Simplex projection walkthrough

Owen Petchey

5 July 2016

Contents

Introduction

Simplex projection is an important method for forecasting times series. The aim of this document is to explain how simplex projection work, in terms that are very easy to understand.

This document was created in Rmarkdown the Rmarkdown version on github.

This document is one from a collection of reproductions / explanations.

Why should you care about simplex projection?

One important use of forecasting via simplex projection is distinguishing chaotic time series from ones that are just random noise. This was invented by George Sugihara and Bob May back in 1990, and presented in the article Sugihara, G. & May, R.M. (1990) Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. Nature, 344, 734–741.

Simplex projection may be very useful for practical forecasting. A closely related method, S-map projection, can make more skillful forecasts of dynamics than even the correct / true mechanistic model (Perretti, C.T., Munch, S.B. & Sugihara, G. (2013) Model-free forecasting outperforms the correct mechanistic model for simulated and experimental data. Proceedings of the National Academy of Sciences of the United States of America, 110, 5253–5257.)

Explanation in the published articles

Published explanations are very clear, and require thinking about a time series in an embedded state space, about nearest neighbours in that state space, and about the projection of a simplex.

The Sugihara and May (1990) article states (the following is modified from the original) *The basic idea of simplex forecasting is that even for a chaotic time series, future values may be predicted from the behaviour of similar past values.*

From the supplementary information of another article: *Simplex projection is a nearest-neighbor forecasting algorithm that involves tracking the forward evolution of nearby points in an embedding (a lagged coordinate state space reconstruction). Thus, similar past events are used to forecast the future, with the important caveat that the dimensionality of the embedding determines what past events are similar (nearby) to the predictee.* Hsieh, C., Glaser, S.M., Lucas, A.J. & Sugihara, G. (2005) Distinguishing random environmental fluctuations from ecological catastrophes for the North Pacific Ocean. Nature, 435, 336–340.

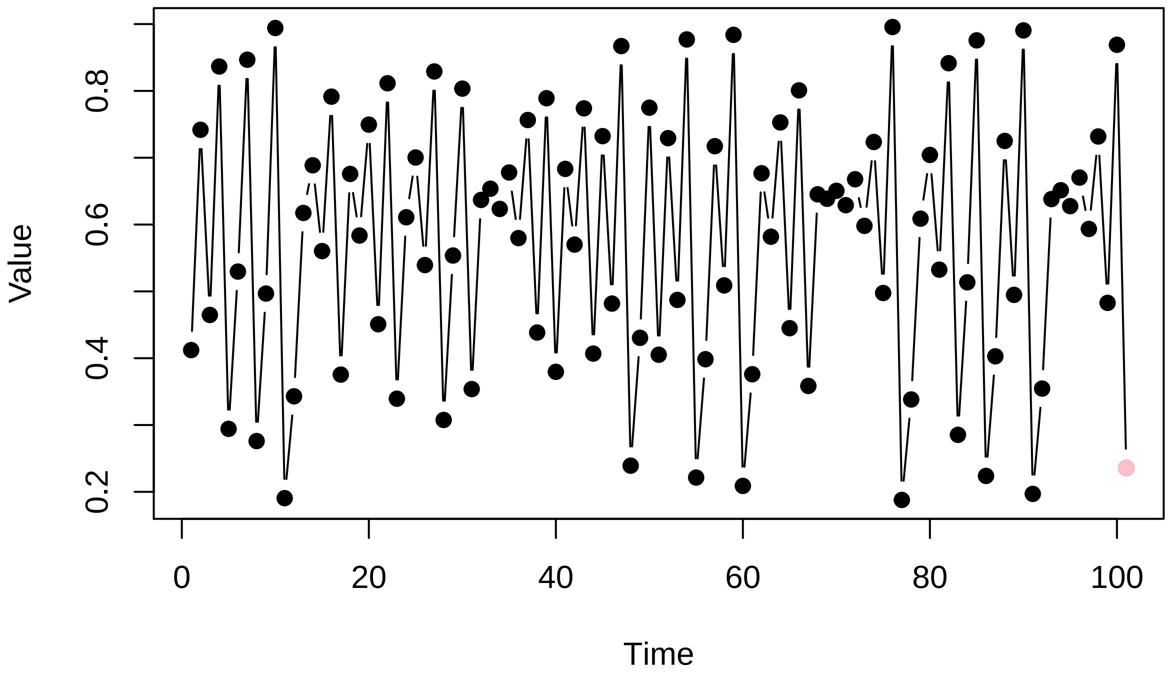
If you've read that, the full articles, and others, and still don't have an idea of what is simplex projection, read on. If you want to verify your understanding, read on. If you're all clear about simplex projection, but think others might not be and you want to help them, then you might read on to see an explanation from a different angle.

Explaining simplex projection

Creating a time series to explain with

As in Sugihara and May (1990) we create chaotic time series (here we use the same model as they did).

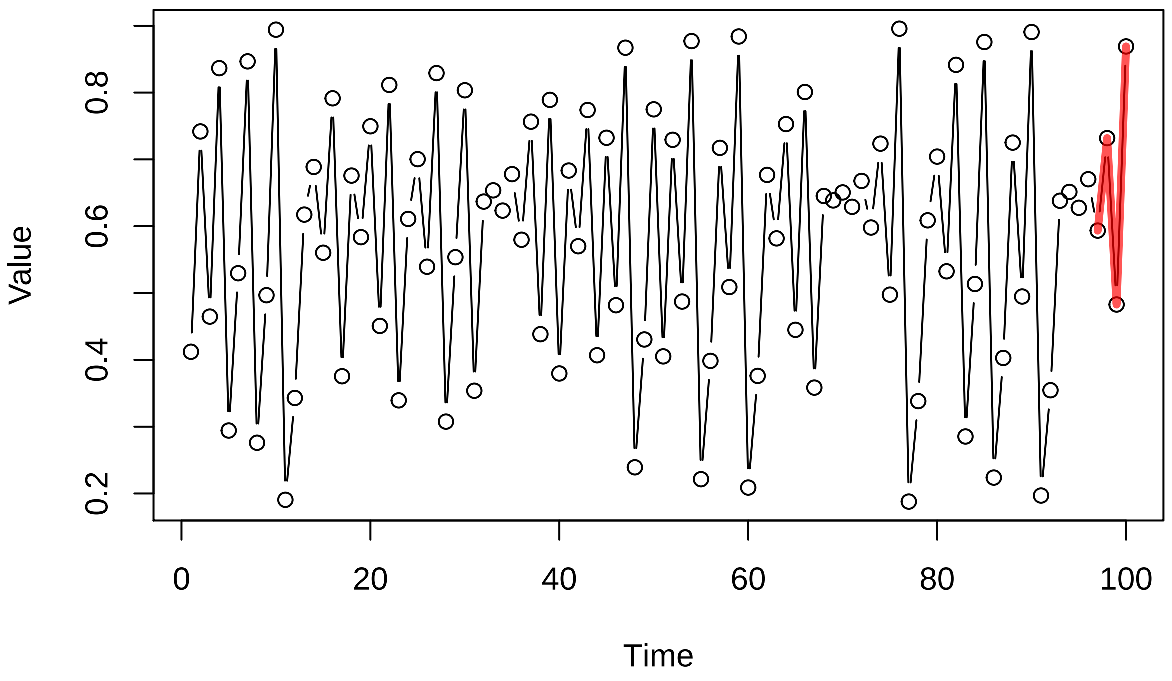
Here's the full time series we're going to work with. Our goal will be to use simplex projection to predict the last value in the time series (the pink one) from the previous values in the time series (the black ones).



What just happened?

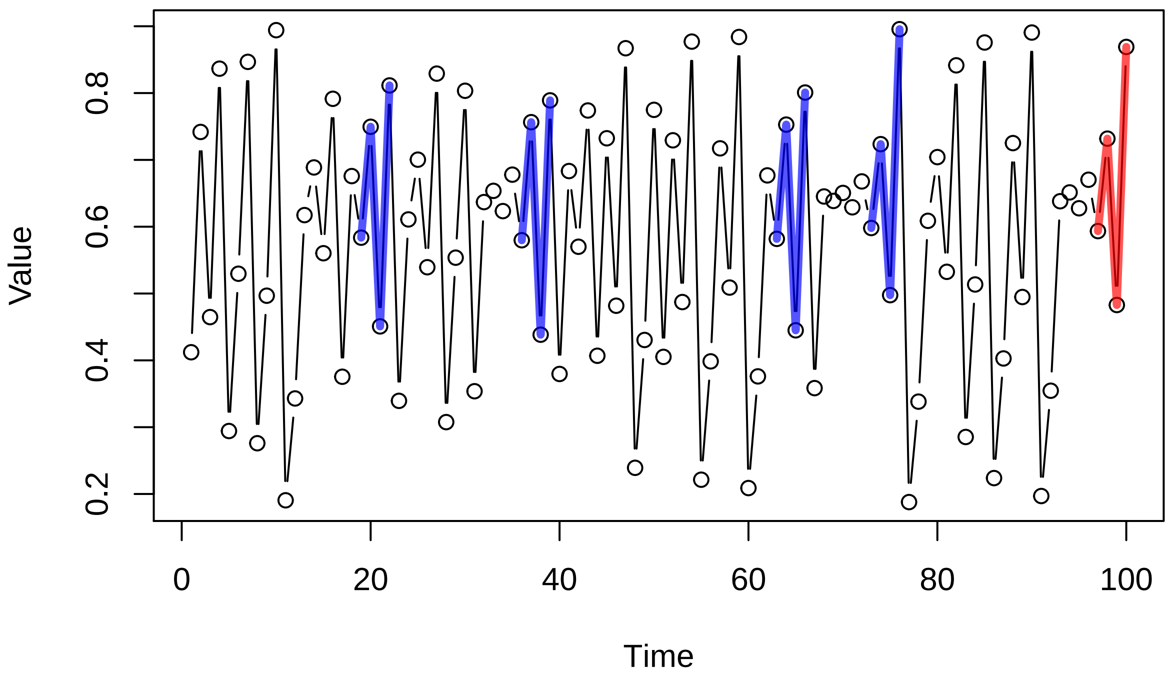
The first step of simplex projection is to find dynamics in the past, that match the dynamics that just happened. What do we mean by the *dynamics that just happened*?

In the following graph the dynamics of the final four data points in the times series are highlighted in red – this is the *dynamics that just happened*. (Note that the pink data point (the one we want to predict) has been removed.)



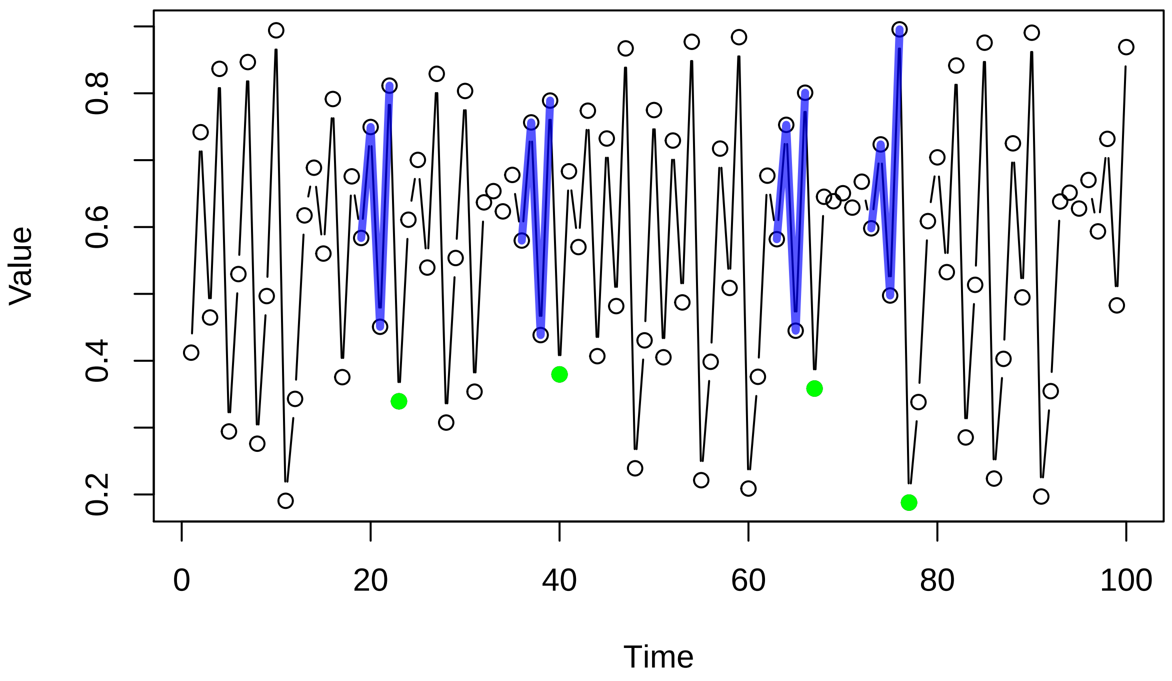
When did that happen before?

Next, we search the time series for close matches to what just happened (i.e. close matches to the red dynamics). The following graph shows the four closest matches highlighted in blue. You can see that the dynamics highlighted in blue are quite similar to the red dynamics.

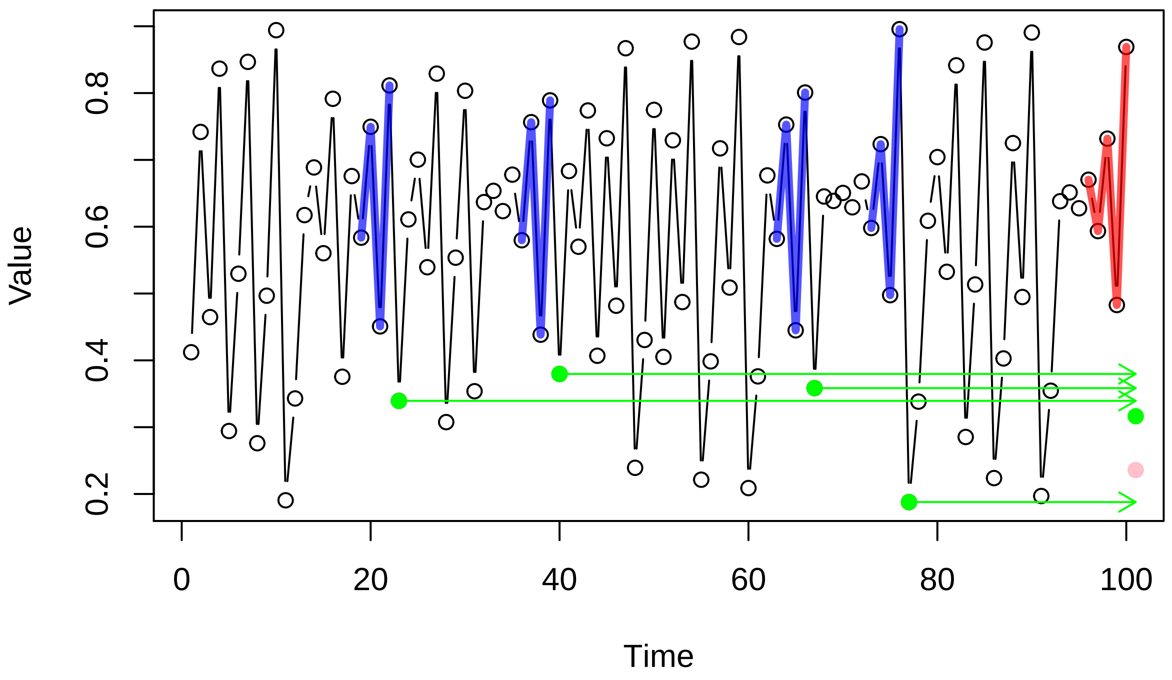


What is likely to happen next?

To make the prediction about what happens next (i.e., the value of the pink data point in the first graph), we use what happened just after the similar dynamics occured in the past. In the following graph, the green data points are the values that follow each of the closely matching past dynamics.



The following graph illustrates that it is these values that are used to project (predict) the next value in the time series. The green data point is the mean of these values. The pink data point is the actual value (the one in the first graph).



The predicted value (in green) is quite close to the actual value (in pink).

How is simplex projection used to distinguish chaotic dynamics from noise

Chaos produced by some deterministic process creates patterns in the time series. The patterns are exploited by simplex projection, and predictions are skillful. However, as one predicts further into the future, the prediction skill decreases. A time series of noise (a series of independent random numbers) does not show this decline in prediction skill with prediction interval. Hence, the prediction skill – prediction interval relationship can distinguish chaos from noise.

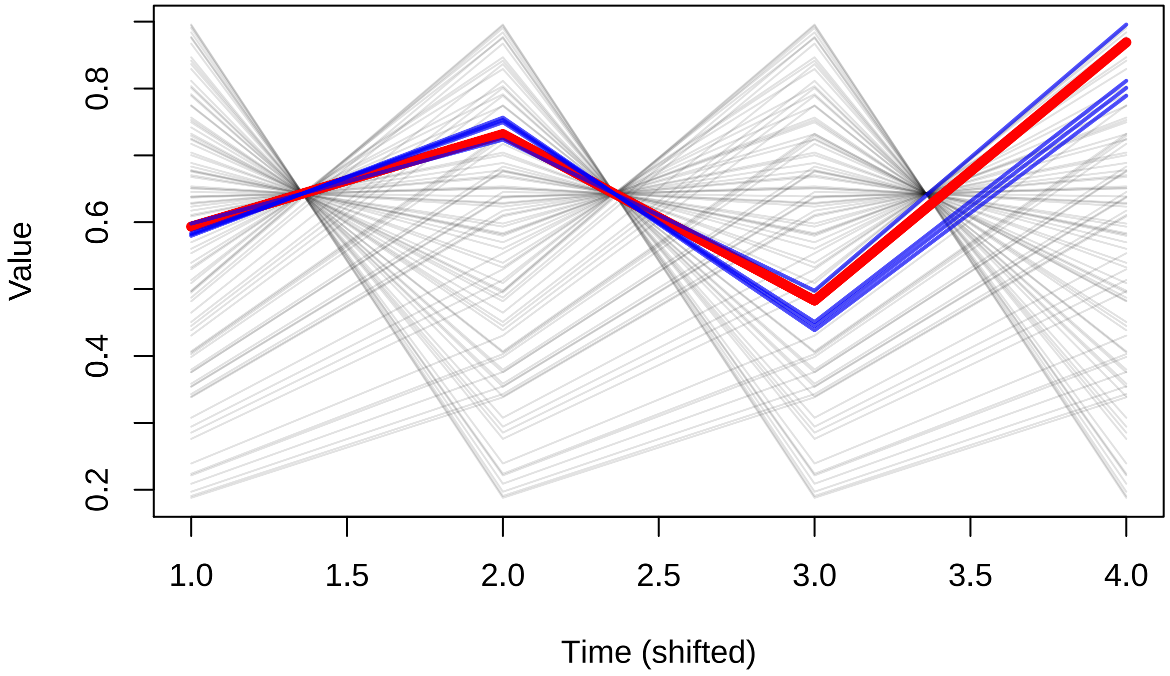
The details

There are lots of details just glossed over. For example:

- One needs to be careful about how many data points are used to characterise what just happened, and to find when this happened in the past. This is a decision about the *embedding dimension*. Here we used four.
- One needs to decide how many matches from the past will be used. In the terms of the published articles, this is the number of nearest neighbours. Here we used four (just by coincidence, the same as the embedding dimension).
- We calculated the means of the four values used to predict. However, usually one uses weighted means, where the weighting is according to how closely the past dynamics (blue) match the dynamics that just happened (red). Predictions are weighted more heavily when they are preceded by dynamics that more closely match.
- Autocorrelated noise can produce declines in prediction skill with increasing prediction interval, and thus appear to be chaotic. See Sugihara, G. & May, R.M. (1990) Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. Nature, 344, 734–741. for a suggestion of how to deal with this.

Another way of looking at it

The following graph shows the last four values in red, and in grey every other sequence of four consecutive values in the time series. These grey patterns are the *library* of past dynamics. We then search this library for dynamics that closely match the recent dynamics. The four closest matches are shown in blue; it is these that are used in the projection / forecasting.



Still another way of looking at it (for R users)

The published articles explain simplex projection in terms of embedded state space, time delay stated space, nearest neighbours, and so on. I think of embedded state space as a convenient method to represent the short segments of dynamics and to then look for matches among them. If you can read R, this might help / solidify your understanding.

In the example above, this is the code that does the embedding and finds the similarity among the short segments of dynamics. You'll see its quite simple.:

This line creates lagged copies of the original time series, and puts each lagged copy in the column of a matrix. Hence, the each row of this matrix `E_X` contains a short segment of the time series. (The `Embed()` function I wrote, and is at the very end of this document, in case you're interested.)

```r
E_X <- Embed(X_d1, E)
```

The next line gets the Euclidean distance between the last segment and all other pairs of segments. I.e., the dissimilarity between the last rown and all other rows. (It does this by calculating all possible distances, and the subsetting only those involving the last segment.)

```r
dists <- as.matrix(dist(E_X))[ts_length-E,]
```

The final line gets the location (time) of each segment, in order of increasing dissimilarity. I.e, the location of the most similar segment comes first, the next most similar second, and so on.

```r
times <- order(dists)
```

The function used to create the embedded state space. A bit boring…

```r
Embed <- function(X, dimension=3) {
  E_X <- matrix(NA, length(X)-dimension, dimension)
  for(i in 1:dimension)
    E_X[,i] <- X[(1+i-1):(length(X)-dimension+i-1)]
  E_X
}
```