitrate, and invasion rates by non-planted species. 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. For the analyses here, we combine planted diversity treatments from 4-8 species planted treatments, and analyze them as a block.

The columns in the dataset `e120_invdat` are as follows: `Exp`, `Year`, `Month`, `Plot`, `Field`, and `FieldPlot` describe experiment, plot identity, and sampling time. `NumSp` and `SpNum` show the planted and realized species diversity in the plot respectively. `AbvBioAnnProd` shows annual aboveground productivity of planted species, in g/m2. `noh020tot` shows soil nitrate levels in the top 20 cm of soil, measured in ug/kg soil. `invrichness` shows species richness of unplanted species in the plot. `SummerPrecip.mm.` shows precipitation annual from May to August measured in mm.

4.1.1 Preparing the Data

E120 includes data from multiple plots, meaning that we first need to collapse it into a single composite time series. As before, we begin by normalizing each time series.

```
data(e120_diversity)
head(e120_diversity)
```

```
##    Exp Year Month Plot NumSp SpNum AbvBioAnnProd noh020tot invrichness
## 1 120 1996     8    3     4     5       35.1670   0.22520          11
## 2 120 1997     8    3     4     5       65.9167   0.14430           7
## 3 120 1998     8    3     4     5      195.5670   0.07070           8
## 4 120 1999     8    3     4     5       69.4092   0.02610           7
## 6 120 2001     8    3     4     5       80.6292   0.18045           3
## 7 120 2002     8    3     4     5      143.5750   0.01130           9
##    Field FieldPlot SummerPrecip.mm.
## 1      1       1 3          447.548
## 2      1       1 3          445.516
## 3      1       1 3          356.108
## 4      1       1 3          487.680
## 6      1       1 3          356.870
## 7      1       1 3          484.886
```

```
# separate time column from data
composite_ts <- e120_diversity[, c(7:9, 12)]

# normalize each time series
n <- NCOL(composite_ts)
blocks <- e120_diversity$Plot
blocks_index <- sort(unique(blocks))
for (j in 1:n) {
    for (i in 1:length(blocks_index)) {
        subs <- which(blocks == blocks_index[i])
        composite_ts[subs, j] <- (composite_ts[subs, j] - mean(composite_ts[subs,
            j]))/sd(composite_ts[subs, j])
    }
}


composite_ts <- cbind(year = e120_diversity$Year, composite_ts)
```

Again, we need to indicate separations between plots so that lagged vectors are not constructed that contain coordinates spanning multiple time series.

```
#make composite library
segments <- NULL
startpos <- 1
for(i in 2:nrow(composite_ts)) {
    if(composite_ts$year[i] < composite_ts$year[i-1]) {
        segments <- rbind(segments, c(startpos, i))
        startpos <- i+1
        }
    }
segments <- rbind(segments, c(max(segments)+1, nrow(composite_ts)))

#Choose random segments for prediction
set.seed(2312)
rndlib <- sort(sample(1:nrow(segments), round(nrow(segments)/2,0), rep=FALSE))
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_diversity[,c("Year", "SummerPrecip.mm.")])
precip_ts <- precip_ts[order(precip_ts$Year),]
```

4.1.2 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:
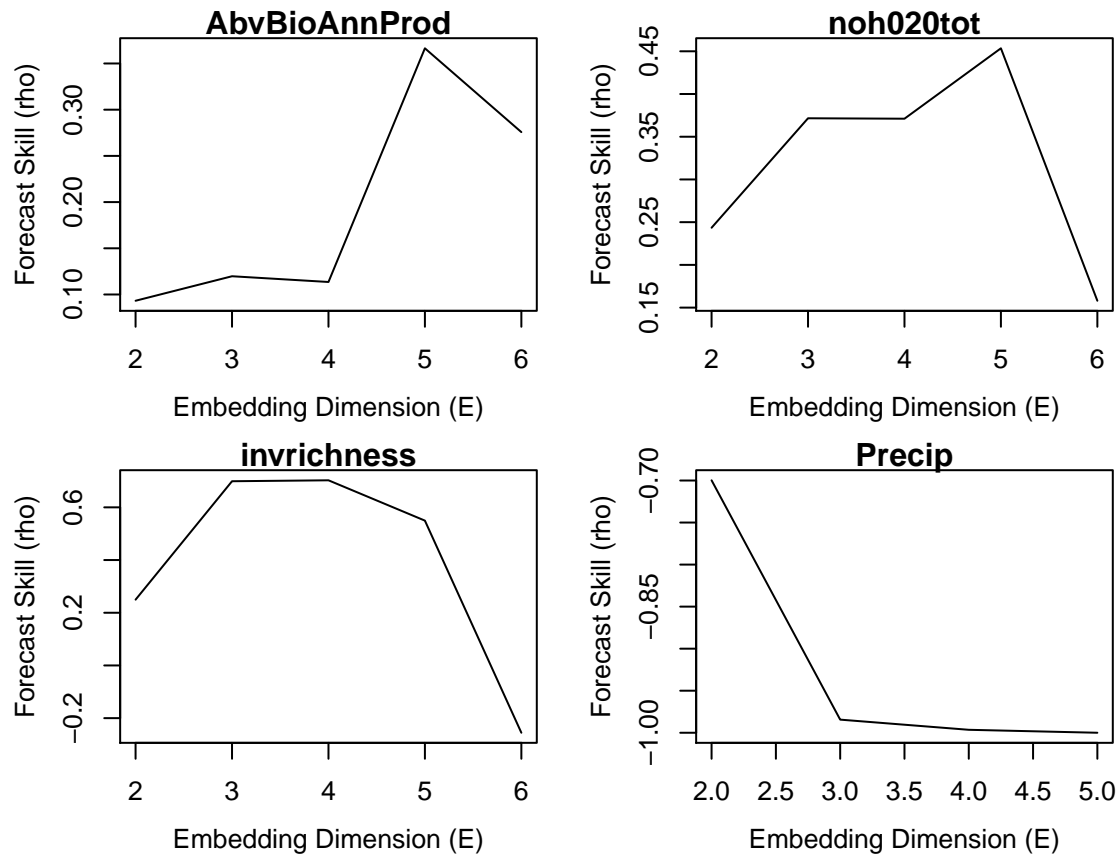
```
par(mar = c(4, 4, 1, 1), mfrow = c(2, 2), mgp = c(2.5, 1, 0))
varlst <- colnames(composite_ts)[2:4]
simplex_output_list <- NULL

for (i in 1:length(varlst)) {
    simplex_output_list[[i]] <- simplex(composite_ts[, c("year", varlst[i])],
        lib = composite_lib, pred = composite_pred, E = c(2:6))
    plot(simplex_output_list[[i]]$E, simplex_output_list[[i]]$rho, type = "l",
        xlab = "Embedding Dimension (E)", ylab = "Forecast Skill (rho)", main = varlst[i])
}

simplex_output_list[[4]] <- simplex(precip_ts, lib = c(1, 7), pred = c(1, 7),
    E = c(2:5), silent = TRUE)
names(simplex_output_list) <- c(varlst, "precipmm")
plot(simplex_output_list[[4]]$E, simplex_output_list[[4]]$rho, type = "l", xlab = "Embedding Dimension
    ylab = "Forecast Skill (rho)", main = "Precip")
```

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

```
bestE <- sapply(simplex_output_list, function(simplex_output) {
    simplex_output$E[which.max(simplex_output$rho)]
    })
bestE
```

```
## AbvBioAnnProd     noh020tot   invrichness       precipmm
##             5             5             4              2
```

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

```
par(mar = c(4, 4, 1, 1), mfrow = c(2, 2), mgp = c(2.5, 1, 0))
smap_output_list <- NULL

for (i in 1:length(varlst)) {
    smap_output_list[[i]] <- s_map(composite_ts[, c("year", varlst[i])], lib = composite_lib,
        pred = composite_pred, E = bestE[i], silent = TRUE)
    plot(smap_output_list[[i]]$theta, smap_output_list[[i]]$rho, type = "l",
        xlab = "Nonlinearity (theta)", ylab = "Forecast Skill (rho)", main = varlst[i])
}

smap_output_list[[4]] <- s_map(precip_ts, E = bestE[4], silent = TRUE)
plot(smap_output_list[[4]]$theta, smap_output_list[[4]]$rho, type = "l", xlab = "Nonlinearity (theta)",
    ylab = "Forecast Skill (rho)", main = "Precip")
```
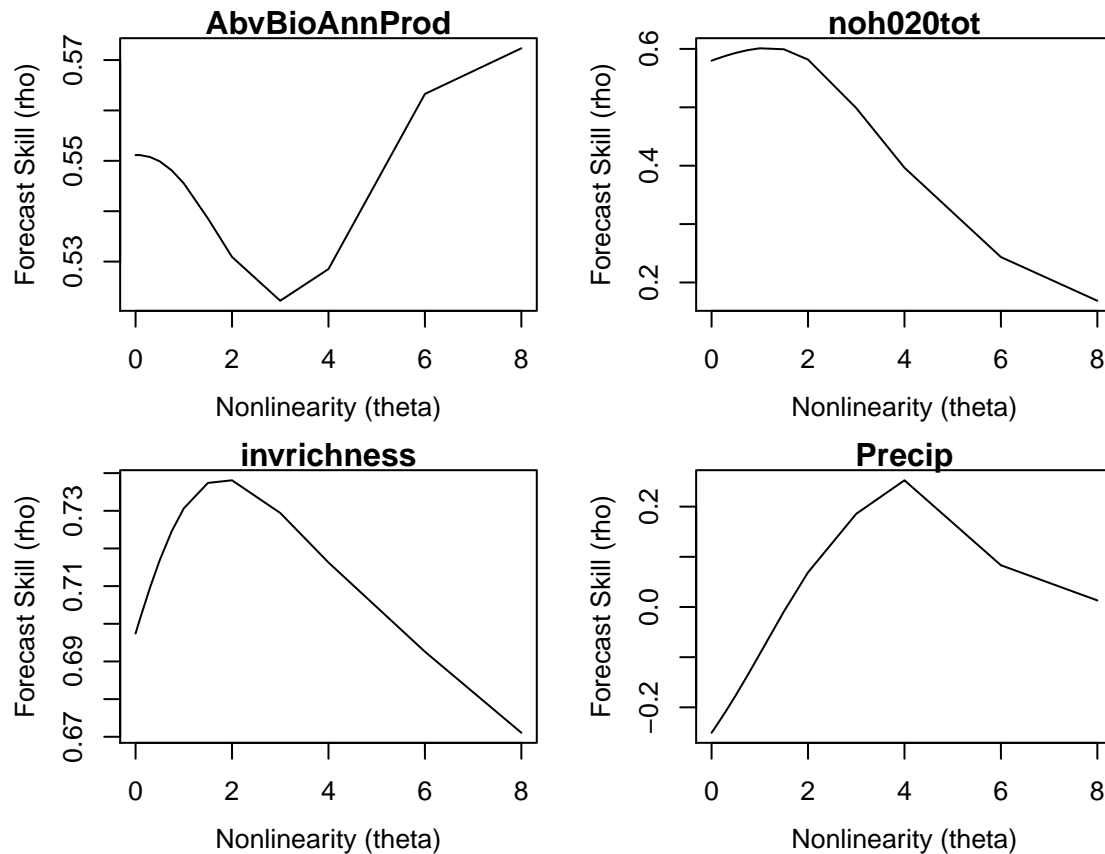
Note that all time series suggest nonlinear dynamics in the data (because of the initial rise in rho for non-zero theta, followed by a sharp drop-off in rho with theta).

4.1.3 Multivariate Models

Next, we can use information from several time series to make better predictions about system dynamics. We can accomplish this with the `block_lnlp` command. First, we need to manually construct 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.

```
n <- NROW(composite_ts)

# Make lags
block_data <- data.frame(year = composite_ts$year)
block_data$AB_tm <- composite_ts$AbvBioAnnProd
block_data$AB_tm1 <- c(NA, block_data$AB_tm[-n])
block_data$AB_tm2 <- c(NA, block_data$AB_tm1[-n])
block_data$AB_tm3 <- c(NA, block_data$AB_tm2[-n])

block_data$NO_tm <- composite_ts$noh020tot
block_data$NO_tm1 <- c(NA, block_data$NO_tm[-n])
block_data$NO_tm2 <- c(NA, block_data$NO_tm1[-n])
block_data$NO_tm3 <- c(NA, block_data$NO_tm2[-n])

block_data$IV_tm <- composite_ts$invrichness
block_data$IV_tm1 <- c(NA, block_data$IV_tm[-n])
```

```r
block_data$IV_tm2 <- c(NA, block_data$IV_tm1[-n])
block_data$IV_tm3 <- c(NA, block_data$IV_tm2[-n])

block_data$PR_tm <- composite_ts$SummerPrecip.mm
block_data$PR_tm1 <- c(NA, block_data$PR_tm[-n])
block_data$PR_tm2 <- c(NA, block_data$PR_tm1[-n])
block_data$PR_tm3 <- c(NA, block_data$PR_tm2[-n])

# Remove overlaps from other plots
startyear <- 1996
for (i in 2:nrow(block_data)) {
    if (block_data$year[i] < block_data$year[i - 1]) {
        startyear <- block_data$year[i]
    }
    if (block_data$year[i] == startyear) {
        block_data[i, c("AB_tm1", "NO_tm1", "IV_tm1", "PR_tm1")] <- NA
        block_data[i, c("AB_tm2", "NO_tm2", "IV_tm2", "PR_tm2")] <- NA
        block_data[i, c("AB_tm3", "NO_tm3", "IV_tm3", "PR_tm3")] <- NA
    }
    if (block_data$year[i] == (startyear + 1)) {
        block_data[i, c("AB_tm2", "NO_tm2", "IV_tm2", "PR_tm2")] <- NA
        block_data[i, c("AB_tm3", "NO_tm3", "IV_tm3", "PR_tm3")] <- NA
    }
    if (block_data$year[i] == (startyear + 2)) {
        block_data[i, c("AB_tm3", "NO_tm3", "IV_tm3", "PR_tm3")] <- NA
    }
}
head(block_data[, 1:5], 20)
```

```
##    year        AB_tm      AB_tm1      AB_tm2      AB_tm3
## 1  1996 -1.0626351          NA          NA          NA
## 2  1997 -0.5456990 -1.0626351          NA          NA
## 3  1998  1.6338642 -0.5456990 -1.0626351          NA
## 4  1999 -0.4869863  1.6338642 -0.5456990 -1.0626351
## 5  2001 -0.2983658 -0.4869863  1.6338642 -0.5456990
## 6  2002  0.7598219 -0.2983658 -0.4869863  1.6338642
## 7  1996 -1.0139507          NA          NA          NA
## 8  1997 -0.9855735 -1.0139507          NA          NA
## 9  1998  1.4960850 -0.9855735 -1.0139507          NA
## 10 1999  0.3898716  1.4960850 -0.9855735 -1.0139507
## 11 2001 -0.4926840  0.3898716  1.4960850 -0.9855735
## 12 2002  0.6062517 -0.4926840  0.3898716  1.4960850
## 13 1996 -1.4989147          NA          NA          NA
## 14 1997 -0.6867934 -1.4989147          NA          NA
## 15 1998  1.5499186 -0.6867934 -1.4989147          NA
## 16 1999  0.3648442  1.5499186 -0.6867934 -1.4989147
## 17 2000  0.6089230  0.3648442  1.5499186 -0.6867934
## 18 2001 -0.5679305  0.6089230  0.3648442  1.5499186
## 19 2002  0.2299527 -0.5679305  0.6089230  0.3648442
## 20 1996 -1.3230421          NA          NA          NA
```

Now, we can run the `block_lnlp` algorithm on the composite, multi-variate 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:

```
block_lnlp_output_AB <- block_lnlp(block_data, lib = composite_lib, pred = composite_pred,
    columns = c("AB_tm", "AB_tm1", "AB_tm2"), target_column = 1, stats_only = FALSE,
    first_column_time = TRUE)

block_lnlp_output_ABPR <- block_lnlp(block_data, lib = composite_lib, pred = composite_pred,
    columns = c("AB_tm", "AB_tm1", "AB_tm2", "PR_tm", "PR_tm1", "PR_tm2"), target_column = 1,
    stats_only = FALSE, first_column_time = TRUE)

block_lnlp_output_ABPR <- block_lnlp(block_data, lib = composite_lib, pred = composite_pred,
    columns = c("AB_tm", "AB_tm1", "AB_tm2", "PR_tm", "PR_tm1", "PR_tm2"), target_column = 1,
    stats_only = FALSE, first_column_time = TRUE)
```

Note that each additional variable adds slightly to the predictive power of the model.

```
observed_AB <- block_lnlp_output_AB[[1]]$model_output$obs
predicted_AB <- block_lnlp_output_AB[[1]]$model_output$pred

observed_ABPR <- block_lnlp_output_ABPR[[1]]$model_output$obs
predicted_ABPR <- block_lnlp_output_ABPR[[1]]$model_output$pred

par(mar = c(4, 4, 1, 1), pty = "s", mgp = c(2.5, 1, 0))
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_ABPR, predicted_ABPR, pch = 2, col = "red")
abline(lm(predicted_ABPR ~ observed_ABPR), col = "red", lty = 3, lwd = 2)
```
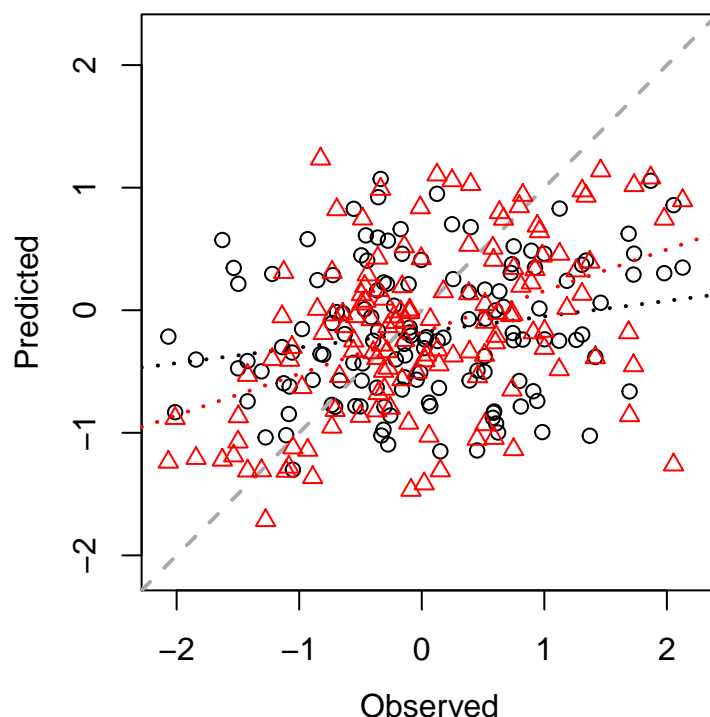
4.1.4 Convergent Cross Mapping

Finally, we can apply CCM to our data in order to test for causal links among variables.

In each case, 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).

```r
# A. repens:
no_xmap_inv <- ccm(composite_ts, lib = segments, pred = segments, E = bestE[4],
    lib_column = "noh020tot", target_column = "invrichness", lib_sizes = c(seq(5,
        55, by = 2), seq(55, 400, by = 50)), num_samples = 100, silent = TRUE)
inv_xmap_no <- ccm(composite_ts, lib = composite_lib, pred = composite_pred,
    E = bestE[1], lib_column = "invrichness", target_column = "noh020tot", lib_sizes = c(seq(5,
        55, by = 2), seq(55, 400, by = 50)), num_samples = 100, silent = TRUE)

n_xmap_i_means <- data.frame(ccm_means(no_xmap_inv), sd.rho = with(no_xmap_inv,
    tapply(rho, lib_size, sd)))
i_xmap_n_means <- data.frame(ccm_means(inv_xmap_no), sd.rho = with(inv_xmap_no,
    tapply(rho, lib_size, sd)))

# S. scoparium:
ab_xmap_inv <- ccm(composite_ts, lib = segments, pred = segments, E = bestE[4],
    lib_column = "AbvBioAnnProd", target_column = "invrichness", lib_sizes = c(seq(5,
        55, by = 2), seq(55, 400, by = 50)), num_samples = 100, silent = TRUE)
inv_xmap_ab <- ccm(composite_ts, lib = segments, pred = segments, E = bestE[2],
    lib_column = "invrichness", target_column = "AbvBioAnnProd", lib_sizes = c(seq(5,
        55, by = 2), seq(55, 400, by = 50)), num_samples = 100, silent = TRUE)

a_xmap_i_means <- data.frame(ccm_means(ab_xmap_inv), sd.rho = with(ab_xmap_inv,
    tapply(rho, lib_size, sd)))
i_xmap_a_means <- data.frame(ccm_means(inv_xmap_ab), sd.rho = with(inv_xmap_ab,
    tapply(rho, lib_size, sd)))

# Plot output
par(mar = c(4, 4, 1, 1))
plot(n_xmap_i_means$lib_size, pmax(0, n_xmap_i_means$rho), type = "l", col = "red",
    xlab = "Library Size", ylab = "Cross Map Skill (rho)", ylim = c(0, 0.6),
    lwd = 2)
lines(i_xmap_n_means$lib_size, pmax(0, i_xmap_n_means$rho), col = "blue", lwd = 2)
legend(x = "topleft", legend = c("Nitrate xmap Inv. Richness", "Inv. Richness xmap Nitrate"),
    col = c("red", "blue"), lwd = 2, inset = 0.02, bty = "n", cex = 0.8)
abline(h = 0, lty = 3, col = "darkgrey", lwd = 2)

# Add CI's
lines(n_xmap_i_means$lib_size, n_xmap_i_means$rho + n_xmap_i_means$sd.rho, col = "red",
    lty = 2, lwd = 2)
lines(n_xmap_i_means$lib_size, n_xmap_i_means$rho - n_xmap_i_means$sd.rho, col = "red",
    lty = 2, lwd = 2)
lines(i_xmap_n_means$lib_size, i_xmap_n_means$rho + i_xmap_n_means$sd.rho, col = "blue",
    lty = 2, lwd = 2)
lines(i_xmap_n_means$lib_size, i_xmap_n_means$rho - i_xmap_n_means$sd.rho, col = "blue",
    lty = 2, lwd = 2)
```
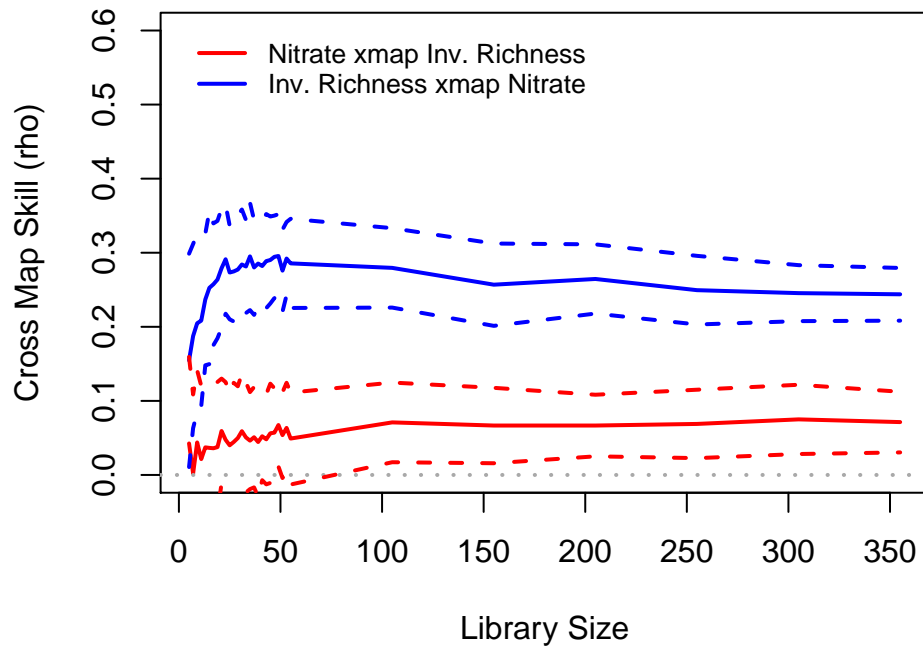
```r
plot(a_xmap_i_means$lib_size, pmax(0, a_xmap_i_means$rho), type = "l", col = "orange",
    xlab = "Library Size", ylab = "Cross Map Skill (rho)", ylim = c(0, 0.6),
    lwd = 2)
lines(i_xmap_a_means$lib_size, pmax(0, i_xmap_a_means$rho), col = "blue", lwd = 2)
legend(x = "topleft", legend = c("Abv. Biomass xmap Inv. Richness", "Inv. Richness xmap Abv. Biomass"),
    col = c("orange", "blue"), lwd = 2, inset = 0.02, bty = "n", cex = 0.8)
abline(h = 0, lty = 3, col = "darkgrey", lwd = 2)

# Add CI's
lines(a_xmap_i_means$lib_size, a_xmap_i_means$rho + a_xmap_i_means$sd.rho, col = "orange",
    lty = 2, lwd = 2)
lines(a_xmap_i_means$lib_size, a_xmap_i_means$rho - a_xmap_i_means$sd.rho, col = "orange",
    lty = 2, lwd = 2)
lines(i_xmap_a_means$lib_size, i_xmap_a_means$rho + i_xmap_a_means$sd.rho, col = "blue",
    lty = 2, lwd = 2)
lines(i_xmap_a_means$lib_size, i_xmap_a_means$rho - i_xmap_a_means$sd.rho, col = "blue",
    lty = 2, lwd = 2)
```
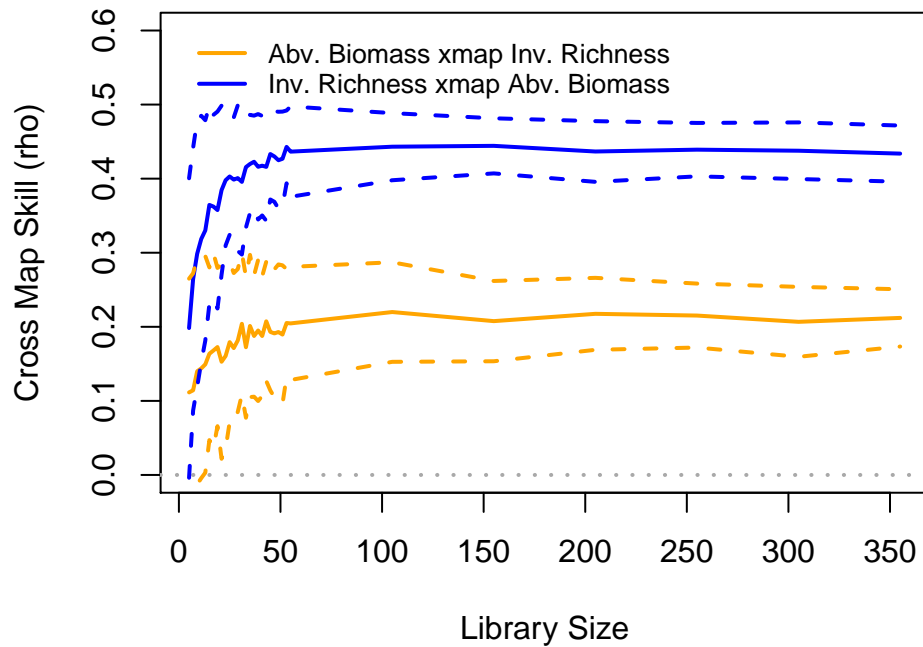
In each case, results suggest that invasive richness is driven by other variables more strongly than it influences them in return. In particular, while invasion dynamics appear to be strongly forced by soil nitrate dynamics, invasion does not appear influence plot soil nitrate at all. This makes sense, as invading species in these plots are quickly weeded out and should not have time to influence local environmental conditions. Causal forcing between biomass and invasion, on the other hand, may be bi-directional based on our analyses. Again, though, it makes sense 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).

4.2 Apple-Blossom Thrips

In this next example, we will use EDM methods to re-examine the classic apple-blossom thrips (Thrips imaginis) time series from the Wait Institute in Australia (Davidson and Andrewartha 1948a, 1948b). 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 lead to the hypothesis that regional-scale climatic factors were responsible for controlling the size of the seasonal outbreaks (what might now be called the Moran effect).

```
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. maxT_degC is the mean maximum daily temperature (degrees F) 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) that acts as an indicator of season.
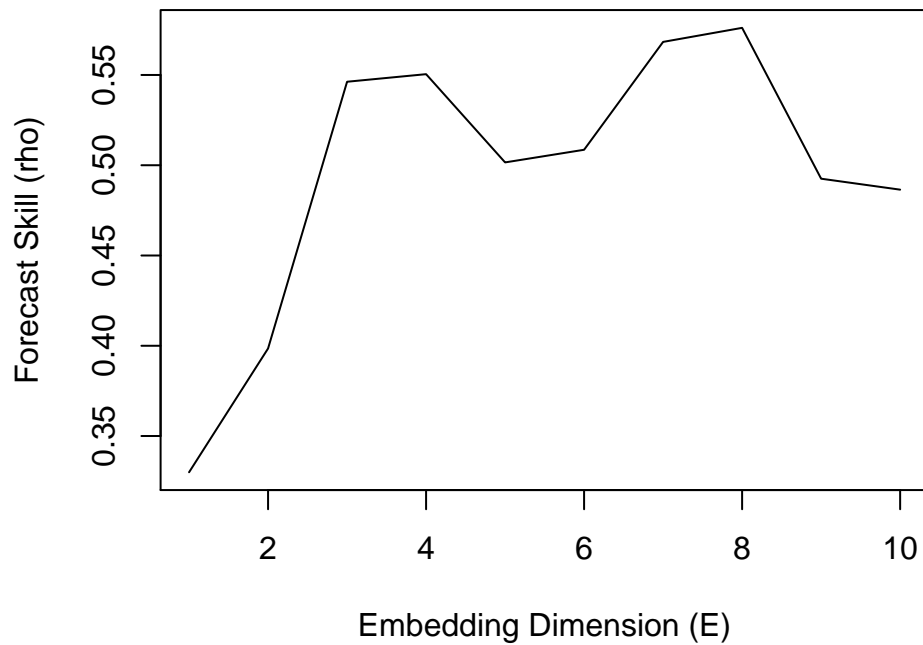
First, we plot the data.
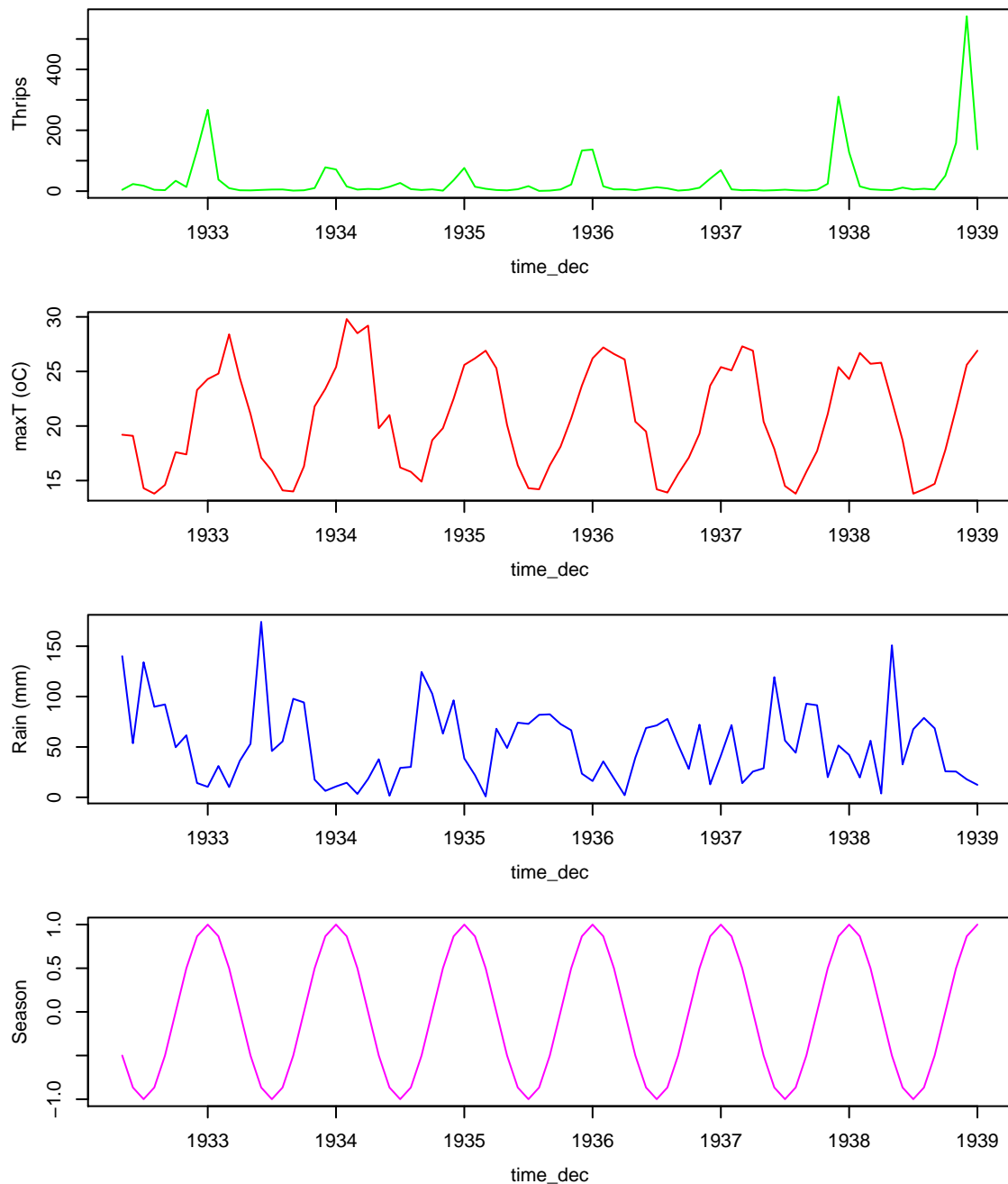
Note that all the time-series variables, particularly the mean maximum daily Temperature, show marked seasonality.

4.2.1 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)
```

While there is an initial peak in the simplex prediction 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-maps 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)
```
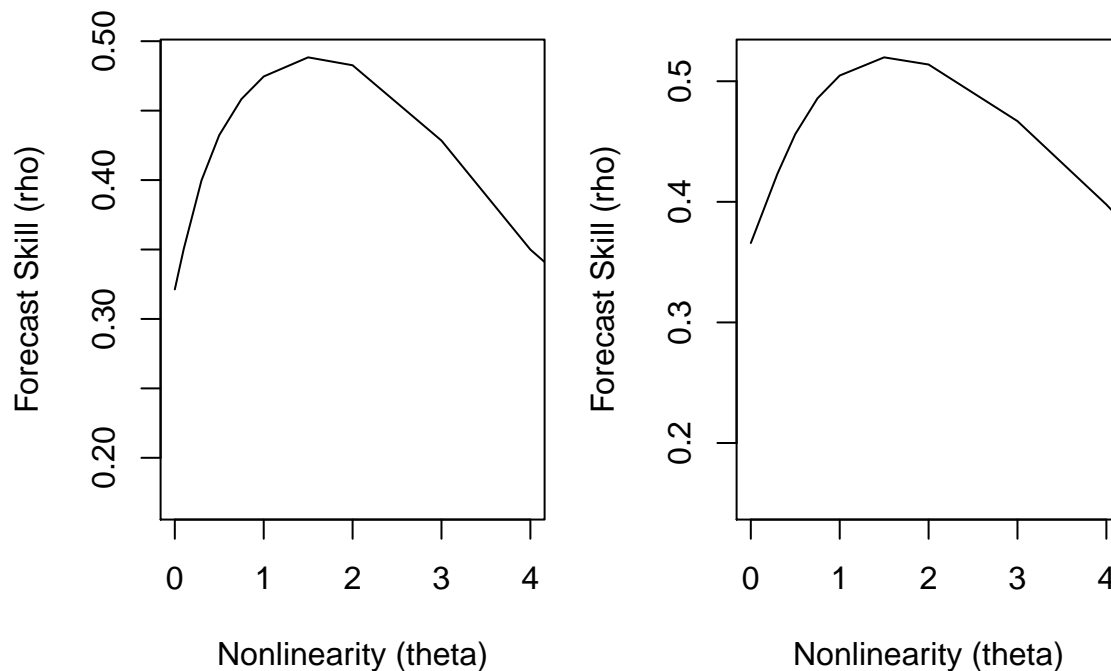

The S-map results demonstrate clear nonlinearity in the Thrips time series, as nonlinear models $\theta > 0$ give substantially better predictions than the linear model $\theta = 0$. This suggests that Thrips, despite the strong

seasonal dynamics, do not simply track the environment passively, but have some intrinsic dynamics. To look more closely at the issue of seasonal drivers, however, we turn to convergent cross-mapping (CCM).

4.2.2 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 with the full time-series is statistically significant. (2) Cross-map prediction demonstrates convergence, i.e. prediction skill increases as more of the time-series is used.

4.2.2.1 Cross-map matrix For an initial summary, we first simply compute the cross-map skill (measured with Pearson's ρ) with the full time-series.

```
ncol <- dim(thrips_block)[2] - 2
M_rho <- array(NA, dim = c(ncol, ncol), dimnames = list(colnames(thrips_block[3:6]),
    colnames(thrips_block[3:6])))

for (i in 1:ncol) {
    for (j in 1:ncol) {
        if (i != j) {
            out_temp <- ccm(thrips_block, E = 8, lib_column = 2 + i, target_column = 2 +
                j, lib_sizes = dim(thrips_block)[1], replace = FALSE, silent = TRUE)
            M_rho[i, j] <- out_temp$rho
        }
    }
}
```

4.2.2.2 Correlation matrix For comparison we also compute the lag cross-correlation, allowing lags up to ±6 months.

```
M_corr <- array(NA, dim = c(ncol, ncol), dimnames = list(colnames(thrips_block[3:6]),
    colnames(thrips_block[3:6])))

for (i in 1:ncol) {
    for (j in 1:ncol) {
        if (i != j) {
            cf_temp <- ccf(x = thrips_block[, 2 + i], y = thrips_block[, 2 +
                j], type = "correlation", lag.max = 6, plot = FALSE)$acf
            M_corr[i, j] <- max(abs(cf_temp))
        }
    }
}
```

We compare the two matrices.

Cross-map

```
##                  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
```

24

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 $maxT$ and the seasonal indicator is extremely high, and $Season$ can almost perfectly cross-map $maxT$, $\rho = 0.9625766$. This makes the interpretation of cross-mapping more complicated, because we have to consider synchrony. Let's ellaborate. It is clear from cross-mapping (or even just visual inspection) that seasonality drives Thrips abundance. Since the monthly mean maximum temperature is almost perfectly synchorinzed to the seasons, it contains the same information as the simple season indicator. Any variable that can predict (cross-map) the seasonal cycle, i.e. was influenced by seasonality, will therefore also predict $maxT$, regardless of if temperature is actually the mechanism of seasonal forcing.

4.2.2.3   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)

xmap_means <- list(ccm_means(thrips_xmap_maxT), ccm_means(maxT_xmap_thrips))
```
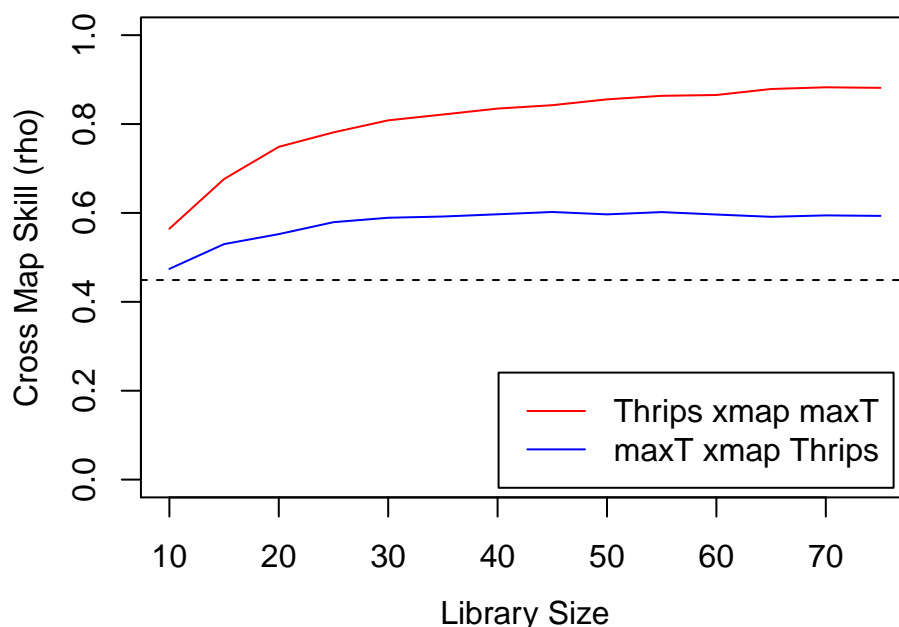

The magnitude of the cross-correlation is shown as a black dashed line for comparison. We repeat the analysis for rainfall.

```
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,
    silent = TRUE)

xmap_means <- list(ccm_means(thrips_xmap_Rain), ccm_means(Rain_xmap_thrips))
```
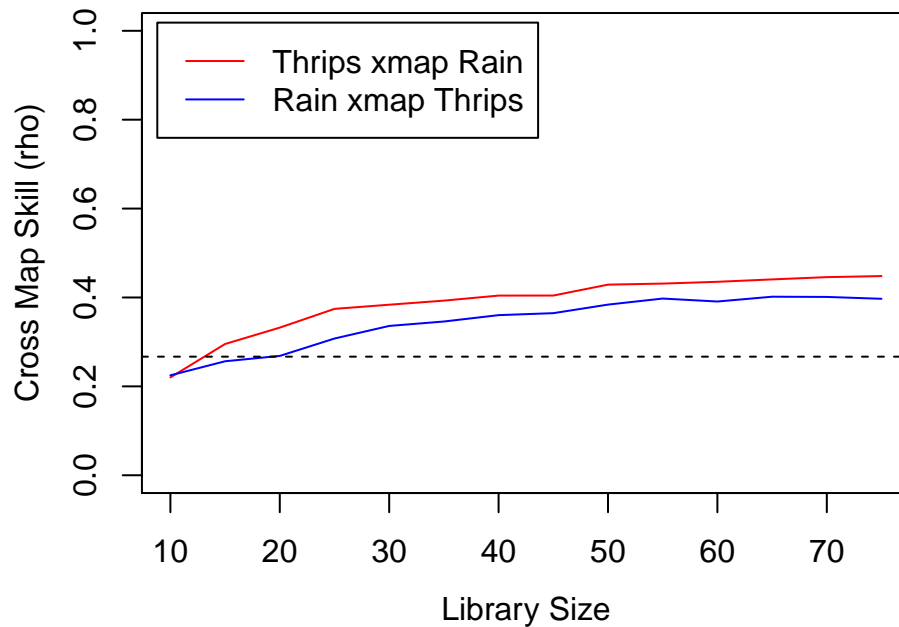

And finally for the deterministic seasonal cycle.

```
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)

xmap_means <- list(ccm_means(thrips_xmap_Season), ccm_means(Season_xmap_thrips))
```
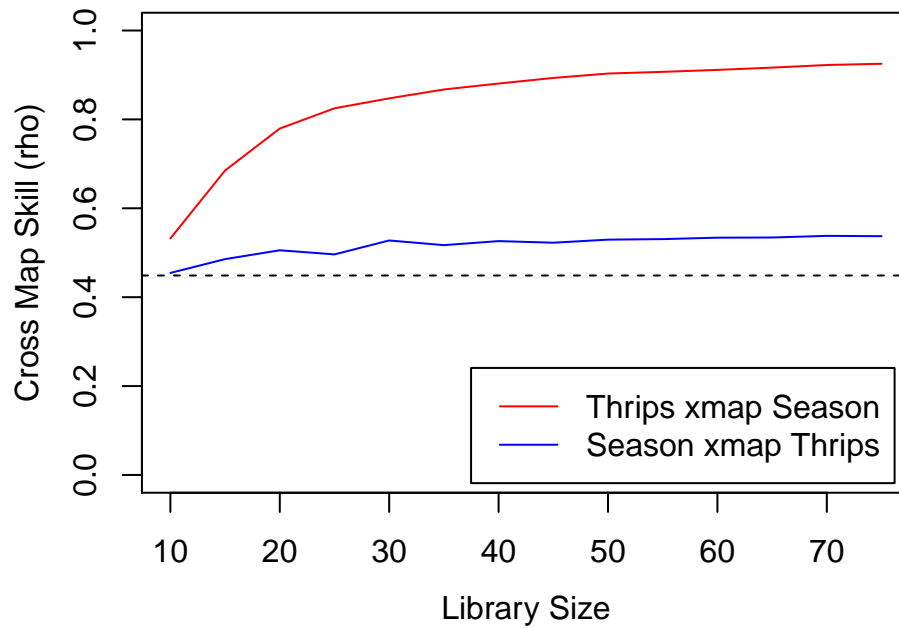
Importantly, the results show clear evidence of convergence for $Thrips$ cross-mapping the climactic variables, and the ρ at maximum L greatly exceeds the linear correlation.

However, we are still left with the conundrum that $maxT$ and to a lesser extent $Rain$ are easily predicted from the seasonal cycle, and so we cannot immediately ignore the possibility that $maxT$ and $Rain$ only show CCM with $Thrips$ because they share strong forcing.

To reframe, we would like to reject the null hypothesis that the causal effects we measure for $maxT$ and $Rain$ with CCM can be solely explained by their shared seasonality. This hypothesis is readily tested using surrogate methods.

```
num_surr <- 1000
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))

for (i in 1:num_surr) {
    rho_surr$maxT[i] <- ccm(cbind(thrips_block$Thrips_imaginis, surr_maxT[,
        i]), E = 8, lib_column = 1, target_column = 2, lib_sizes = NROW(thrips_block),
        replace = FALSE)$rho

    rho_surr$Rain[i] <- ccm(cbind(thrips_block$Thrips_imaginis, surr_Rain[,
        i]), E = 8, lib_column = 1, target_column = 2, lib_sizes = NROW(thrips_block),
        replace = FALSE)$rho
}
```

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

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

```
## [1] 0.05494505
```

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

```
## [1] 0.1748252
```

In both cases, the CCM we measure for the real time series are better than the median expectation under the null hypothesis. For rainfall, moreover, the effect is clearly significant based on the common threshold of $p < 0.05$. 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 $maxT$. We note that the original Thrips data collections were at a much higher frequency than those available through the GPDD, and that $maxT$ 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.

5 Technical Details

5.1 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.

5.2 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
  - (same format as lib, but specifes the portions of the time series to make predictions for)

- norm_type

    - `"L2 norm"` (default) or `"L1 norm"`: specifies which distance metric to use when doing calculations
    - `"L2 norm"` 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 $|x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n|$.

- 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

    - the prediction horizon (how many steps ahead to make forecasts)
    - negative values will also work

- num_neighbors

    - the number of neighbors to use
    - `"e+1"`, `"E+1"`, `"e + 1"`, and `"E + 1"` will all peg this parameter to be `E+1` for each run
    - values less than 1 will use all possible neighbors

- theta

    - the nonlinear tuning parameter (for use with S-maps) that adjusts how distance is factored into computation of the local linear map (`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

    - `TRUE` (default) or `FALSE`: specifies whether the output should just contain statistics of the predictions, or also contain all the predictions that were made

- exclusion_radius

    - `exclusion_radius` sets the threshold whereby all vectors with time indices too close to the predictee will be excluded from being considered nearest neighbors
    - e.g., `1` means that vectors must have an associated time index more than 1 away from potential nearest neighbors
    - by default, exclusion_radius is set to NULL (turning this filtering off)

- epsilon

    - `epsilon` sets the threshold whereby all vectors with distance too far away from the predictee will be excluded from being considered nearest neighbors
    - e.g., `2` means that vectors must have be within a distance of 2 from potential nearest neighbors
    - by default, epsilon is set to NULL (turning this filtering off)

- silent

    - `TRUE` or `FALSE` (default): specifies whether to suppress warning messages from being printed to the R console

- save_smap_coefficients

    - `TRUE` or `FALSE` (default): specifies whether to include a table of s-map coefficients with the output
    - (note that setting this to `TRUE` forces the full output as if `stats_only = FALSE`)

6   Acknowledgements

References

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

Davidson, J., and H. G. Andrewartha. 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, E. R., and G. Sugihara. 2011. Generalized theorems for nonlinear state space reconstruction. PLoS ONE 6:e18295.

Fargione, J. E., and D. Tilman. 2005. Diversity decreases invasion via both sampling and complementarity effects. Ecology Letters 8:604–611.

Sauer, T., J. A. Yorke, and M. Casdagli. 1991. Embedology. Journal of Statistical Physics 65:579–616.

Sugihara, G. 1994. Nonlinear forecasting for the classification of natural time series. Philosophical Transactions: Physical Sciences and Engineering 348:477–495.

Sugihara, G., and R. M. May. 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., J. Knops, D. Wedin, P. Reich, M. Ritchie, and E. Siemann. 1997. The influence of functional diversity and composition on ecosystem processes. Science 277:1300–1302.