QUANTIFYING DETERMINISM IN CORAL REEFS AND COASTAL ZONES

Nick Cortale

A Thesis Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Center for Marine Science

University of North Carolina Wilmington

2015

Approved by

Advisory Committee

Dylan McNamara

Jim Blum

Mark Lammers

Accepted by

Dean, Graduate School

TABLE OF CONTENTS

ABSTRACT

Natural systems are notoriously difficult to forecast. Ocean coastlines and coral reefs are both regions that draw considerable human attention as economic investment and infrastructure are threatened by a range of stressors (i.e., sea level rise and overfishing). Understanding the coastline's evolution over intermediate time scales (daily) or the species distribution on coral reefs in space (tens of meters) is hindered by the complexity of sediment transport and species interaction respectively. Modern remote sensing systems and newly developed underwater photographic techniques provide an efficient means to collect data within these regions. Methods in nonlinear time series forecasting, adjusted here for spatial data, applied to these systems corroborate that both coastal morphology and species distribution are predominately driven by nonlinear internal dynamics. Results also indicate that these forecasting techniques are capable of elucidating the relative deterministic nature of each of the systems. A metric is developed to succinctly summarize the results of the non-linear forecasting. The metric is applied before and after an engineered beach nourishment as well as along a gradient of coral reef health. The metric indicates that beach evolution subsequent to a nourishment is dominated by nonlinear dynamics relative to the pre-nourished behavior and degraded coral reefs show more evidence of random spatial dynamics than pristine reefs.

ACKNOWLEDGMENTS

I would like to thank the Gordon and Betty Moore Foundation for their financial support of this project and my schooling. I would also like to thank the Scripps Institution of Oceanography for the photomosaics of the coral reefs that were analyzed in this thesis. I also wish to thank the staff at the Center for Marine Science who have been been enormously helpful throughout my graduate career. Additionally, I thank my committee members Dr. Jim Blum and Dr. Mark Lammers for their support, useful critiques of my work, and sitting through my numerous presentations. I am also grateful to Dr. Kenneth Ells who provided valuable feedback on this work and pushed me to write better code. Finally, my sincere thanks goes to my advisor of six years, Dr. Dylan McNamara, who provided me with an enormous academic opportunity that was almost too good to be true and for that I will always be grateful. Additionally, his mentorship both academically and non-academically has been invaluable throughout my graduate career. I truly enjoyed my time working in the Complex Adaptive Systems Lab.

1 INTRODUCTION

Emergence of large-scale coherence and patterning in systems of many interacting constituents is a hallmark of complex systems, in which feedbacks and nonlinear internal dynamics dominate evolution [10]. The extent to which evolution of the system is controlled by internal nonlinear dynamics, as opposed to responding primarily to the noisy forcing environment, is difficult to quantify. Most numerical models that simulate system dynamics rely heavily on detailed forcing information to make forecasts and often only account for weakly nonlinear intrinsic dynamics. In contrast, the methods presented here attempt to forecast based solely on previous states in both space and time without direct knowledge of concurrent forcing or model equations of system dynamics. The efficacy of these predictions does, however, provide insight about the dynamics that dominate the evolution of the system.

Techniques that model the evolution of macroscopic features, like shoreline position or species distribution, using process-based approaches that ramp up granular physics or detailed chemical and physical species level dynamics via explicit parameterizations are well established [3] [27]. However, these techniques have suboptimal forecast performance and site adaptability. Internal nonlinear dynamics have also been shown to influence a system's response to forcing, where behavior may not simply be connected to forcing signatures in time. For example, sediment transport systems can exhibit behavior that masks supply signals within certain ranges of amplitude, outside of which the forcing appears to overbear these dynamics [17]. These examples and others illustrate the importance of nonlinear internal dynamics in contributing to the evolution of natural phenomena without a need for knowledge of fast-time scale dynamics or detailed forcing features.

The relative position of the ocean coastline is an important resource, as it dictates the usable recreational beach and influences property values [14]. Despite its importance to static coastal communities, the region is highly dynamic, varying with tidal elevation, wave set-up, and storm-surge. The magnitude of change in coastline position is regulated by the local slope of the shore-face, the profile of which is not typically linear [2]. Characteristics

(shape) of the intertidal beach and surf-zone profile are known to adjust in response to changing environmental conditions, and adjustments are not necessarily uniform along the profile [7] [35]. Interestingly, despite the myriad of hydrodynamic forcings and sediment compositions, sandy coastlines exhibit a limited range of morphological modes.

Likewise, coral reefs provide valuable economic value to their adjacent communities through tourism and food provision [5]. These valuable coral reefs, however, are susceptible to drastic changes due to human induced activities such as fishing, pollution, and climate change [15]. In the most extreme examples of these disturbances, coral reefs can systematically transition to a benthos covered entirely in algae species [15]. Similar to coastlines then, despite a myriad of complicated chemical, biological, and physical interactions at short time scales, coral reefs appear to exhibit a small range of benthic configurations, either healthy with coral species covering the benthos or disturbed with algae out-competing all species for space [8].

As a study domain, with the advent of remote imaging systems and new underwater photographing techniques, the beach and coral environments are no longer data poor [31]. Comprehensive observations and the development of techniques to extract rigorous, quantifiable features from these sources have led to the development of data driven modeling techniques and forecasting [30]. Time series obtained from imaging systems contain information about internal dynamics as they manifest in the systems evolution [26]. According to Takens Theorem, given sufficient data, a deterministic systems phase space trajectory is reproducible and system evolution may be forecast [28] [1]. Specifically, forecasting is based on neighbor trajectories within the embedded phase space, and skill can be affected by the choice of embedding dimension, weighting of neighbor trajectories, the amount of noise in the data, and the prevalence of nonlinear interactions [29] [33].

These nonlinear forecasting techniques have proven capable of outperforming mechanistic models in noisy, non-linear ecological systems, and they have shown the ability to distinguish noisy natural time series from those governed by nonlinear dynamics [32] [22].

Here, techniques in nonlinear time series analysis and forecasting are used to explore the extent to which local nonlinear interactions affect day-to-day intertidal profile adjustments and species distribution on coral reefs. Additionally, a metric is created to quantify the relative role of deterministic non-linear interactions and random behavior.

2 METHODS

2.1 THE LORENZ SYSTEM

The well-known Lorenz equations will be used as an initial illustration of the nonlinear forecasting technique. The Lorenz equations comprise a coupled dynamical system that represent a simple model of two-dimensional fluid flow when forced by a temperature gradient [20]. Formally, the Lorenz equations are given by

$$\frac{dx}{dt} = \sigma(y - x),$$

$$\frac{dy}{dt} = x(\rho - z) - y,$$

$$\frac{dz}{dt} = xy - \beta z,$$

where x is related to the rotational speed of the flow, y is proportional to temperature differences between rising and sinking flows, z is the deviation of temperature from the mean, σ is the fluid viscosity, ρ is the Rayleigh number, and β is the ratio of width to height of the fluid. The numerical solutions to these equations with $\rho = 28$, $\sigma = 10$, and $\beta = \frac{8}{3}$ produce time series (Figure 1) with classic hallmarks of deterministic chaos, namely sensitivity to initial conditions and saturated spectrums. Sensitivity to initial conditions is illustrated here by solving the system of equations, having started from slightly different initial conditions, and plotting the difference in system evolution for each dynamical variable (Figure 2). The difference between the two solutions is initially small, but the solutions exponentially diverge from each other and the initially small variation quickly approaches the bounding size of the system.

A phase space is a space that displays all possible configurations of a system [18], where the axes of the space represent the variables describing the system. Plotting X vs. Y vs. Z for the Lorenz system shows a single trajectory in the phase space started from the initial condition $[X_0, Y_0, Z_0]$ (Figure 3). Importantly, although trajectories started from different

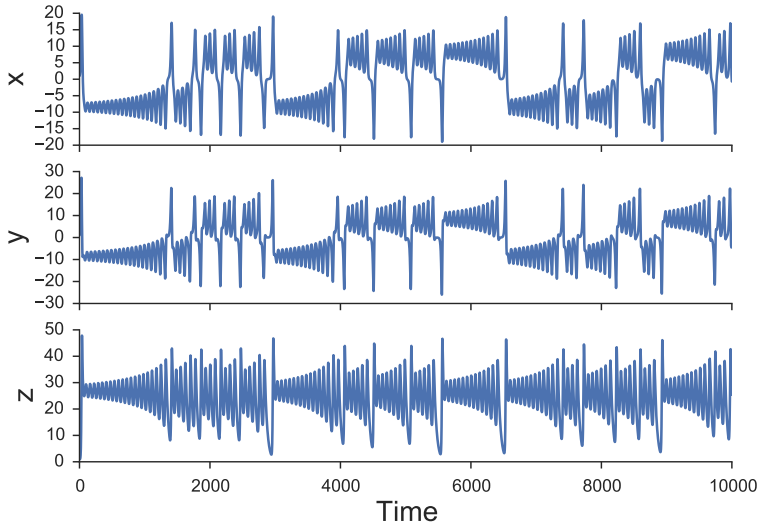

Figure 1: Lorenz system solved with parameters:$\rho = 28$, $\sigma = 10$, and $\beta = \frac{8}{3}$.

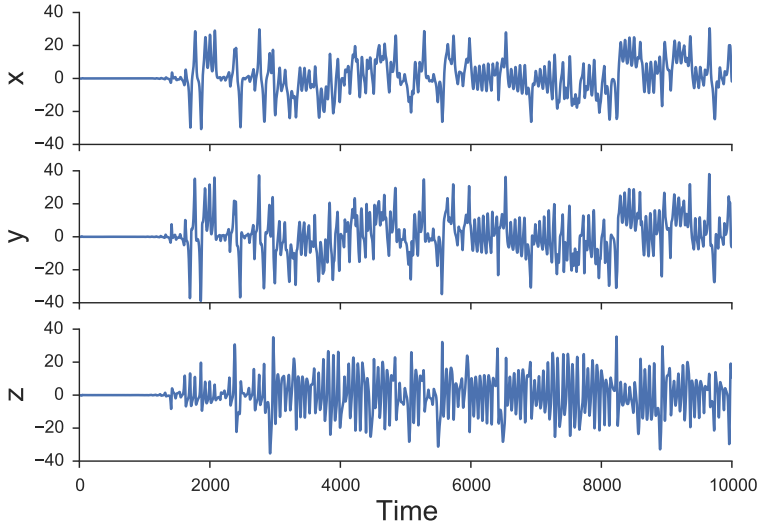

Figure 2: Difference between two Lorenz systems solved with parameters:$\rho = 28$, $\sigma = 10$, and $\beta = \frac{8}{3}$, but started with initial conditions $[X_0 = 1.01, Y_0 = 1.01, Z_0 = 1.01]$ and $[X_0 = 1.00, Y_0 = 1.00, Z_0 = 1.00]$.

initial conditions diverge, all trajectories for this system remain confined to a subset of the phase space, termed the attractor. Additionally, trajectories in the phase space can never intersect. Intersection of trajectories would indicate that the system is not deterministic and that a given point in the phase space could evolve in two separate ways.

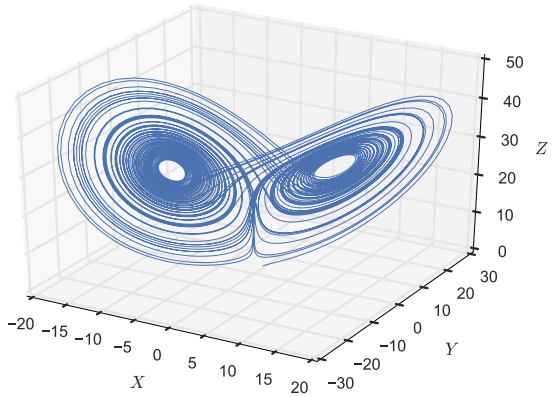

Figure 3: A single trajectory in the Lorenz phase space solved with parameters:$\rho = 28$, $\sigma = 10$, and $\beta = \frac{8}{3}$ and started from $[X_0 = 1.00, Y_0 = 1.00, Z_0 = 1.00]$.

2.2 RECONSTRUCTED STATE SPACE

In most natural systems, the underlying dynamical equations that govern a system are unknown. Often, insight is derived empirically by taking measurements of a single variable. As a proxy for this scenario, imagine having taken measurements of the Lorenz system by only capturing the variable X. Florence Taken has shown [34] that time lagged values of a single dynamical variable within a multi-dimensional system can be used to reconstruct the basic features of the attractor, a technique termed state space reconstruction. Specifically, although the reconstructed version of the attractor is not identical in appearance, it maintains the dynamically important features of the original attractor such as topology, sensitivity to initial conditions (as measured by the Lyaponov exponent), and attractor dimension. Thus, the reconstructed attractor illuminates the dynamics, or flows within the phase space,

through measurements of a single variable. The reconstruction is done by forming vectors, $\vec{V_t}$, in m dimensional space. These vectors provide a means of embedding the one dimensional time series into a multi-dimensional space. Let the embedding vectors be given by

$$\vec{V_t}(X) = (X_t, X_{t-\tau}, \ldots X_{t-(m-1)\tau}).$$

Here τ represents a fixed lag in time and can be found by calculating the first minimum of the mutual information [9]. The embedding dimension can be found by using the false near neighbor test [1], however when using this technique on natural time series, the dimension used is more commonly the one which results in the highest forecast skill [19]. For the Lorenz system, the false near neighbor test yields an embedding dimension of three and embedding the X time series in a three dimensional space provides the highest forecast skill. The reconstructed state space is shown in Figure 4.

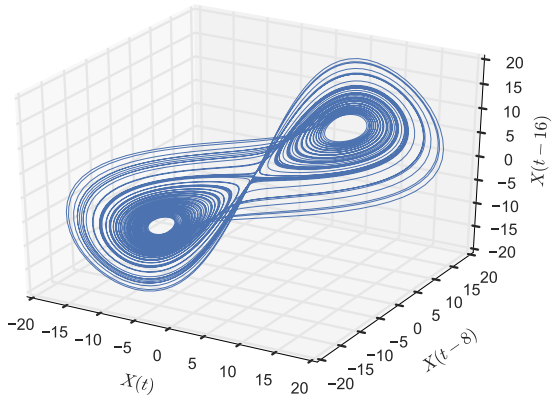

Figure 4: Reconstructed state space using X values from the Lorenz system.

The attractor in the reconstructed space qualitatively and quantitatively preserves the same shape as the original Lorenz attractor. Trajectories are smooth and do not cross in the reconstructed space preserving the deterministic nature of the system.

2.3 FORECASTING

In order to forecast X_{t+n}, where n is the forecast distance, the reconstructed state space is used. Given a point $\vec{V}_t(X)$, in the embedded space, the K nearest points (near neighbors) are calculated using the euclidean distance. The trajectories of the K nearest neighbors in the embedded space are then averaged and compared to the actual evolution of the point in question. For the purpose of distinguishing system dynamics, forecasts are made over a range of near neighbors and a range of forecast distances [28]. In order to quantify the accuracy of the forecasts, the coefficient of determination, R^2, is calculated as:

$$R^2 = 1 - \frac{\sum(y_i - f_i)^2}{\sum(y_i - \bar{y})^2},$$

where y_i is the true value, f_i is the forecast value, and $\bar{y}$ is the mean of the true values. The results of this forecasting technique when applied to the Lorenz system are shown in Figure 5.

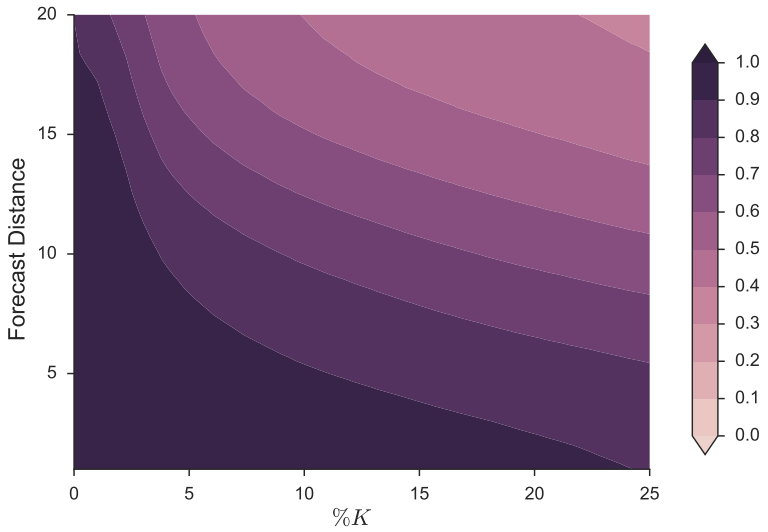

Figure 5: The coefficient of determination (R^2) plotted against near neighbors (K) and forecast distance.

Figure 5 captures the Lorenz system's sensitivity to initial conditions as evident by the decreasing R^2 value with increasing forecast distance. Deterministic nonlinearity is evident

by the decrease of forecast skill when increasing the number of near neighbors used in the forecast. This trend reflects the importance of localized flow dynamics within the phase space as trajectories that are farther away from a test forecasting point are less accurate representations of future behavior.

2.4 EVALUATING DETERMINISM

In order to evaluate the amount of stochasticity in a system, increasing amplitudes of uncorrelated noise are added to the embedded Lorenz x values as,

$$x_n = x + \alpha\eta,$$

where x_n is the Lorenz x values with added noise, α is the amplitude of the noise, and η is uncorrelated noise having values between zero and one. The noise complicates trajectories in the state space as the original smooth trajectories become jagged and overlapping. An example of a reconstructed state for x_n is shown in Figure 6. The non-linear forecasting technique is applied and the results are shown in Figure 7.

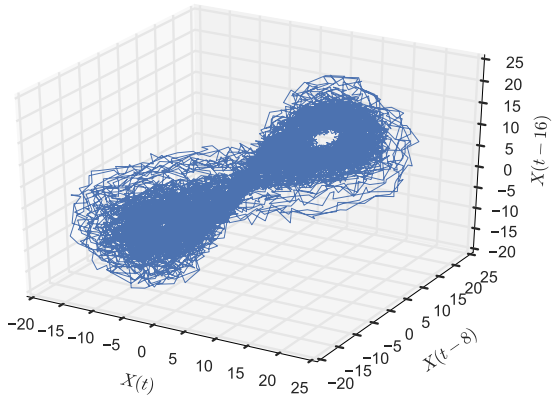

Figure 6: The embedded x time series from the Lorenz system with uncorrelated noise added to the time series.

A simple metric is created that attempts to calculate the relative contributions of deter-

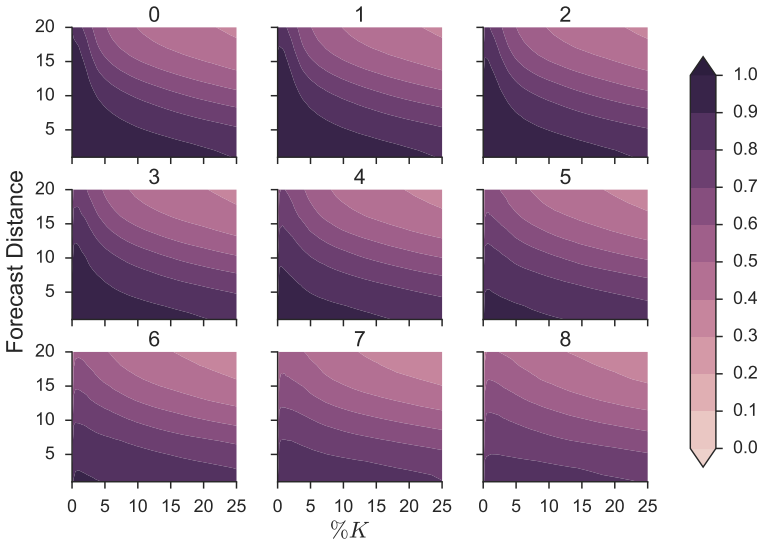

Figure 7: The coefficient of determination (R^2) plotted against near neighbors (K) and forecast distance. Title corresponds to the amplitude of the noise, α.

minism and randomness in a system. In order to apply the metric, two assumptions must be made. First, the systems being compared must have the same underlying dynamics. For example, comparing the amount of determinism in a coastal formation to the amount of determinism in a biological pattern would not be valid. Conversely, comparing the amount of determinism across different biological patterns within the same system or comparing the amount of determinism across different coastal formations could provide insight. The second assumption is that the phase space is well populated within the system attractor.

The goal of the metric is to collapse the contours in Figure 7 into a single value. To capture the high forecast skill at low K for deterministic systems, the metric is represented as:

$$D = \sum_{d=1}^{n} max(R^2_{d,\%K<12.5}) - min(R^2_{d,\%K\geq 12.5}),$$

where D is the deterministic metric, d is the forecast distance, n is equal to one-half of the mutual information of the system, $\%K$ is the percent of near neighbors, and R^2 is the coefficient of determination. For each forecast distance, the metric calculates the maximum

R^2 value at low numbers of near neighbors and subtracts the minimum R^2 value at high near neighbors. It then sums up the differences out to a forecast distance that is one-half of the mutual information of the system. The metric is run on the nine contour plots in Figure 7 and the results are shown in Figure 8.

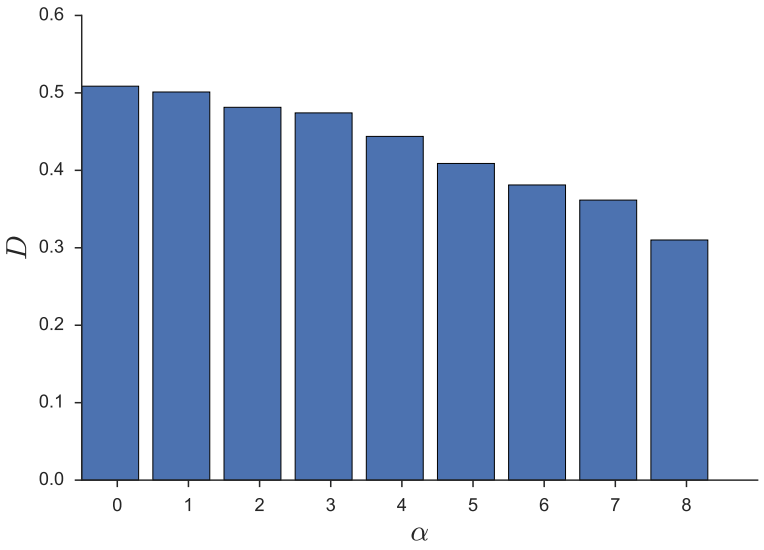

Figure 8: The deterministic metric calculated from the Lorenz system. The x-axis corresponds to the amplitude of the noise, α.

The larger the relative role of determinism versus noise,the larger the value of D. The metric captures the fact that stochastic systems benefit from averaging higher amounts of near neighbor trajectories in forecasting system behavior while systems with stronger deterministic dynamics benefit from averaging trajectories near in the reconstructed state space.

2.5 ONE DIMENSIONAL SYSTEMS

In order to further illustrate the non-linear forecasting technique and the deterministic metric, two different one dimensional systems with different underlying dynamics are analyzed under verying amplitudes of noise. The first is the logistic map,

$$X_{t+1} = rX_t(1 - X_t) + \alpha\eta,$$

where r is a parameter of the system, α is the amplitude of the noise, and η is uncorrelated noise with values between 0 and 1. For certain values of r and zero noise ($\alpha = 0$), the map produces deterministic chaos [21]. The addition of varying amplitudes of noise (Figure 9) does not seem to change the qualitative appearance of the resulting time series.

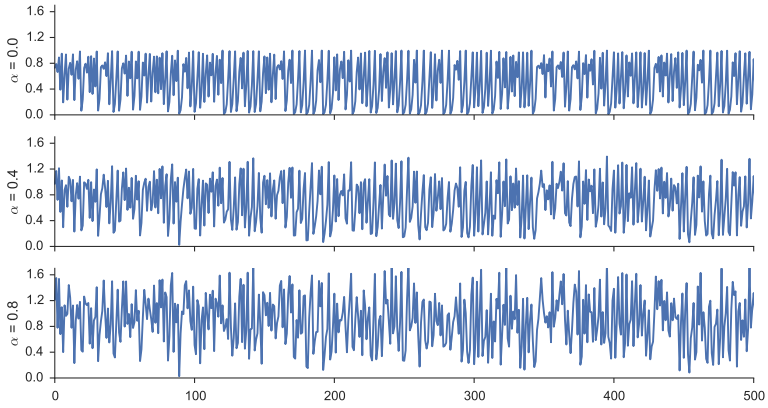

Figure 9: Time series from the logistic map with different amplitudes of noise as indicated by α.

Next a periodic function with noise added,

$$X(t) = sin(t) + \frac{1}{2}cos(t) + \frac{1}{4}sin(\frac{1}{4}t) + \alpha\eta,$$

is used to generate a number of different time series with various amplitudes of noise for analysis (Figure 10).

The non-linear forecasting technique is applied to both systems (Figure 11) and (Figure 12) for nine different amplitudes of noise added to each system.

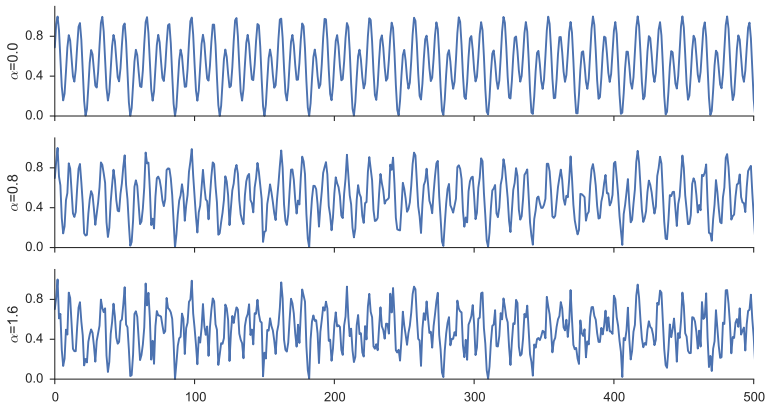

Figure 10: Time series resulting from the periodic equation with different amplitudes of noise as indicated by α.

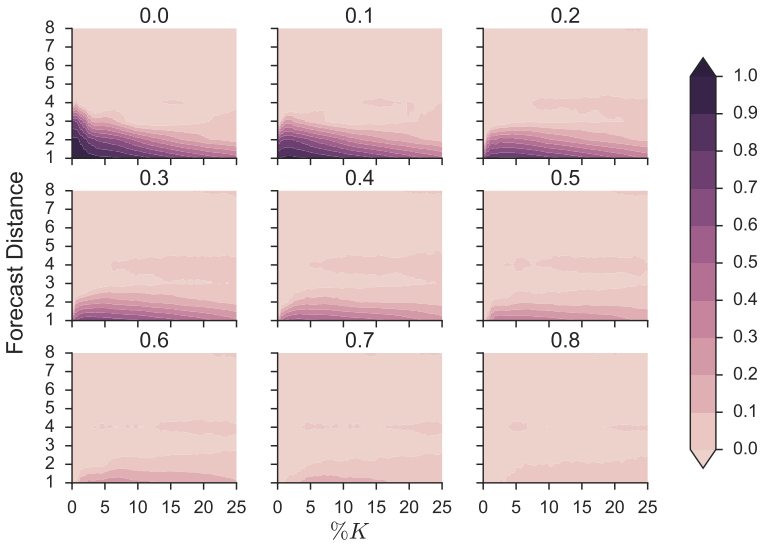

Figure 11: Non-linear forecasting results for the logistic map with different amplitudes of noise. The coefficient of determination, R^2, is plotted against near neighbors (K) and forecast distance. Title corresponds to the amplitude of the noise, α.

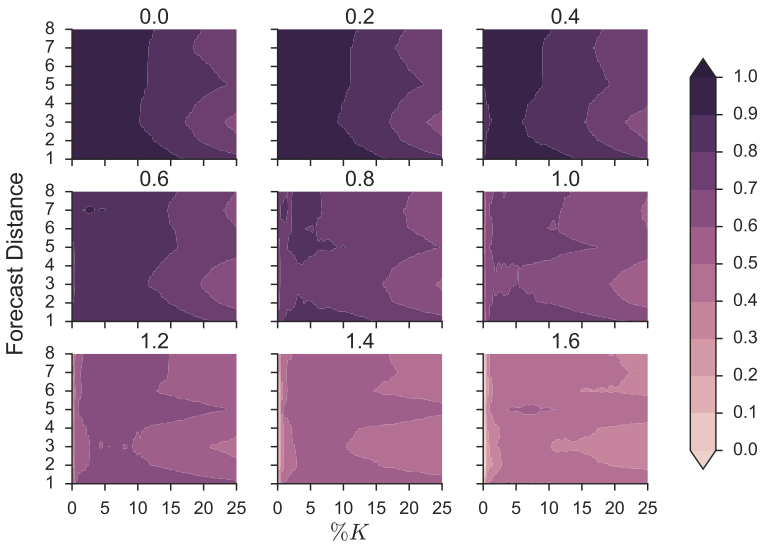

Figure 12: Non-linear forecasting results for the periodic equation with different amplitudes of noise. The coefficient of determination, R^2, is plotted against near neighbors (K) and forecast distance. Title corresponds to the amplitude of the noise, α.

The results from the non-linear forecasting technique illustrate the different underlying dynamics of each system. For the periodic equation, the R^2 values do not decrease with increasing forecasting distance and they do not decrease as rapidly as the logistic map with increasing near neighbors used. The system simply repeats its behavior and thus forecasting is trivial. Moreover, the reconstructed state space for the periodic equation is geometrically less complex than the logistic map (Figure 13). This allows a greater percent of near neighbor trajectories to be averaged before R^2 values start to decrease.

The analysis of the logistic map time series displays the same trends as the forecasting results from the Lorenz time series. High R^2 values at low values of K and low forecast distance reveals the system's locality of phase space dynamics and sensitivity to initial conditions.

The deterministic metric is calculated for both the logistic and periodic systems and the results are shown in Figure 14 and Figure 15 respectively.

The deterministic metric is able to distinguish the relative level of noise in the logistic map time series. This is again because the metric captures the benefit to using local neighbors in the reconstructed phase space and as noise is added to the system, the importance of

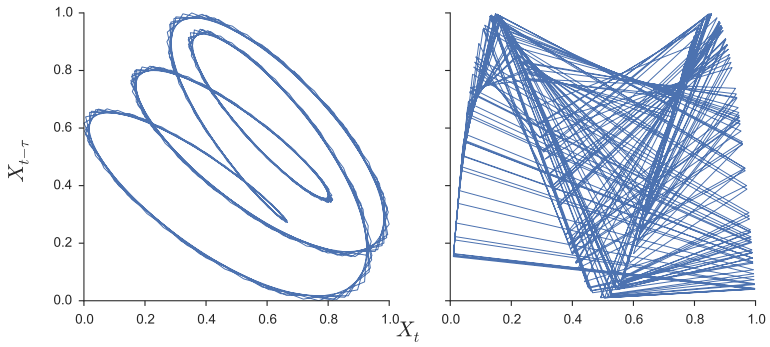

Figure 13: Two-dimensional reconstructions of the phase space for the periodic equation (left panel) and the logistic map (right panel) with lag values found from the first minimum in the mutual information.

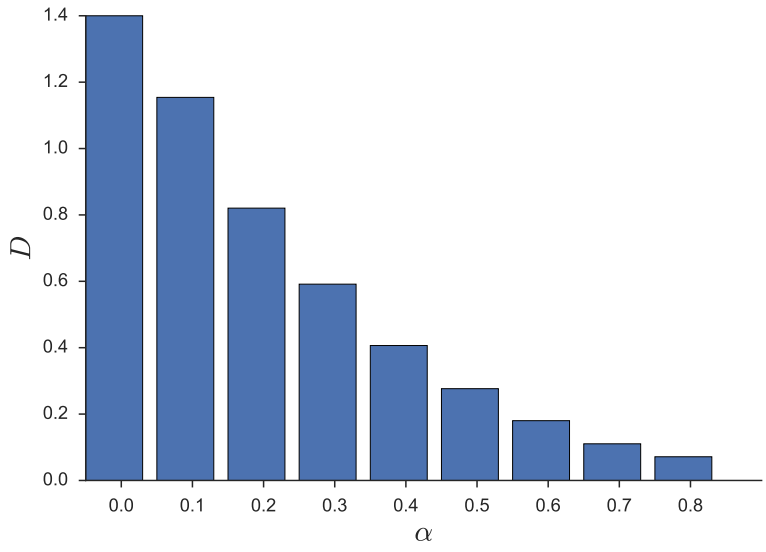

Figure 14: Deterministic metric for the logistic map with varying amplitudes, α of added noise.

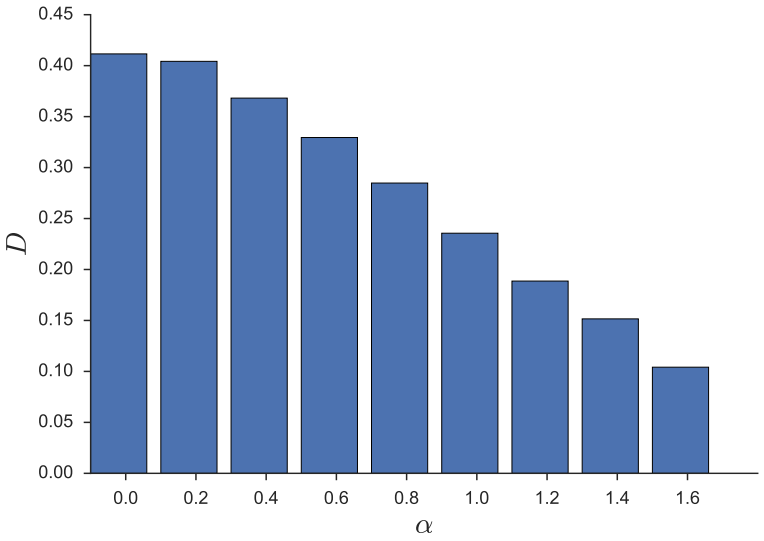

Figure 15: Deterministic metric for the periodic equation with varying amplitudes, α of added noise.

localized phase space dynamics is reduced. Additionally, when sufficiently high amounts of noise are present in the system(e.g., the logistic map with noise levels between 0.7-0.9) it becomes difficult to distinguish the relative determinism. The overall low forecast skill masks determinism in the system's state space. For the periodic system, since there is minimal benefit to using localized neighbors in forecasting even with low levels of noise, the deterministic metric shows a smaller reduction as noise is added to the system. Hence, the metric is capturing determinism that results from localized, or nonlinear, dynamics in the phase space.

2.6 TWO-DIMENSIONAL SYSTEMS

In this section, the non-linear forecasting technique is expanded to series in two dimensions: space (x) and time (y). In order to rebuild the state space, lagged values in both space, τ_c, and time, τ_r, are used [28] [22], where again lags are found using the first minimum in the average mutual information. The technique of constructing an embedding vector is illustrated in Figure 16. The top half of the image (training) is used to reconstruct the state space and forecasts are made for the bottom half of the image (testing). In this case forecasts are made along the time axis. The forecasts are compared to the actual evolution of the system and the deterministic metric is calculated as before.

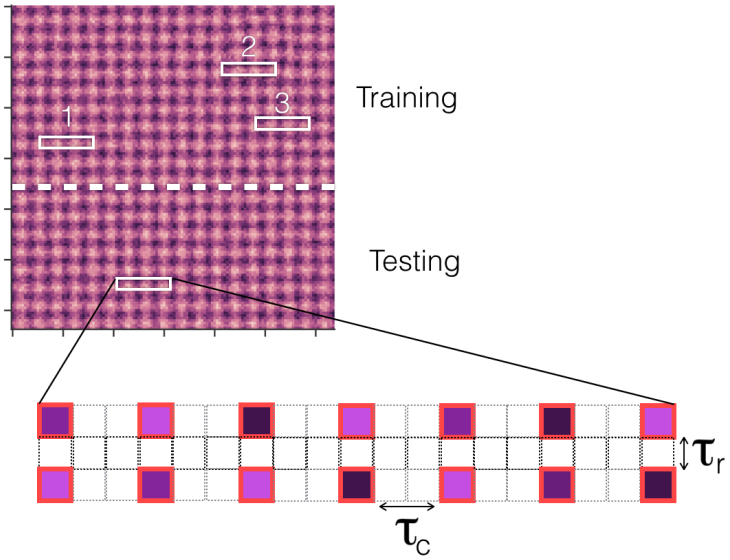

Figure 16: Two-dimensional embedding technique. τ_c is the lag value of the columns. τ_r is lag value of the rows. The image shows an example of a fourteen-dimensional embedding. The three nearest neighbors in the training set are shown.

To test the technique, two-dimensional analogs of the one-dimensional series are created and analyzed. The first image (Figure 17) is a spatio-temporal logistic map with uncorrelated noise and is defined as:

$$x_{t+1} = Ax_t(1 - x_t) \equiv f(x_t)$$

$$x_{t+1,s} = \frac{1}{1+4\epsilon}[f(x_{t,s}) + \epsilon f(x_{t,s\pm1}) + \epsilon f(x_{t,s\pm2})] + \alpha\eta.$$

Where x is the series being generated, A is a constant, f is the logistic function, ϵ parameterizes the spatial coupling, α is the amplitude of the noise, and η is uncorrelated noise between zero and one. The second image is a periodic function in space added to a periodic function in time with uncorrelated noise added(Figure 18). The noise amplitude, *alpha*, is increased incrementally for both images and the trends of the R^2 values are analyzed over a range of near neighbors and forecast distances.

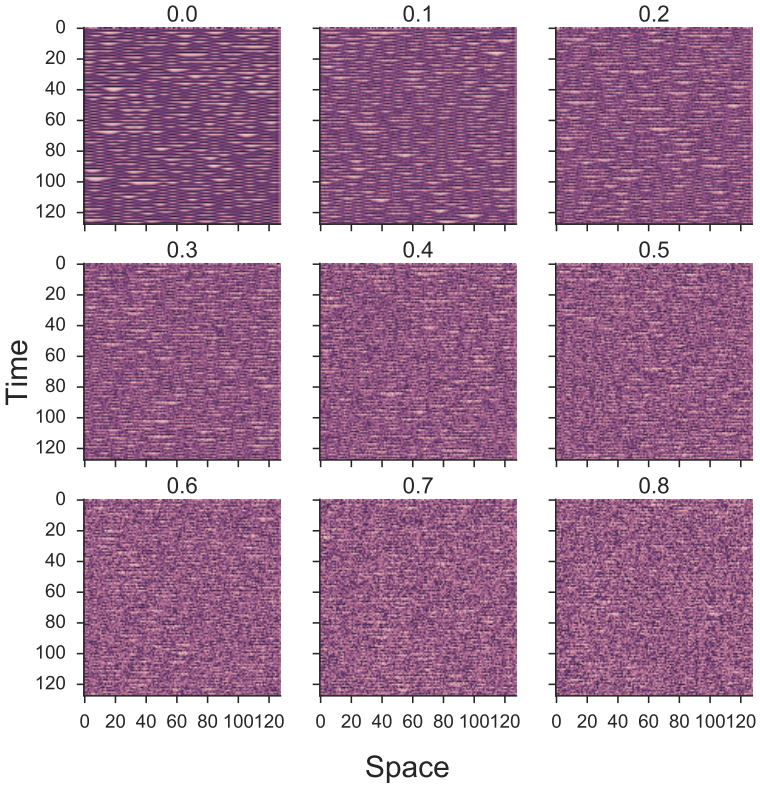

Figure 17: Two-dimensional logistic map. The number on the plot indicates the amplitude of the added noise, α.

The spatio-temporal non-linear forecasting technique is applied to each of the generated images. The results for the spatio-temporal chaotic image and spatio-temporal periodic image are shown in Figure 19 and Figure 20 respectively.

For both systems, the results of using the non-linear forecasting analysis are similar to the

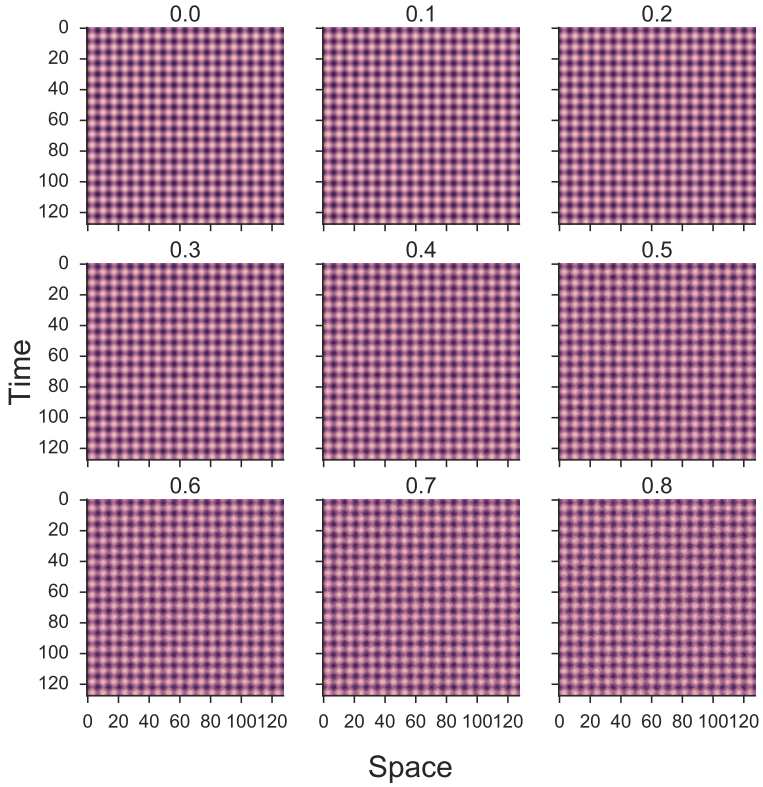

Figure 18: Two-dimensional periodic image. The number on the plot indicates the amplitude of the added noise, α.

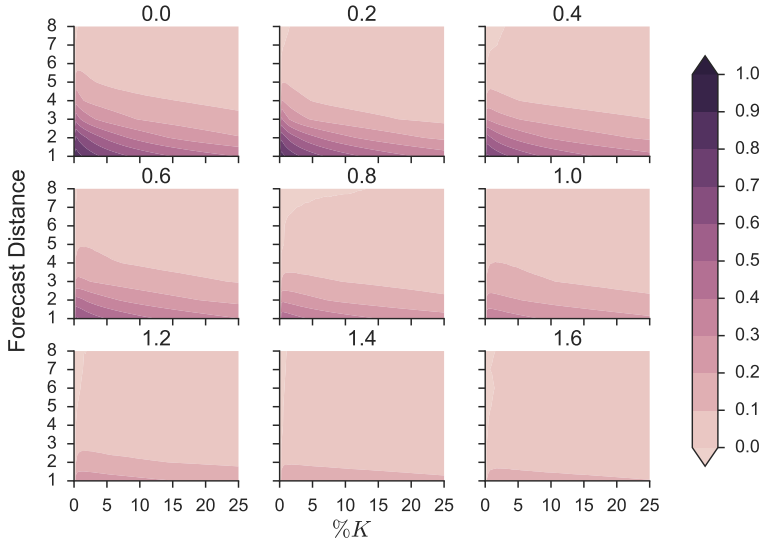

Figure 19: Non-linear forecasting results for the spatio-temporal logistic map with different amplitudes of noise. The coefficient of determination, R^2, is plotted against near neighbors (K) and forecast distance. The number on the plot indicates the amplitude of the noise, α.

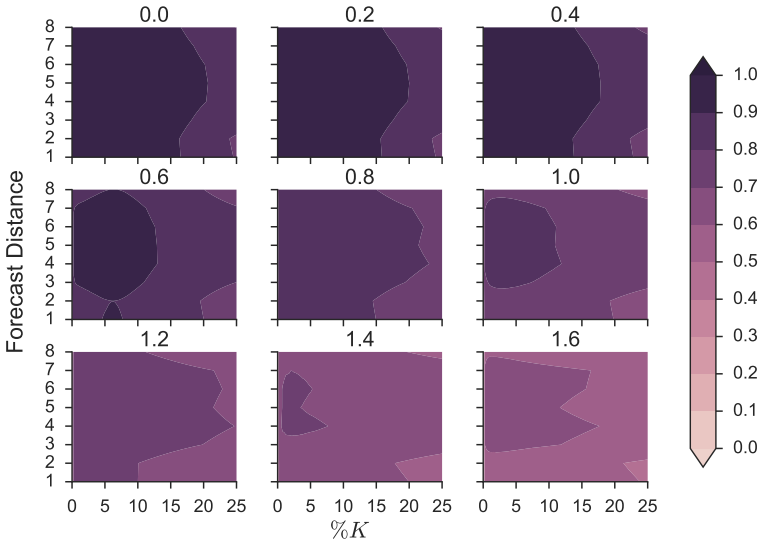

Figure 20: Non-linear forecasting results for the spatio-temporal periodic image with different amplitudes of noise. The coefficient of determination, R^2, is plotted against near neighbors (K) and forecast distance. The number on the plot indicates the amplitude of the noise, α.

one-dimensional systems. For the spatio-temporal chaotic map, high R^2 values are obtained at low K values and low forecast distances. This trend is present across all levels of noise and illuminates the nonlinear deterministic dynamics in the system and the sensitivity to initial conditions. Additionally, the two-dimensional periodic image preserves the trends of the one-dimensional periodic equation. Forecast skill does not noticeably decrease with increasing forecast distance and decreases slowly with number of neighbors used to forecast.

The deterministic metric is calculated for each of the contours and the results are shown for the chaotic image and the periodic image in Figures 21 and 22.

The trends in D as noise is added to the system illustrate the effectiveness of the metric in spatio-temporal images. Systems that are more deterministic (less noise) have higher D values and systems that are more stochastic (more noise) have lower D values. Again, it is important to note that the metric is not able to cross over systems with different system dynamics. Comparing D values for a system with periodic dynamics to a system with chaotic dynamics is not useful.

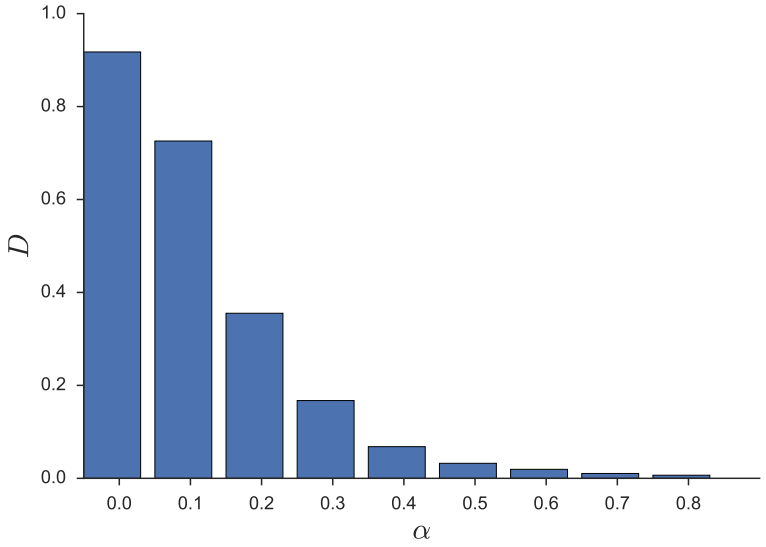

Figure 21: Deterministic metric for the spatio-temporal logistic map. The x-axis indicates the amplitude of the noise, α.

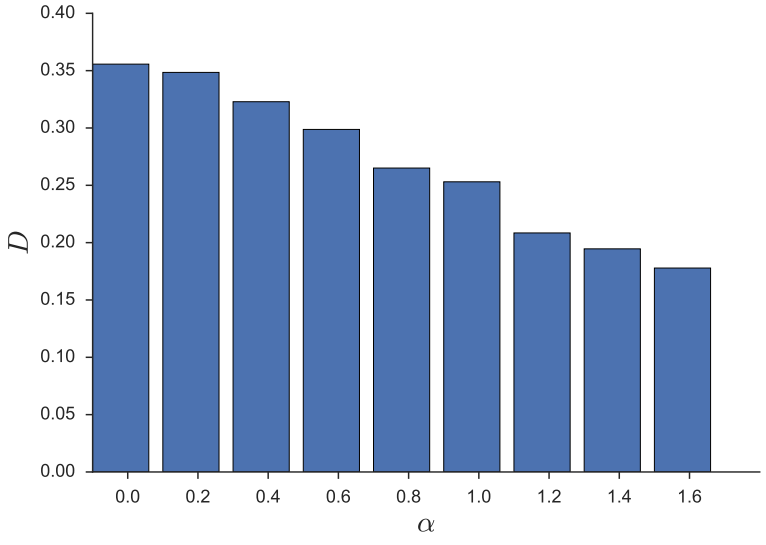

Figure 22: Deterministic metric for the two-dimensional periodic image. The x-axis indicates the amplitude of the noise, α.

2.7 DISCRETE SPACES

The two-dimensional technique is extended to discrete images where each pixel belongs to a class. Images of two different stochastic systems are analyzed (Figure 23). The first image is randomly placed circles of different sizes where the majority of the space is a single class (light blue). The second image is voronoi polygons where each class is equally represented in the space and each is roughly the same size.

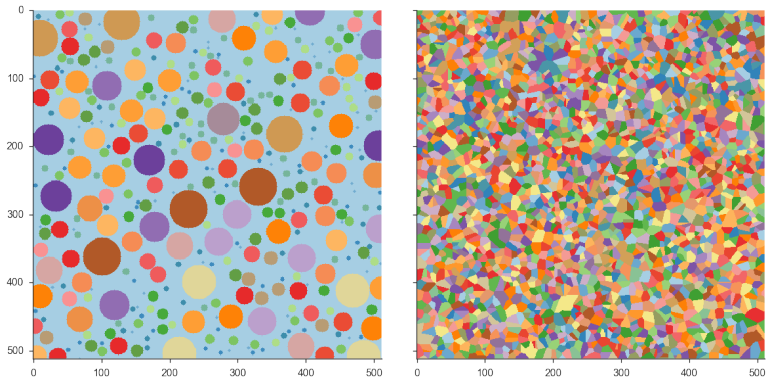

Figure 23: Left: Randomly placed circles of varying size. Right: Voronoi polygons. Color in both images corresponds to class.

In order to apply the non-linear forecasting technique, some modifications are necessary. First, rather than computing the R^2 value when making forecasts, the percent correctly forecast is calculated as:

$$P_c = \frac{N_c}{N_t},$$

where P_c is the percent correct, N_c is the number correctly forecast, and N_t is the total number of forecasts. Second, the hamming distance between two points in the state space is calculated instead of the euclidean distance. This is to account for classes being discrete labels and not continuous values. Figure 24 shows contours of the percent correct for each surrogate image when using the nonlinear forecasting technique adjusted for discrete data.

For both images, forecast skill increases as K increases indicating that both systems are not deterministic. Averaging larger amounts of near neighbors results in better forecast skill

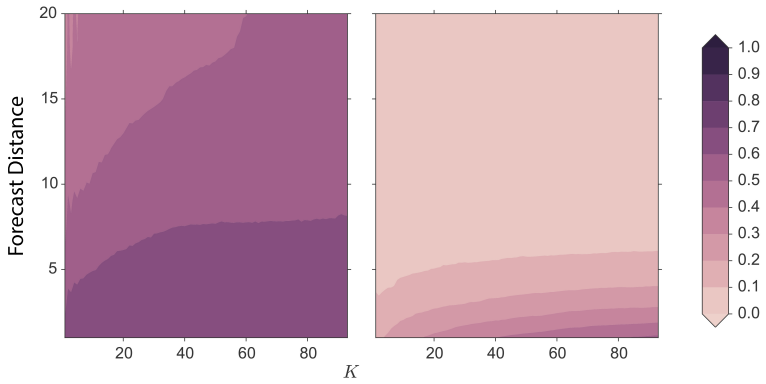

Figure 24: Non-linear forecasting results for the random circles (left) and voronoi polygons (right). The coefficient of determination, R^2, is plotted against near neighbors (K) and forecast distance.

and thus near neighbors in the reconstructed space do not evolve similarly. Additionally, forecast skill decreases as forecast distance increases. The trend, however, is not due to chaotic dynamics and is instead due to the high autocorrelation of the images coupled with its stochastic structure. Specifically, the autocorrelation causes high P_c values at low forecast distances, but the stochasticity causes higher forecast distances to result in lower P_c values. For the randomly dropped circles, averaging additional trajectories results in a forecast of the mode of the space (light blue). For the voronoi polygons, P_c drops to zero at a forecast distance of five pixels, which is roughly the radius of the polygons.

While there is no comparison here of the deterministic metric along a gradient of noise, the patterns associated with stochastic systems are present: increasing forecast skill with increasing K values. Moreover, both images produce a negative deterministic metric indicating that the system is stochastic.

3 DATA COLLECTION

3.1 COASTAL ZONES

To illustrate the forecasting technique on a natural time series, data is collected for coastal evolution. The study site is a 300 meter, alongshore section of south-central Wrightsville Beach, North Carolina. Figure 25 illustrates the location of the camera (red star), the region of interest (yellow boundary), the camera system, and an oblique image from the dataset. The device is located on the roof of a nine-story condominium. Hourly sampling of the region began on August 22, 2013 and spans to September 1, 2015. On April 23, 2014, the beach underwent an engineered re-nourishment where sand was taken from a nearby inlet and placed on the subaerial beach and shoreface.

In conjunction with snapshots, the camera captures a five minute video and averages frames from the video to generate a single time lapse image [13]. Using an orthorectification procedure [38], image coordinates are converted to physical ground units. The oblique image of Figure 25 is displayed in world coordinates in Figure 26. The image then undergoes a classification procedure using a neural network trained on RGB values of hand classified images [12]. The image is then post-processed to cleanly segment the image into beach, ocean, dune, and foam. More details about the camera, capturing procedure, and image segmentation can been obtained in Reference [11].

After segmentation, the width of the beach is calculated. The profile of the inter-tidal region is constructed by plotting the beach width against the tidal level having corrected for swash run-up and setup due to gradients in breaking wave heights [25]. The daily intertidal profile is then interpolated to give continuous values at all tidal levels. Finally, the daily profiles are concatenated over the study period (Figure 27).

The study period is split into pre-nourishment and post-nourishment segments. The pre-nourishment data spans from August 22, 2013 to April 23, 2014. The post-nourishment data spans from April 24, 2014 to September 1, 2015.

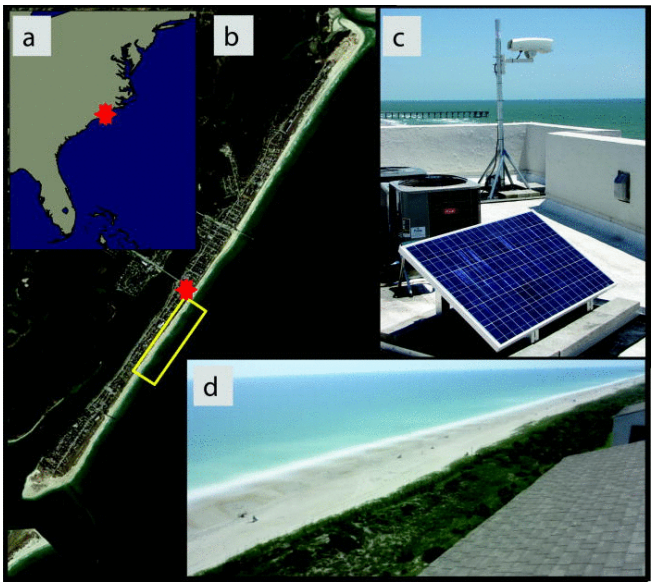

Figure 25: A: Location of Study: Wilmington, NC. B: Region of data collection. C: Solar powered camera system. D: Sample image from the camera.

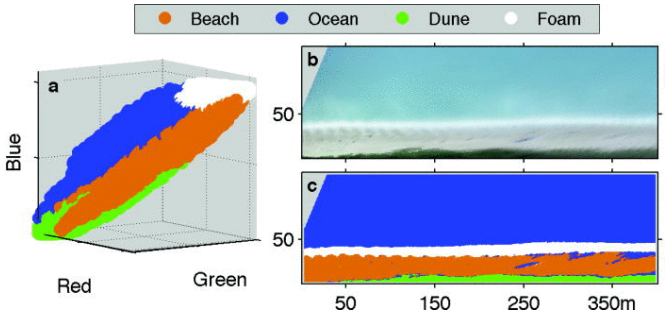

Figure 26: (a) 3-D axes where RGB vectors from sample images are plotted and colored based on the ANN determined class (see legend). (b) A sample rectified image. (c) The images classified counterpart.

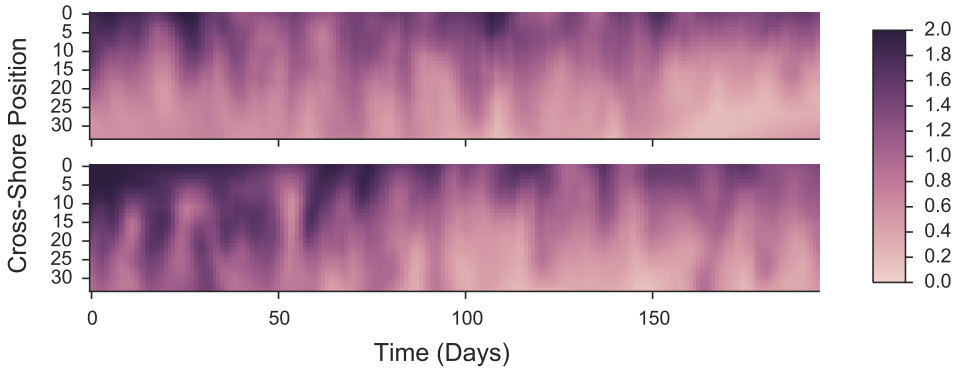

Figure 27: Intertidal profile height plotted against time. Top panel is pre-nourishment. Bottom panel is post-nourishment.

3.2 CORAL REEFS

For an additional illustration of nonlinear forecasting on natural data, photographic data of coral reefs are collected. Specifically, images of the benthos were captured along islands within the Line Islands Chain in the Equatorial Pacific by a team from the Scripps Institution of Oceanography (SIO). Divers swam over coral reefs in organized patterns and used specialized underwater cameras to capture a series of overlapping images. A team at the University of Miami stitched together images captured from the divers to create a single large mosaic image of the reef (Figure 28). Marine biology graduate students at SIO then manually scanned the mosaic images to identify each benthic species within the image. Figure 29 shows a manually classified image, where each color represents a different species.

Mosaics were collected in 2012 and again in 2013 at three islands within the Line Islands (Palmyra, Fanning, and Christmas). At each island, 20 meter by 20 meter zones were identified for monitoring and given a unique name for efficient data collection. Each island contains different human influence on coral reef habitat as the populations, water pollution, and sediment concentrations in the water vary between islands. These varying degrees of human activity have caused correspondingly different levels of coral health [4] (Table 1). Manually classified mosaics for each of the islands studied here are shown in Figure 30 where species have been coalesced into different morphological groupings.

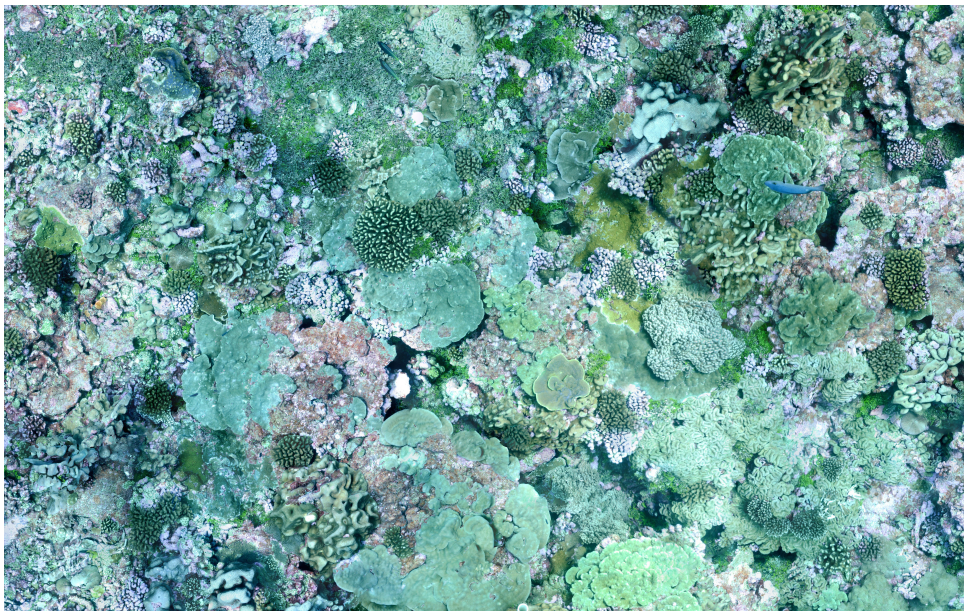

Figure 28: A portion of the raw photo-mosaic from Palmyra Island.

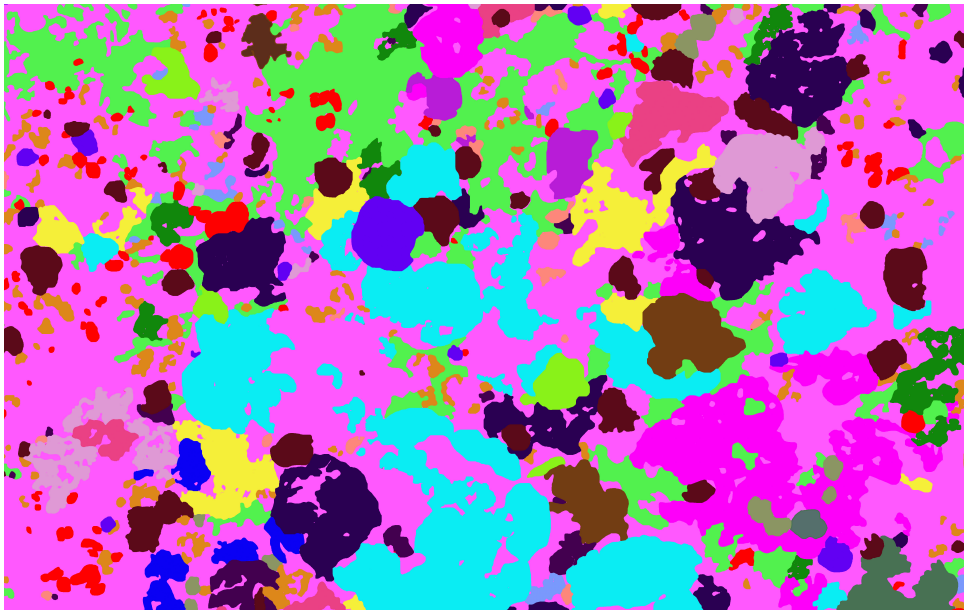

Figure 29: A portion of a classified photo-mosaic from Palmyra Island. Each color corresponds to a different coral or algae benthic species.

Table 1: Coral Zone Summary

Zone	Location	Population	Coral Health	Study Sites
Palmyra	5.8833°N, 162.0833° W	4	Great	FR 3, 5, 7, 9
Fanning	3.825°N, 162.349S°W	2,500	Good	FB 3, 4, 6, 9, 10, 11
Christmas	2.008°N, 157.489°W	5,000	Poor	KB4

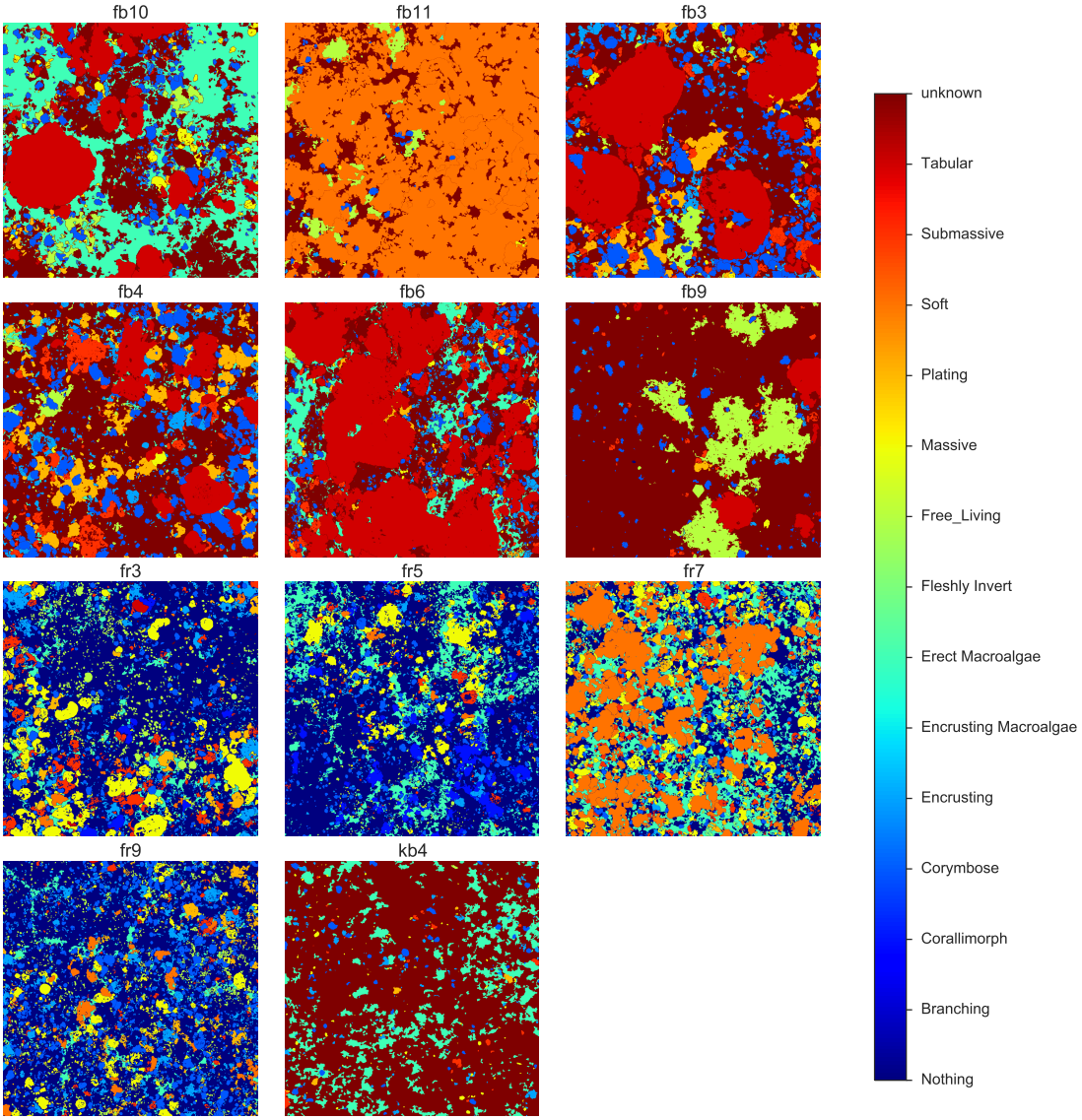

Figure 30: Manually classified photo-mosaics from Palmyra, Fanning, and Christmas islands. Each color corresponds to a different benthic morphological group. Image size is 1,500x1,500 pixels.

4 DETERMINISTIC ANALYSIS

4.1 COASTAL ANALYSIS

The non-linear forecasting technique is applied independently to the pre-nourishment and post-nourishment intertidal beach profiles (Figure 27). The R^2 values are shown in Figure 31.

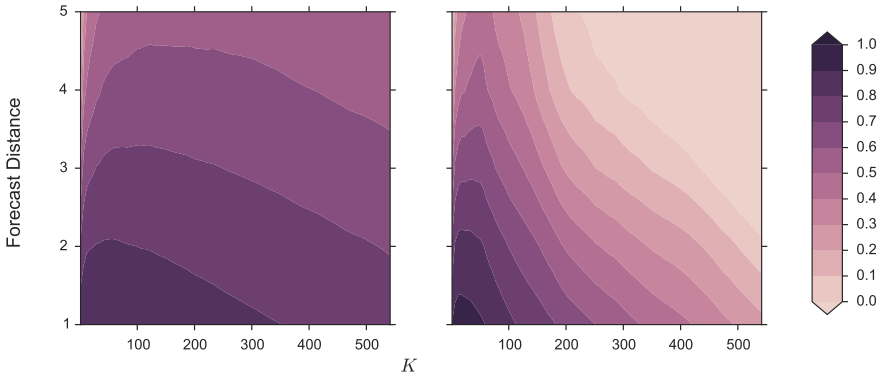

Figure 31: Non-linear forecasting results for the pre-nourishment beach (left) and post-nourishment beach (right). The coefficient of determination, R^2, is plotted against near neighbors (K) and forecast distance.

The non-linear forecasting results show two very different trends. For the pre-nourishment beach, the R^2 values drop off more slowly than the post-nourishment beach for both forecast distance and near neighbors. One interpretation of the results for the pre-nourishment beach is that the high forecast skill over a relatively wide range of low K and forecast distance indicates that the state space for the pre-nourishment beach is more populated than the post-nourishment beach. Additionally, since forecast skill drops off more quickly for the post-nourishment beach, near neighbors in the state space do not evolve as similarly as the pre-nourishment beach. These trends seem to suggest that the nourishment episode abruptly moved the intertidal region away from its attractor and that it is slowly progressing back toward it's steady state and in doing so is dominated by internal system dynamics as opposed to responding to noisy forcing.

4.2 CORAL ANALYSIS

The non-linear forecasting technique is applied to the coral images and the resulting contour plots of forecast skill are shown in Figure 32.

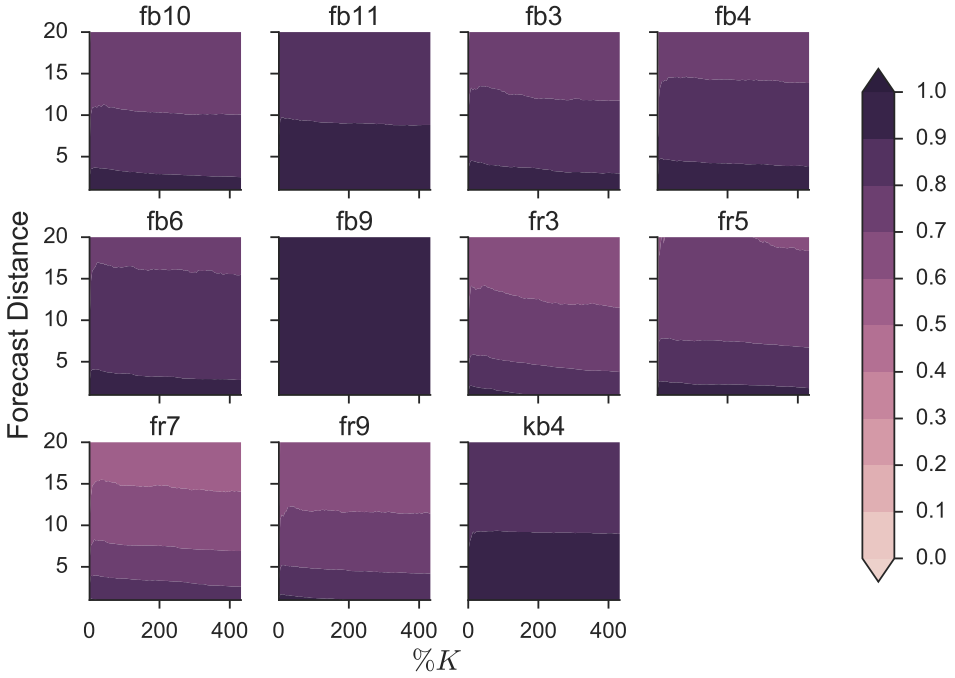

Figure 32: Non-linear forecasting results for the coral zones. The percent correctly forecast, P_c, is plotted against near neighbors (K) and forecast distance.

All zones display the same trend: highest P_c values are obtained at low K values and low forecast distances. This indicates that each zone is deterministic and that nearby regions in the reconstructed state space evolve similarly. Physically, this means that species on reefs influence the distribution of species around them. Assuming that all of zones are generated by the same dynamics (competition for benthic space between species on the coral reef) and that all of the state spaces are well populated, the deterministic metric can be compared across the various islands (Figure 33).

The metric shows a distinct trend with respect to each of the three islands with the highest values occurring at Palmyra. It appears as though the relatively pristine reefs of Palmyra have a more spatially deterministic structure, suggesting that non-linear internal

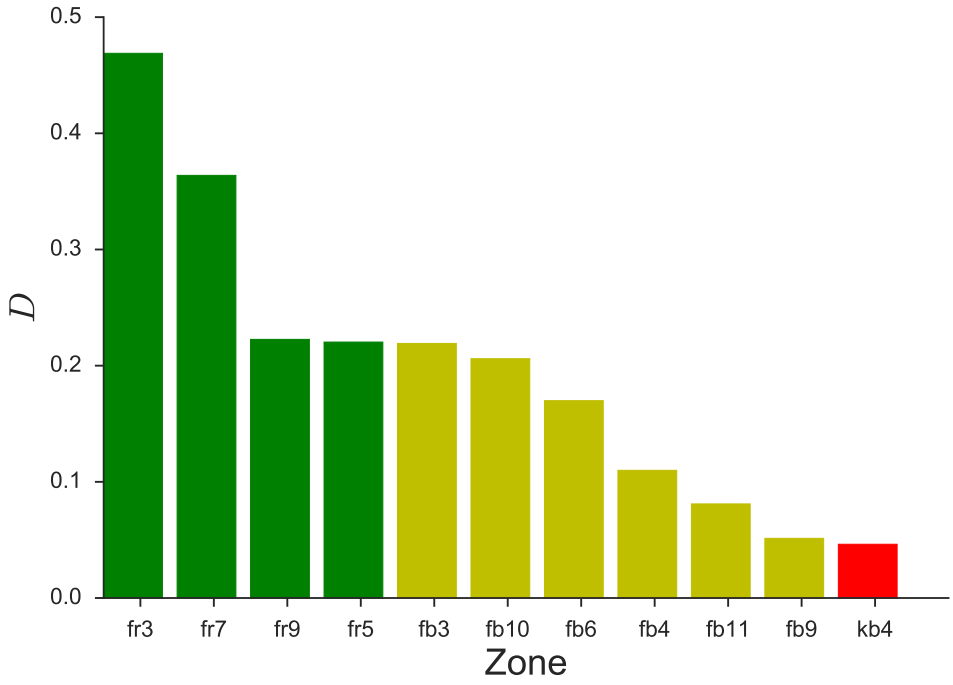

Figure 33: Deterministic metric, D, for the coral zones. The x-axis and color indicate location (green is Palmyra, yellow is Fanning, and red is Christmas).

spatial dynamics play a larger role in dictating benthic patterns at Palmyra than at the other islands in this study. The second grouping is four of the six Fanning Island zones. This indicates that there is still structure in the benthic patterning and that internal non-linear dynamics play a role in shaping the structure, but the relative influence of spatial noise is stronger than at Palmyra. The last grouping consists of three zones at Fanning and Christmas. These locations show the least deterministic spatial structure according to the metric and these locations are in fact the most degraded of all the study zones. These results indicate that the non-linear forecasting technique coupled with the deterministic metric is able to distinguish relative reef health. This suggests that pollution and disturbance of the reef ecosystem destroys natural spatial relationships (non-linear internal dynamics) resulting in a benthic spatial structure that is more random.

5 CONCLUSION

Collection of remote images of the nearshore region and photomosaics of the coral reef benthos has provided a series of spatiotemporal images of the upper shore-face and a series of spatial images along a gradient of coral reef degradation. Forecast skill from the novel application of both spatiotemporal nonlinear time series forecasting and the spatial analog suggest that the evolution of coastal and coral zones are governed by internal non-linear dynamics. Both forecast techniques elucidated the relative amount of stochasticity in the respective systems.

Although the metric was able to determine the relative amount of stochasticity within the systems, it does not provide an absolute measure of determinism across systems generated by different dynamics. For example, the absolute value of the deterministic metric was not able to capture the amount of determinism in the one-dimensional periodic equation when compared to the one-dimensional chaotic map. Moreover, the technique does not provide insight into the actual dynamics that generate a system. For the coral images, nothing is discovered concerning what spatial dynamics created the benthos other than the fact that the final configuration is deterministic.

The largest drawback of the technique is the large amount of data needed while the system is in it's steady state. Incomplete reconstructed state spaces result in forecast skill that mimics noise within the system even if the system is perfectly deterministic. Experimentation with small amounts of data for the one-dimensional chaotic map have shown suppressed R^2 values even for perfectly deterministic systems. Additionally, for high dimensional reconstructed state spaces, it is difficult to tell whether the system is in not in a steady state or the system is suffering from an incomplete attractor. This was apparent with the post-nourishment beach. The system most likely was not in a steady state, but it is also possible that the phase space for the post-nourishment beach is incomplete and more data is needed.

Nonetheless, the technique presented here was able to clearly distinguish the different system evolution for the pre-nourishment beach and post-nourishment as well as the different

spatial configurations along the gradient of coral degradation. Likewise, the deterministic metric was able to succinctly illuminate the non-linear influence between near neighbors in the reconstructed state space between both the mathematical equations along a gradient of noise and natural systems (beach and coral). With significant human enterprise located in coastal regions around the world and the high economic benefit of coral reefs, there is a premium placed on the ability to understand coastline behavior and coral reef health. The results presented here suggest that image data coupled with nonlinear forecasting techniques can result in an insight into the evolution of the systems in time or space.

APPENDIX

A IMPLEMENTATION OF THE NON-LINEAR FORECASTING TECHNIQUE

The technique presented in this thesis was implemented in the Anaconda distribution of Python 2.7. Array manipulation was implemented in Numpy [36]. Plotting was handled by Matplotlib [16] and stylized by Seaborn [37]. The heart of the non-linear forecasting technique is the near neighbor algorithm from Sci-kit Learn [23]. The work was completed in IPython Notebooks [24]. The raw code for this project and accompanying IPython Notebooks can be found at nickc1.github.com.

The following subsections will go into more detail about the non-linear forecasting technique, but the general algorithm follows these steps:

1. Input one-dimensional time series or two-dimensional image.
2. Calculate the lag value which is the first minimum of the mutual information.
3. Split data into training set and testing set.
4. Embed each in an m-dimensional space.
5. Calculate the distance from all vectors in the test set to all vectors in the training set.
6. Average the evolution of the K closest vectors in the training set to make forecasts for the testing set.
7. Compare forecasts to actual evolution of the system.
8. Repeat steps 6 and 7 with K ranging from 1 to 25% of the total size of the training set.
9. Repeat steps 4 through 8 to find the embedding dimension with highest forecast skill.
10. Calculate the Deterministic Metric.

A.1 LAG VALUE CALCULATION

Before embedding a time series or an image, a lag value must be calculated. Lag values are given by the first minimum in the mutual information. A calculation of the mutual information for a time series, X, is implemented in python as:

```
def mutual_information(X, max_lag):
  '''
  Calculates the mutual information
  '''

  #number of bins - say ~ 20 pts / bin for joint distribution
  #and that at least 4 bins are required
  N = max(X.shape)
  num_bins = max(4.,np.floor(np.sqrt(N/20))).astype('int')

  m_score = np.zeros((max_lag))

  for jj in range(max_lag):
    lag = jj+1

    ts = X[0:-lag]
    ts_shift = X[lag::]

    min_ts = np.min(X)
    max_ts = np.max(X)+.0001 #needed to bin them up

    bins = np.linspace(min_ts,max_ts,num_bins+1)

    bin_tracker = np.zeros_like(ts)
    bin_tracker_shift = np.zeros_like(ts_shift)

    for ii in range(num_bins):
```

```
28
29          locs = np.logical_and( ts>=bins[ii], ts<bins[ii+1] )
30          bin_tracker[locs] = ii
31
32          locs_shift = np.logical_and( ts_shift>=bins[ii], ts_shift<bins[ii+1] )
33          bin_tracker_shift[locs_shift]=ii
34
35
36      m_score[jj] = metrics.mutual_info_score(bin_tracker,bin_tracker_shift)
37    return m_score
```

In this implementation, X is the time series and max_lag is the maximum number of time steps that the time series is shifted. After shifting the time series and binning the time series values, sci-kit learn's mutual information implementation is calculated as,

$$MI(U,V) = \sum_{i=1}^{R}\sum_{j=1}^{C} P(i,j) log \frac{P(i,j)}{P(i)P'(j)}.$$

Here $P(i)$ is the probability of a random sample occurring in U_i (the un-shifted time series) and $P'(j)$ is the probability of a random sample occurring in V_j (the shifted time series). $P(i,j)$ is the probability of both U_i and V_j occurring in the same random sample. This function will output a series of mutual information values corresponding to the amount that the time series is shifted. For more information on lagged values for reconstructing state spaces refer to Reference [9].

For two-dimensional images, the mutual information along each row and each column is calculated and then summed together. This is implemented in python as,

```
1 def mutual_information_spatial(M, max_lag, percent_calc=.5):
2   '''
3   Calculates the mutual information along the rows and columns at a
4   certain number of indices (percent_calc) and returns
5   the sum of the mutual informaiton along the columns and along the rows.
6
```

```
M: Input matrix
num_bins: Parameter for mutual info calculation
max_lag: How far to shift the space
percent_calc: How many rows and columns to use

Returns:
R_mut: the mutual inforation down the rows (vertical)
C_mut: the mutual information across the columns (horizontal)

'''

rs, cs = np.shape(M)

rs_iters = int(rs*percent_calc)
cs_iters = int(cs*percent_calc)

r_picks = np.random.choice(np.arange(rs),size=rs_iters,replace=False)
c_picks = np.random.choice(np.arange(cs),size=cs_iters,replace=False)


# The r_picks are used to calculate the MI in the columns
# and the c_picks are used to calculate the MI in the rows

c_mi = np.zeros((max_lag,rs_iters))
r_mi = np.zeros((max_lag,cs_iters))

for ii in range(rs_iters):

  m_slice = M[r_picks[ii],:]

  c_mi[:,ii] = mutual_infomation(m_slice,max_lag)

for ii in range(cs_iters):
```

```
40
41      m_slice = M[:, c_picks[ii]]
42      r_mi[:, ii] = mutual_infomation(m_slice, max_lag)
43
44  r_mut = np.sum(r_mi, axis=1)
45  c_mut = np.sum(c_mi, axis=1)
46
47  return r_mut, c_mut, r_mi, c_mi
```

In this implementation M is the input image, *max_lag* is the maximum amount of image shift, and *percent_calc* is the percent of rows and columns that are summed over to calculate the mutual information. The *percent_calc* is used to speed up the computation. Experimentation has shown that no accuracy is lost summing over a smaller amount of the rows or columns for the mutual information calculation. This function utilizes the one dimensional $mutual\_information$ function above.

A.2 TRAINING AND TESTING SETS

When implementing the non-linear forecasting technique, the time series and images are split into a training set and a testing set. Usually, the training set consists of the first two-thirds of the data and the testing set is composed of the final one-third. By splitting the time series and images, forecasts for the test set using the training set can be compared to the actual values of the testing set. The accuracy of the forecasts can then be quantified. An implementation in python is given as,

```
1 def split(X, train_split=2./3):
2   '''
3   Split a space into training and test set.
4   '''
5
6   r_split = int( X.shape[0] * train_split )
7
```

```
    try:
        XTrain= X[0:r_split,:]  # select the top
        XTest = X[r_split:,:]  # select the bottom
    except:
        XTrain[0:r_split]
        XTest[r_split::]

    return XTrain, XTest
```

This function is capable of handling both one dimensional and two dimensional inputs. The function utilizes the *try*, *except* block to determine if the input is one dimensional or two dimensional.

A.3 EMBEDDING

The testing set and training sets are embedded independently in an m-dimensional space. The dimensionality, m, is iterated at a later step using a false near neighbors technique to determine the optimal embedding [18]. An example of a single embedded vector of dimension 3 and a lag value of 2 for the first eight values of a time series, T is given by the green squares in Figure 34.

Figure 34: Embedding technique illustrating an embedding dimension of three with lag value of two. Green is the embedded time series values and red is the forecast y values.

The green squares are shifted by one to embed the next vector in the m-dimensional space. All of the embedded vectors are stored in a matrix, X. The red squares in Figure 34 represent the evolution vectors of the system. Likewise, the red squares are shifted over and the next evolution vector is obtained. These values are stored in a matrix y. The process is repeated for the embedded vectors and evolution vectors over the length of the time series. A generalization of this technique is given by:

$$X = \begin{bmatrix} T_t & T_{t+\tau} & T_{t+2\tau} & \dots & T_{t+m\tau} \\ T_{t+1} & T_{t+1+\tau} & T_{t+1+2\tau} & \dots & T_{t+1+m\tau} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ T_{t+n} & T_{t+n+\tau} & T_{t+n+2\tau} & \dots & T_{t+n+m\tau} \end{bmatrix}$$

$$Y = \begin{bmatrix} T_{t+m\tau+1} & T_{t+m\tau+2} & \dots & T_{t+m\tau+p} \\ T_{t+1+m\tau+1} & T_{t+1+m\tau+2} & \dots & T_{t+1+m\tau+p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ T_{t+n+m\tau+1} & T_{t+n+\tau+2} & \dots & T_{t+n+m\tau+p} \end{bmatrix}$$

Where τ is the lag value, m is the embedding dimension, p is the forecast distance, and n is the length of T minus the prediction distance. This implemented in python is given as,

```
def lagReshape(X,em,lag,predict):
    '''
    Constructs the space for prediction using the specified lag
    value and embedding dimension.

    Example:

    X = [0,1,2,3,4,5,6,7,8,9,10]

    em = 3
    lag = 2
    predict=3

    returns:
    features = [0,2,4], [1,3,5], [2,4,6], [3,5,7]
    targets = [5,6,7], [6,7,8], [7,8,9], [8,9,10]
    '''

    tsize = X.shape[0]
```

```
  t_iter = tsize-predict-(lag*(em-1))

  features = np.zeros((t_iter,em))
  targets = np.zeros((t_iter,predict))

  for ii in range(t_iter):

    end_val = ii+lag*(em-1)+1

    part = X[ii : end_val]

    features[ii,:] = part[::(lag)]
    targets[ii,:] = X[end_val:end_val+predict]
  return features, targets
```

The two-dimensional version of the embedding is similar to the time series version. An example embedding of a spatio-temporal image is given in Figure 35.

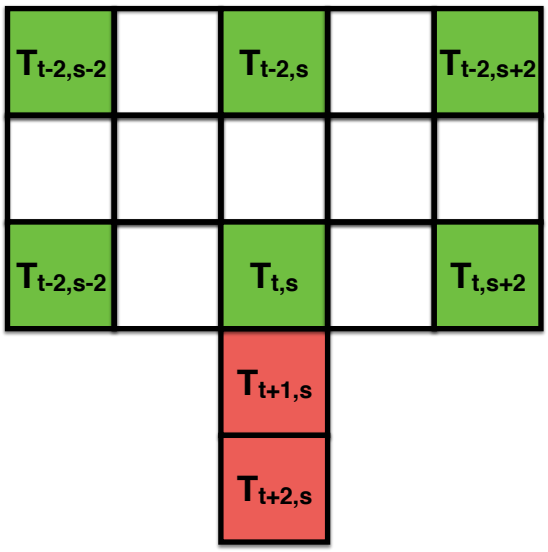

Figure 35: Embedding technique illustrating an embedding dimension of six with lag value of two in both space and time. Green is the embedded spatio-temporal series values and red is the forecast y values.

The embedded vector (green) is six dimensional and has a spatial lag of 2 and a temporal

lag of 2. The evolution vector is given in red. Both the embedded vector and the evolution vector are shifted over and the next embedded vector and evolution vector are calculated. The embedded vectors are stored in a matrix, $\vec{X}$, and the evolution vectors are stored in a matrix, y. Formally, $\vec{X}$ and $\vec{Y}$ can be defined as,

$$\vec{X}_{(t,s)}(x) = (x_{(t,s)}, x_{(t-\tau,s)}, \ldots, x_{t-(m-1)\tau,s\pm\frac{n-1}{2}\sigma}),$$

$$\vec{Y}_{(t,s)}(x) = (x_{(t+1,s)}, x_{(t+2,s)}, \ldots, x_{(t+p,s)}),$$

where x is the spatio-temporal image, s refers to the spatial component of the series, τ is the spatial lag, σ is the spatial embedding dimension, and p is the forecast distance. An implementation in python is given by:

```
def ftReshapePercent(X, lag, embed, predict, percent=None):
  '''
  X:       Matrix to embed
  lag:     tuple of row and column lag values (r,c)
  embed:   tuple of embedding shape (r,c)
  predict:  How far to predict outward
  percent:  What percent of the space to reshape (pics random locs)

  Can think of the lag and embedding as height,width


  Example:
  lag = (3,4)
  embed = (2,5)
  predict = 2


  [ ] - - - [ ] - - - [ ] - - - [ ] - - - [ ]
```

```
19  |         |         |         |         |
20  |         |         |         |         |
21  [ ] - - - [ ] - - - [ ] - - - [ ] - - - [ ]
22   *
23   *
24  ,,,
25
26  rsize = X.shape[0]
27  csize = X.shape[1]
28
29  r_lag,c_lag = lag
30  rem,cem = embed
31
32
33  # determine how many iterations we will have and
34  # the empty feature and target matrices
35
36  c_iter = csize - c_lag*(cem-1)
37  r_iter = rsize  - predict - r_lag*(rem-1)
38
39  #create tuples of all the possible x,y values for the image
40  # creates a bunch of tuples
41  xx,yy = np.meshgrid(range(r_iter),range(c_iter))
42  z = zip(xx.ravel(),yy.ravel())
43
44  #choose only a percent of them if percent is defined
45  if percent:
46    tot = r_iter*c_iter
47    percent_tot = int(tot*percent)
48    rand_pic = np.random.choice(tot,percent_tot,replace=False)
49
50    z = [z[ii] for ii in rand_pic]
51
```

```
52      targets = np.zeros((percent_tot,predict))
53      features = np.zeros((percent_tot,rem*cem))
54
55
56  else:
57      targets = np.zeros((c_iter*r_iter,predict))
58      features = np.zeros((c_iter*r_iter,rem*cem))
59
60
61
62  print 'targets before loop:', targets.shape
63
64  for ii in range(features.shape[0]):
65
66      rs,cs = z[ii]
67
68
69      r_end_val = rs+r_lag*(rem-1)+1
70      c_end_val = cs+c_lag*(cem-1)+1
71
72      part = X[rs : r_end_val, cs : c_end_val ]
73
74      features[ii,:] = part[::r_lag,::c_lag].ravel()
75      targets[ii,:] = X[r_end_val:r_end_val+predict,cs+ c_lag*(cem-1)/2]
76
77
78  return features,targets
```

This implementation allows for a certain percent of the space to be transformed into embedded vectors and evolution vectors. This is for computational efficiency in the calculation of distances. Furthermore, experimentation has shown that it is not necessary to test every single location. For most systems 25% is effective. Additionally note that the lag values

and embedding dimension are set up to be tuples. The first value is the lag or embedding dimension down the rows and the second value is across the columns.

A.4 K-NEAREST NEIGHBOR ALGORITHM

At the heart of the non-linear forecasting technique is the K-Nearest Neighbors Algorithm [6]. The algorithm calculates the euclidean distances from each of the testing embedded vectors, $\vec{X}_{test}$ to each of the training embedded vectors, $\vec{X}_{train}$. The distance from the $i'th$ embedded train vector to the $j'th$ embedded test vector is,

$$d_i(X_{train,i}, X_{test,j}) = \sqrt{(X_{train,i} - X_{test,j})^2}$$

The y_{train} values of the closest k vectors in the training set are averaged and a forecast, y_p, is made. This takes the form,

$$y_p = \frac{y_{train,1} + y_{train,2} + \dots y_{train,k}}{k}.$$

The results of the forecast is then compared to the actual evolution vectors and the coefficient of determination, R^2 is calculated as,

$$R^2 = 1 - \frac{\sum(y_i - f_i)^2}{\sum(y_i - \bar{y})^2},$$

where y_i is the i^{th} prediction, f_i is the true value, and $\bar{y}$ is the mean of the y values. The near neighbor algorithm is implemented in python as,

```
def nnEfficient(F_train,T_train,F_test,T_test,
  nn_iters=100,percent_nn=.25,weights='uniform',compare='score'):
  '''
  Steps from 1 to [percent_nn] percent of total NN in 10 increments.

  F_train: Feature training matrix
  T_train: Target training matrix
```

```
F_test: Feature testing matrix
T_test: Testing training matrix
nn_iters: number of increments for the near neighbors
percent_nn: max percent of nn to test

Returns correlation coefficient vs nn_range
Weight/non_weight code adapted/stolen from scikit-learn

Weights can take on two different forms:
1. uniform
2. weighted
'''

try:
  predict_out = T_test.shape[1]
except:
  predict_out=1


# Step through NN
min_nn = 1
max_nn = np.around(F_train.shape[0] * percent_nn)
step = max_nn/nn_iters
nn_range = np.arange(min_nn,max_nn,step)

cc=np.empty((nn_range.shape[0],predict_out))

#calculate the distances to max_nn using scikit-learn
knn = neighbors.KNeighborsRegressor(int(max_nn),weights=weights,metric='
  minkowski') # uniform | distance
knn_func = knn.fit(F_train,T_train)
d,ind = knn_func.kneighbors(F_test)

```

```
  for ii in range(nn_range.shape[0]):
    nn= int(nn_range[ii])

    if weights== 'uniform':

      neigh_ind = ind[:,0:nn]

      y_pred = np.mean(T_train[neigh_ind], axis=1)

    elif weights =='distance':
      dist = d[:,0:nn]
      neigh_ind = ind[:,0:nn]
      W= 1./dist

      y_pred = np.empty((F_test.shape[0], T_test.shape[1]), dtype=np.float)
      denom = np.sum(W, axis=1)

      for j in range(T_test.shape[1]):
        num = np.sum(T_train[ neigh_ind , j] * W, axis=1)
        y_pred[:, j] = num / denom
        #print num.shape

    if compare == 'score':
      cc[ii,:] = score(y_pred,T_test)


  return nn_range,cc
```

This function utilizes sci-kit learn's *kneighbors* function for efficient calculation of the distances from the testing set to the training set. It also has the ability to calculate weighted averages. For this paper, uniform weightings are used throughout.

In order to forecast discrete classes, the implementation is the same except the hamming

distance is used to calculate the proximity of points and the mode of the k nearest y_train class labels is the forecast where the mode is calculated as,

$$Y_p = mode(y_1, y_2, \ldots, y_k).$$

This slightly different version is implemented in python as,

```
def nnClassificationEfficient(F_train,T_train,F_test,T_test,nn_iters=100,
  percent_nn=.01,weights='uniform',compare='classCompare'):
  '''
  Steps from 1 to [percent_nn] percent of total NN in nn_iters increments.

  F_train: Feature training matrix
  T_train: Target training matrix
  F_test: Feature testing matrix
  T_test: Testing training matrix
  nn_iters: number of increments for the near neighbors
  percent_nn: max percent of nn to test

  Returns percent correct vs nn_range
  Weight/non_weight code adapted/stolen from scikit-learn

  Weights can take on two different forms:
  1. uniform
  2. weighted
  '''

 # Step through NN
  min_nn = 1
  max_nn = np.around(F_train.shape[0] * percent_nn)
  step = max_nn/nn_iters
  nn_range = np.arange(min_nn,max_nn,step)

```

```
27  cc=np.empty((nn_range.shape[0],predict_out))
28
29  #calculate the distances to max_nn using scikit-learn
30  knn = neighbors.KNeighborsRegressor(int(max_nn),weights=weights,metric='
      hamming')
31  knn_func = knn.fit(F_train,T_train)
32  d,ind = knn_func.kneighbors(F_test)
33
34  _y = T_train
35
36  classes_ = np.unique(_y)
37
38  for ii in range(nn_range.shape[0]):
39
40    nn= int(nn_range[ii])
41    neigh_dist = d[:,0:nn]
42    neigh_ind = ind[:,0:nn]
43
44    n_outputs = len(classes_)
45    n_samples = T_test.shape[0]
46
47    W= 1./neigh_dist
48
49    y_pred = np.empty((n_samples, predict_out), dtype=classes_[0].dtype)
50
51    #for k, classes_k in enumerate(classes_):
52    if weights=='uniform':
53      mode, _ = stats.mode(_y[neigh_ind], axis=1)
54    else:
55      for j in range(T_test.shape[1]):
56        mode, _ = weighted_mode(_y[neigh_ind,j], W, axis=1)
57
58        mode = np.asarray(mode.ravel(), dtype=np.intp)
```

```
59
60          y_pred[:, j] = mode
61
62      if compare =='classCompare':
63        cc[ii,:] = classCompare(y_pred, T_test)
64
65    return nn_range, cc
```

Finally in order to evaluate the forecasts a metric is developed that calculates the percent correctly predicted. This is represented simply as,

$$P = \frac{N_c}{N_t},$$

where N_c is the number correctly predicted and N_t is the total number of predictions. The algorithm is then iterated over a number of embedding dimensions m to find the value of m that gives the highest forecast skill. This is implemented in python as,

```
1  def classCompare(preds, test):
2    '''
3    Compares the predicted to the actual classes. It's a binary classification.
4    The output is what percent it got correct.
5    '''
6
7    try:
8      num_preds = preds.shape[1]
9    except:
10     num_preds=1
11
12   cc = np.empty((num_preds,))
13
14   if num_preds ==1:
15     cc = np.mean( preds == test )
16
```

```
  else:

    for ii in range(num_preds):
      cc[ii] =np.mean( preds[:,ii] == test[:,ii] )

  return cc
```

A.5 DETERMINISTIC METRIC

The deterministic metric attempts to quantify the amount of determinism in a given time series or image. After finding the best embedding dimension, m, the algorithm is calculated as,

$$D = \sum_{d=1}^{n} max(R^2_{d,\%K<12.5}) - min(R^2_{d,\%K\geq 12.5}).$$

This function is implemented in python as,

```
def deterministic(X, mut_info):
  '''
  Calculates the deterministic metric out to 1/2 of the mutual
  information for $R^2$ values over a range of near neighbors and forecast
   distances

  Input:
  X : R^2 values over a range of near neighbors and forecast distances
  mut_info : mutual information of the system
  '''

  sz = np.floor(mut_info/2.)
  det_metric = np.zeros((sz,))
  half = X.shape[1]


```

```
  for ii in range(sz):

    det_metric[ii] += np.max(X[0:half,ii]) - np.min(X[half::,ii])

  return det_metric
```

A.6 EXAMPLE: ONE-DIMENSIONAL

Here the the x values from the lorenz system are used to illustrate the workflow for the non-linear forecasting technique. The first step is to generate the time series and import the needed libraries. This is done as python as,

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import integrate
from sklearn import metrics
sns.set()
from sklearn import neighbors

sns.set_context('paper',font_scale=2)
sns.set_style('ticks')

def lorenz(sz=10000,noise=0):
    x0 = [1, 1, 1]  # starting vector
    t = np.linspace(0, 100, sz)  # one thousand time steps
    X = integrate.odeint(lorentz_deriv, x0, t)

    return X


def lorentz_deriv((x, y, z), t0, sigma=10., beta=8./3, rho=28.0):
    """Defining lorenz equation to call above"""
```

```
    return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]


X = lorenz(sz=10000)[::5,0]
plt.plot(X)
plt.xlabel("Time",size=25)
plt.ylabel(r'$X$',size=25)
sns.despine()
```

The required libraries are imported and Seaborn is used to style the plots. Scipy's integrate function is used to integrate the lorenz equations. The resulting plot is shown in Figure 36.

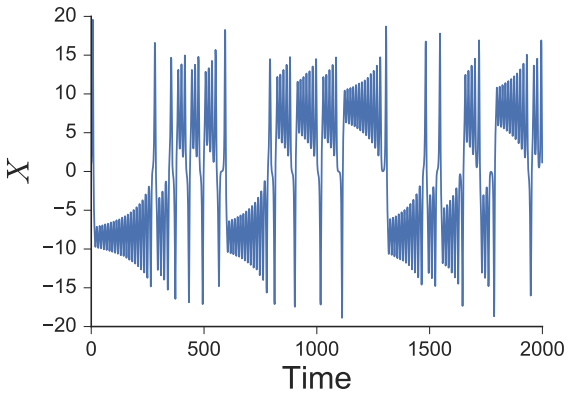

Figure 36: X values from the lorenz equations.

The next step is to calculate a lag value for the system. This is done by finding the first minimum of the mutual information between a shifted time series and an unshifted one. The results of the code are shown in Figure 37.

```
def mutual_information(X,max_lag):
    '''
    Calculates the mutual information
    '''

    #number of bins - say ~ 20 pts / bin for joint distribution
    #and that at least 4 bins are required
```

```
    N = max(X.shape)
    num_bins = max(4.,np.floor(np.sqrt(N/20))).astype('int')

    m_score = np.zeros((max_lag))

    for jj in range(max_lag):
        lag = jj+1

        ts = X[0:-lag]
        ts_shift = X[lag::]

        min_ts = np.min(X)
        max_ts = np.max(X)+.0001 #needed to bin them up

        bins = np.linspace(min_ts,max_ts,num_bins+1)

        bin_tracker = np.zeros_like(ts)
        bin_tracker_shift = np.zeros_like(ts_shift)

        for ii in range(num_bins):

            locs = np.logical_and( ts>=bins[ii], ts<bins[ii+1] )
            bin_tracker[locs] = ii

            locs_shift = np.logical_and( ts_shift>=bins[ii], ts_shift<bins[ii
+1] )
            bin_tracker_shift[locs_shift]=ii


        m_score[jj] = metrics.mutual_info_score(bin_tracker,bin_tracker_shift)
    return m_score

```

```
mi = mutual_information(X,100);

plt.plot(np.arange(1,101),mi)

plt.xlim(1,10)
sns.despine()
plt.xlabel('Shift Amount',size=25)
plt.ylabel('Mutual Information',size=25)
```

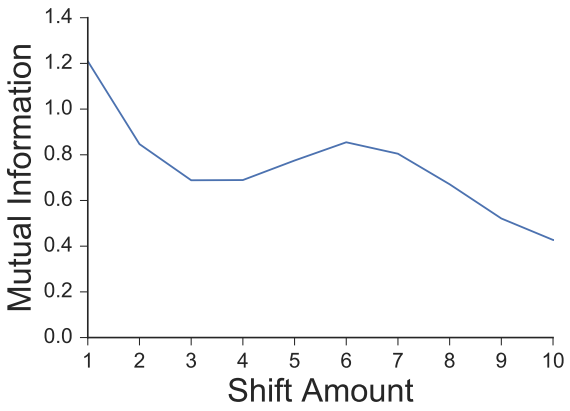

Figure 37: Mutual information between the shifted X values and unshifted X values from the lorenz equations.

Now we can split our time series into a testing set and a training set. This is done in python as,

```
def split(X, train_split=2./3):
    '''
    Split a space into training and test set.
    '''

    r_split = int( X.shape[0] * train_split )

    try:
        XTrain= X[0:r_split,:]  # select the top
        XTest = X[r_split:,:]  # select the bottom
    except:
```

```
        XTrain = X[0:r_split]
        XTest = X[r_split::]

    return XTrain, XTest

xtrain,xtest = split(X)
```

The next step is to iterate over the embedding dimension m to find the embedding that provides the maximum forecast skill. This takes three functions: $nnEfficient$, $score$, and em_check. em_check iterates through m to find the ideal embedding, while $nnEfficient$ finds the near neighbors in the reconstructed state space. $score$ quantifies the forecast skill. The results are shown in Figure 38

```
def nnEfficient(F_train,T_train,F_test,T_test,
    nn_iters=100,percent_nn=.25,weights='uniform',compare='score',metric='
    minkowski'):
    '''
    Steps from 1 to [percent_nn] percent of total NN in 10 increments.

    F_train: Feature training matrix
    T_train: Target training matrix
    F_test: Feature testing matrix
    T_test: Testing training matrix
    nn_iters: number of increments for the near neighbors
    percent_nn: max percent of nn to test

    Returns correlation coefficient vs nn_range
    Weight/non_weight code adapted/stolen from scikit-learn
    45 seconds to run a 128 x 128 region.
    There are three ways to compare the predicted and actual values:
    1. Forecast Error - fnc: forecastError
    2. Correlation Coefficient - fnc: corrCoef
    3. Class Compare - fnc: classCompare
```

```
    ** Must use metric='hamming' for classCompare. Also toggles 'mode' for predictions **

    Weights can take on two different forms:
    1. uniform
    2. weighted
    '''

    try:
        predict_out = T_test.shape[1]
    except:
        predict_out=1


    # Step through NN
    min_nn = 1
    max_nn = np.around(F_train.shape[0] * percent_nn)
    step = max_nn/nn_iters
    nn_range = np.arange(min_nn,max_nn,step)

    cc=np.empty((nn_range.shape[0],predict_out))

    #calculate the distances to max_nn using scikit-learn
    knn = neighbors.KNeighborsRegressor(int(max_nn),weights=weights,metric=metric) # uniform | distance
    knn_func = knn.fit(F_train,T_train)
    d,ind = knn_func.kneighbors(F_test)


    #print 'weights=', weights
    #print 'metric=', metric
    #print 'compare=', compare

```

```
51      for ii in range(nn_range.shape[0]):
52          nn= int(nn_range[ii])
53
54
55
56          if weights== 'uniform':
57
58              neigh_ind = ind[:,0:nn]
59
60
61              y_pred = np.mean(T_train[neigh_ind], axis=1)
62
63
64          elif weights =='distance':
65              dist = d[:,0:nn]
66              neigh_ind = ind[:,0:nn]
67              W= 1./dist
68
69              y_pred = np.empty((F_test.shape[0], T_test.shape[1]), dtype=np.
    float)
70              denom = np.sum(W, axis=1)
71
72              for j in range(T_test.shape[1]):
73                  num = np.sum(T_train[ neigh_ind , j] * W, axis=1)
74                  y_pred[:, j] = num / denom
75                  #print num.shape
76
77          # do the testing
78          if compare == 'forecastError':
79              cc[ii,:] = forecastError(y_pred,T_test)
80
81          elif compare == 'score':
82              cc[ii,:] = score(y_pred,T_test)
```

```
83
84          elif compare == 'corrCoef':
85              cc[ii,:] = corrCoef(y_pred,T_test)
86
87          elif compare =='classCompare':
88              cc[ii,:] = classCompare(y_pred,T_test)
89
90          elif compare =='varianceExplained':
91              cc[ii,:] = varianceExplained(y_pred,T_test)
92
93
94      return nn_range,cc
95
96  def score(preds,test):
97      '''
98      The coefficient R^2 is defined as (1 - u/v), where u is the regression
99      sum of squares ((y_true - y_pred) ** 2).sum() and v is the residual
100     sum of squares ((y_true - y_true.mean()) ** 2).sum(). Best possible
101     score is 1.0, lower values are worse.
102     '''
103
104     num_preds = preds.shape[1]
105     r2 = np.empty((num_preds,))
106
107     for ii in range(num_preds):
108
109         u = np.square(test[:,ii]-preds[:,ii]).sum()
110         v = np.square(test[:,ii]-test[:,ii].mean()).sum()
111         r2[ii] = 1 - u/v
112
113     return r2
114
115
```

```
def em_check(xtrain,xtest,lag,predict,max_em):

    em_mean = np.empty((max_em,))
    em_max = np.empty((max_em,))
    for ii in range(max_em):
        em = ii+1
        f_train, t_train = lagReshape(xtrain,em,lag,predict)
        f_test, t_test = lagReshape(xtest,em,lag,predict)
        nn_range,cc = nnEfficient(f_train,t_train,f_test,t_test,weights='
    uniform')

        em_mean[ii] = np.mean(cc)
        em_max[ii] = np.max(cc)
    return em_mean, em_max


em_mean,em_max = em_check(xtrain,xtest,lag,30,10)

fig,ax = plt.subplots(nrows=2,sharex=True)
m = np.arange(1,11)

ax[0].plot(m,em_mean,linewidth=2)
ax[0].set_ylabel('Mean $R^2$')
sns.despine()

ax[1].plot(m,em_max,linewidth=2)
ax[1].set_ylabel('Max $R^2$')
sns.despine()


fig.subplots_adjust(hspace=.4)
```

According to the ideal embedding for the time series is two. This is then used to embed

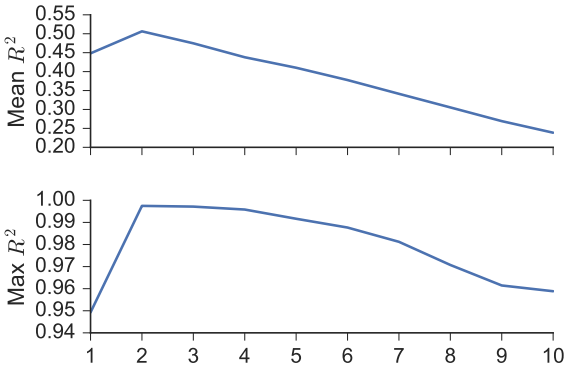

Figure 38: Coefficient of determination R^2 versus embedding dimension, m.

our data. We can visualize the embedding through the code below that produces Figure 39.

```
em = 2
lag = 3
predict = 20
f_train, t_train = lagReshape(xtrain,em,lag,predict)
f_test, t_test = lagReshape(xtest,em,lag,predict)

plt.plot(f_train[:,0],f_train[:,1],linewidth=.5)
sns.despine()
```

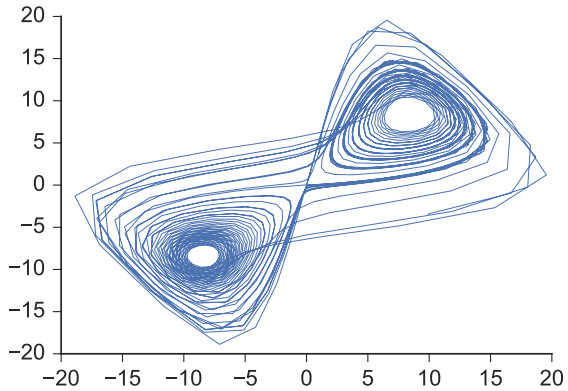

Figure 39: Two-dimensional embedding of the lorenz X values.

Finally we can run $nnEfficient$ and produce a contour plot of the results in order to visualize the trends in forecast skill with respect to the number of neighbors used. The code below produces Figure 40

```
nn_range,cc = nnEfficient(f_train,t_train,f_test,t_test,weights='uniform')
plt.contourf(cc.T);
plt.colorbar()
```

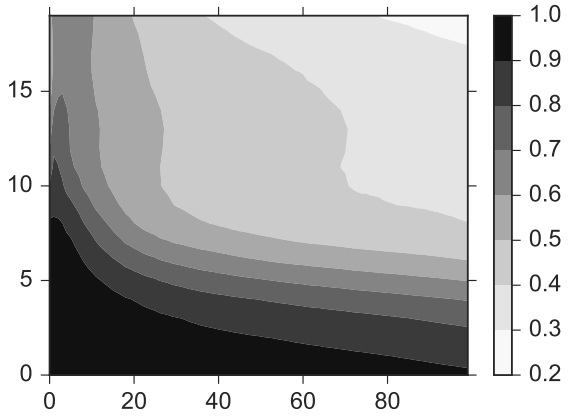

Figure 40: Coefficient of determination R^2 versus embedding dimension, m.

REFERENCES

[1] Henry Abarbanel. *Analysis of observed chaotic data*. Springer Science & Business Media, 2012.

[2] David G Aubrey, Douglas L Inman, and Charles E Nordstrom. Beach profiles at torrey pines, california. *Coastal Engineering Proceedings*, 1(15), 1976.

[3] James A Bailard. An energetics total load sediment transport model for a plane sloping beach. *Journal of Geophysical Research: Oceans (1978–2012)*, 86(C11):10938–10954, 1981.

[4] Katie L Barott, Gareth J Williams, Mark JA Vermeij, Jill Harris, Jennifer E Smith, Forest L Rohwer, and Stuart A Sandin. Natural history of coral- algae competition across a gradient of human activity in the line islands. *Institute for Biodiversity and Ecosystem Dynamics (IBED)*, 2012.

[5] Herman Cesar, Lauretta Burke, and Lida Pet-Soede. The economics of worldwide coral reef degradation. 2003.

[6] Thomas M Cover and Peter E Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[7] Robert G Dean. Equilibrium beach profiles: characteristics and applications. *Journal of coastal research*, pages 53–84, 1991.

[8] Carl Folke, Steve Carpenter, Brian Walker, Marten Scheffer, Thomas Elmqvist, Lance Gunderson, and CS Holling. Regime shifts, resilience, and biodiversity in ecosystem management. *Annual Review of Ecology, Evolution, and Systematics*, pages 557–581, 2004.

[9] Andrew M Fraser and Harry L Swinney. Independent coordinates for strange attractors from mutual information. *Physical review A*, 33(2):1134, 1986.

[10] Jeffrey Goldstein. Emergence as a construct: History and issues. *Emergence*, 1(1):49–72, 1999.

[11] DJ Grimes, N Cortale, K Baker, and DE McNamara. Nonlinear forecasting of intertidal shoreface evolution. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 25(10):103116, 2015.

[12] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[13] Todd K Holland, Robert Holman, Thomas C Lippmann, John Stanley, Nathaniel Plant, et al. Practical use of video imagery in nearshore oceanographic field studies. *Oceanic Engineering, IEEE Journal of*, 22(1):81–92, 1997.

[14] James R Houston. The economic value of beaches: a 2008 update. *Shore and Beach*, 76(3):22–26, 2008.

[15] Terence P Hughes. Catastrophes, phase shifts, and large-scale degradation of a caribbean coral reef. *Science-AAAS-Weekly Paper Edition*, 265(5178):1547–1551, 1994.

[16] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science and engineering*, 9(3):90–95, 2007.

[17] Douglas J Jerolmack and Chris Paola. Shredding of environmental signals by sediment transport. *Geophysical Research Letters*, 37(19), 2010.

[18] Holger Kantz and Thomas Schreiber. *Nonlinear time series analysis*, volume 7. Cambridge university press, 2004.

[19] Matthew B Kennel, Reggie Brown, and Henry DI Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical review A*, 45(6):3403, 1992.

[20] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2):130–141, 1963.

[21] Robert M May et al. Simple mathematical models with very complicated dynamics. *Nature*, 261(5560):459–467, 1976.

[22] Ulrich Parlitz and Christian Merkwirth. Prediction of spatiotemporal time series based on reconstructed local states. *Physical review letters*, 84(9):1890, 2000.

[23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] Fernando Pérez and Brian E Granger. Ipython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29, 2007.

[25] Nathaniel G Plant and Rob A Holman. Intertidal beach profile estimation using video images. *Marine Geology*, 140(1):1–24, 1997.

[26] Nathaniel G Plant, K Todd Holland, and Rob A Holman. A dynamical attractor governs beach response to storms. *Geophysical Research Letters*, 33(17), 2006.

[27] Jeffrey J Polovina. Model of a coral reef ecosystem. *Coral reefs*, 3(1):1–11, 1984.

[28] David M Rubin. Use of forecasting signatures to help distinguish periodicity, randomness, and chaos in ripples and other spatial patterns. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 2(4):525–535, 1992.

[29] Tim Sauer, James A Yorke, and Martin Casdagli. Embedology. *Journal of statistical Physics*, 65(3-4):579–616, 1991.

[30] MWJ Smit, SGJ Aarninkhof, KM Wijnberg, M González, KS Kingston, HN Southgate, BG Ruessink, RA Holman, E Siegle, M Davidson, et al. The role of video imagery in predicting daily to monthly coastal evolution. *Coastal engineering*, 54(6):539–553, 2007.

[31] RK Smith and KR Bryan. Monitoring beach face volume with a combination of intermittent profiling and video imagery. *Journal of Coastal Research*, pages 892–898, 2007.

[32] George Sugihara. Nonlinear forecasting for the classification of natural time series. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 348(1688):477–495, 1994.

[33] George Sugihara and R Mayf. Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series, 1990.

[34] Floris Takens. *Detecting strange attractors in turbulence*. Springer, 1981.

[35] BG Thom and W Hall. Behaviour of beach profiles during accretion and erosion dominated periods. *Earth Surface Processes and Landforms*, 16(2):113–127, 1991.

[36] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[37] Michael Waskom, Olga Botvinnik, Paul Hobson, John B Cole, Yaroslav Halchenko, Stephan Hoyer, Alistair Miles, et al. Seaborn: v0. 5.0 (november 2014), 2014.

[38] Paul R Wolf. Elements of photogrammetry, with air photo interpretation and remote sensing, 2, 1983.