

A COMPARISON OF LOCAL AND GLOBAL TIME SERIES PREDICTION TECHNIQUES

ARTHUR LAKES LIBRARY
COLORADO SCHOOL OF MINES
GOLDEN, CO 80401

by
Nadine Filosi

ProQuest Number: 10794374

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10794374

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado
Date October 21, 1998

Signed: Nadine Filosi
Nadine Filosi

Approved: Erik S. Van Vleck
Dr. Erik S. Van Vleck
Thesis Advisor

Golden, Colorado
Date October 21, 1998

Graeme Fairweather
Dr. Graeme Fairweather
Professor and Head
Department of Mathematical and
Computer Sciences

ABSTRACT

Two methods are implemented for nonlinear time series modeling and prediction. The first technique is based upon a local linear model using nearest neighbors, and the second is based upon a global model and utilizes a radial basis function approach. Numerical techniques are developed and implemented using Matlab. The predictions of the two methods are analyzed and tested against both synthetic and actual data. Comparisons are made between the local and global prediction methods and the true continuation of the data using the mean squared error.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS	vi
LIST OF TABLES	vii
LIST OF FIGURES	viii
1 INTRODUCTION	1
2 BACKGROUND	4
2.1 Interpolation	4
2.2 Delay Coordinates	5
2.3 Dimension	9
2.4 Cosine Transform	13
2.5 Wavelet Transform	14
3 HIGH LEVEL ALGORITHM DESCRIPTION	15
3.1 Interpolation	16
3.2 Delay Coordinates	16
3.3 Noise Reduction	17
3.3.1 Cosine Filtering	17
3.3.2 Wavelet Filtering	18
3.4 Local Linear Prediction Algorithm	19
3.5 Radial Basis Functions	23
3.6 Measuring the Error	25
4 NUMERICAL RESULTS	26
4.1 Filtering techniques	27

4.2	Linear function	28
4.3	Quadratic function	29
4.4	Sin(x)	30
4.5	xsin(x)	31
4.6	Lorenz function	32
4.7	Seismic data #4	33
4.8	Seismic data #143	34
5	CONCLUSIONS	44
	BIBLIOGRAPHY	47
	APPENDIX A: DATA SETS	49
	APPENDIX B: IMPLEMENTATION	52

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Erik Van Vleck, my thesis advisor, for all his ideas and continued support over the past two years. I pursued my degree at Colorado School of Mines specifically to study under Dr. Erik Van Vleck. However, it is due to the collective efforts of all faculty and students that I have this opportunity to present my thesis. In particular, I also wish to thank my committee members, Dr. Graeme Fairweather and Dr. Luis Tenorio for their interest in my endeavors.

LIST OF TABLES

3.1	Radial Basis Functions	24
4.1	Comparison of MSE for prediction of a linear function.	36
4.2	Comparison of MSE for prediction of quadratic functions.	36
4.3	Comparison of MSE for prediction of a sinusoid.	37
4.4	Comparison of MSE for prediction of $x \sin x$ function.	37
4.5	Comparison of MSE for prediction of a Lorenz function.	37
4.6	Comparison of MSE for prediction of seismic data no.4.	38
4.7	Comparison of MSE for prediction of seismic data no.143.	38

LIST OF FIGURES

4.1	Seismic time series #27	38
4.2	Filtered seismic time series #27, using a cosine filtering technique. . .	39
4.3	Filtered seismic time series #27, using a wavelet filtering technique. .	39
4.4	Comparison of predictions of a linear function.	40
4.5	Comparison of predictions of a linear function using a multiquadric radial basis function, varying the constant.	40
4.6	Comparisons of predictions of a quadratic function.	41
4.7	Prediction of $x\sin x$ with too small embedding dimension.	41
4.8	Comparison of predictions of the $x\sin x$ function.	42
4.9	Comparison of prediction of the Lorenz function.	42
4.10	Comparison of prediction of seismic data No.4.	43
4.11	Comparison of prediction of seismic data No.143.	43

Chapter 1

INTRODUCTION

In this thesis we utilize two existing methods, a local linear method using nearest neighbors, and a global method using radial basis functions to predict from discrete, chaotic time series. Our contribution is the implementation of the local and global approaches to nonlinear prediction using Matlab. We provide on-line help and a modular format to the subroutines, which are intended to facilitate accessibility and extensibility. With each of the local and global methods, we compare their predictions with the true continuation. A measure of prediction accuracy is found by comparing the mean squared errors.

The possibility of chaotic systems, in which a deterministic system exhibits sensitive dependence on initial conditions arises in many fields. Time ordered data from chaotic systems are known as chaotic time series. Chaotic time series are found in financial and economic data, environmental processes, and medical diagnoses. The nonlinear systems determining the time series are commonly modeled using *dynamical systems*.

Definition 1.0.1 *A dynamical system is a deterministic mathematical prescription for evolving the state of a system forward in time.*

Linear modeling is a common time series analysis technique, based on the fact that the signal produced by a finite dimensional linear system comprises a finite number of frequencies, a necessary condition for successful prediction using linear models. Time series produced by chaotic dynamical systems do not have a finite number of frequencies, but rather a continuous Fourier spectrum. Since existing time series prediction techniques lack predictive power in this case, new techniques must be sought to predict chaotic time series. Two new techniques, one based on a local linear method, and a global method based on radial basis functions are implemented in this thesis to predict chaotic time series.

This thesis is organized as follows. Chapter 2 contains pertinent background information as well as a justification of the methods of reconstructing and modeling the state space. Reconstructing the state space begins with interpolating and filtering the data to form a dense and denoised data set. In addition to Fourier filtering, we utilize a wavelet filtering technique created by Donoho et al., [5, 6, 7], and implemented in a Matlab subroutine group, WaveLab. Utilizing the concept of delay coordinate embedding, introduced by Sauer in the early 1980's [13], we complete the reconstruction of the state space. Deterministic models for the data are formed using Sauer's local linear method [13], and Casdagli's global radial basis function approach [4]. The high level algorithms are addressed in Chapter 3, whereas the numerical analysis and results are presented in Chapter 4. To measure prediction accuracy of the nearest neighbor and radial basis function approaches for nonlinear prediction, we utilize the mean squared error. We also discuss prediction length and accuracy. The final chapter, Chapter 5, contains conclusions and ideas for future work. Two

appendices supplement the thesis: the first contains data sets used in prediction and the second is a complete listing of all code.

Chapter 2

BACKGROUND

This chapter contains background on the methods considered for prediction of time series, without introducing the prediction methods themselves. Therefore, this chapter serves as a reference for the user concerned with the prediction algorithm, but is essential for the user interested in the process. A dense and denoised data set is necessary for accurate predictions, therefore, a sparse time series must first be interpolated, and a noisy time series filtered. The Fourier and wavelet transforms, each used in a filtering process, are introduced in this chapter. Discussed in detail are delay coordinates that form the foundation of our ability to predict time series.

2.1 Interpolation

We rely on the assumption that the underlying dynamics of a time series are deterministic, that is, the future is entirely determined by the past. If the past is not well defined, as in a sparse data set, inaccurate predictions may result. Interpolation is therefore the first crucial step in the prediction process. Interpolation itself may produce errors, which may be minimized using higher order interpolation. Utilizing

piecewise linear interpolation and subsequently sampling the function, we create a dense data set which we assume to be accurate.

2.2 Delay Coordinates

In 1980, Packard [10] attributed the suggestion of the connection between the underlying dynamics of a system and the delay coordinates, or time lagged vectors of observable measurements for that system, to D. Ruelle. The following year, Takens [17] gave a mathematical proof of the legitimacy of delay coordinates, which Sauer et al., [15] strengthened. This section states Takens' and Sauer's theorems which justify the use of delay coordinates and explain their construction.

Definition 2.2.1 *The time series of a system is defined by a scalar sequence, $\{x_1, x_2, \dots, x_N\}$, where $\{x_1, x_2, \dots, x_N\}$ represents the observable measurements of a system sampled at regular intervals.*

Definition 2.2.2 *The time lagged vector, $\mathbf{a}_t = [x_t, x_{t-\tau}, \dots, x_{t-(m-1)\tau}]^T$, with time lag τ , which contains sufficient information to determine the future evolution of the system is known as the state of the system.*

Definition 2.2.3 *The minimum embedding dimension, m , is the minimum number of delay coordinates necessary to construct the state of the system.*

Definition 2.2.4 *A diffeomorphism, h , is a C^1 function $h : \mathbb{R}^m \rightarrow \mathbb{R}$ with a C^1 inverse.*

Using only a time series, we wish to reconstruct the state of the system. The idea is that \mathbf{a}_t evolves via a diffeomorphism h , and gives rise to a scalar $x_t \in \mathbb{R}$ such that

$$x_t = h(\mathbf{a}_t). \quad (2.1)$$

Definition 2.2.5 *The delay coordinate vector is given by*

$$\begin{aligned} \mathbf{y}_t &= [x_t, x_{t-\tau}, \dots, x_{t-(m-1)\tau}]^T \\ &= [h(\mathbf{a}_t), h(\mathbf{a}_{t-\tau}), \dots, h(\mathbf{a}_{t-(m-1)\tau})]^T. \end{aligned} \quad (2.2)$$

Definition 2.2.6 *The elements of the delay vector, $x_t, x_{t-\tau}, \dots, x_{t-(m-1)\tau}$ are defined as delay coordinates.*

Definition 2.2.7 *An invariant set, A , is a set such that any trajectory, $\mathbf{x}(t)$ that starts in A stays in A for all time.*

Definition 2.2.8 *An attractor is a closed set A with the following properties:*

1. *A is an invariant set;*
2. *A attracts all trajectories that start sufficiently close to it;*
3. *there is no proper subset of A that satisfies conditions 1 and 2.*

The state-space, or phase-space, of a k -dimensional system is parameterized by k variables. A point x in the phase-space corresponds to a state of the system, and, as the system evolves with time, x moves through the space, tracing out a curve. Systems with phase-space contraction are commonly characterized by the presence of attractors. If the attractor is a point, then the attractor is a set of dimension zero, and if the attractor is a closed curve, then the attractor is a set of dimension 1. However, the attracting set can be irregular, and have a fractional dimension.

Definition 2.2.9 *A fractal is an irregular attracting set with fractional dimension.*

Definition 2.2.10 *The motion of an attractor exhibits sensitive dependence on initial conditions, if two trajectories that start very close together rapidly diverge.*

Definition 2.2.11 *A strange attractor is an attractor that exhibits sensitive dependence on initial conditions.*

To calculate the dimension of such an attractor, we consider its box-counting dimension, D_0 .

Definition 2.2.12 *The box-counting dimension is defined by*

$$D_0 = \lim_{\epsilon \rightarrow 0} \frac{\ln M(\epsilon)}{\ln(1/\epsilon)}, \quad (2.3)$$

where the attracting set is the k -dimensional phase space to be covered by k -dimensional cubes of edge length ϵ , and $M(\epsilon)$ is the minimum number of such cubes needed to cover the set.

We would like the observed quantities of the dynamical system to have some overall structure.

Definition 2.2.13 *The intersection of a countable collection of open sets is defined to be a G_δ .*

Definition 2.2.14 *The distance in \mathbb{R}^k is defined by $d(x, y) = \|x - y\|$, $x, y \in \mathbb{R}^k$ where $\| \cdot \|$ is defined to be the euclidean norm.*

Definition 2.2.15 *A neighborhood of a point $p \in X$ is a set $N_r(p)$ consisting of all points q such that $d(p, q) < r$. The number r is called the radius of $N_r(p)$.*

Definition 2.2.16 *A point in X is a limit point of the subset E of X , if every neighborhood of p contains a point $q \neq p$ such that $q \in E$.*

Definition 2.2.17 *Let X be a metric space. A subset E of X is dense in X if every point of X is a limit point of E , or a point of E , or both.*

Definition 2.2.18 *A set is called generic if it contains a dense G_δ .*

Definition 2.2.19 *A Hausdorff space is a topological space which satisfies the following property*

given two distinct points x and y , there are disjoint open sets O_1 and O_2 such that $x \in O_1$ and $y \in O_2$.

Definition 2.2.20 *A continuous map $h: \mathbb{R} \rightarrow \mathbb{R}$ with a continuous inverse is called a homeomorphism of \mathbb{R} .*

Definition 2.2.21 *An n -dimensional manifold is a connected Hausdorff space M such that each point has a neighborhood which is homeomorphic to a neighborhood in \mathbb{R}^n .*

Definition 2.2.22 *A space X is compact if every open covering U of X has a finite subcovering.*

Utilizing the definitions presented, Takens [17] justifies the embedding techniques to reconstruct the state space of a system from a time series where the attractor of the system defining the time series is an integer.

Theorem 2.2.1 [17] *If the dynamical system and observed quantities are generic, then the delay coordinate map from a d -dimensional smooth compact manifold to a $2d+1$ dimensional reconstruction space is a diffeomorphism, where d is an integer.*

Sauer [15] expands on the work of Takens to include both integer and fractal manifolds, with additional restrictions on the dynamical system.

Theorem 2.2.2 [15] *Assume that a continuous-time dynamical system has a compact invariant set A of box counting dimension D_0 , and let $m > 2D_0$. Let τ be the time delay, and m the minimum number of delay coordinates needed to guarantee that the reconstructed set in \mathbb{R}^m has the same dimension as the original state-space attractor. Assume that A contains a finite number of equilibria and a fixed number of periodic orbits of period $p\tau$ for $3 \leq p \leq m$ with no periodic orbits of period τ or 2τ . Then with probability one, there exists a measurement function h which yields a 1-1 function $h : A \rightarrow h(A)$.*

2.3 Dimension

In the previous section, we described a justification for using delay coordinates as long as the minimum embedding dimension, m , is greater than twice the dimension of the attractor. In this section, we describe a statistical method that we use to compute the minimum embedding dimension, which is necessary to construct delay coordinate vectors. This method eliminates the need for time lag, utilizing all sequential data.

Definition 2.3.1 *The set, S , of all possible outcomes of a particular experiment is called the sample space for the experiment.*

Definition 2.3.2 *A random variable is a function from a sample space S onto the real numbers.*

Definition 2.3.3 *If X and Y are discrete random variables, the probability that X will take on the value x and Y will take on the value y is written as $P(X=x, Y=y)$.*

Definition 2.3.4 *The function $f(x, y)$ is a joint probability mass function of the random variables X and Y if $f(x, y) = P(X = x, Y = y)$.*

Definition 2.3.5 *Let Y_1, \dots, Y_N , be discrete random variables. The multivariate probability distribution function, $p(y_1, \dots, y_N)$, for Y_1, \dots, Y_N is given by*

$$p(y_1, \dots, y_N) = P(Y_1 = y_1, \dots, Y_N = y_N), \quad (2.4)$$

defined for all real numbers y_1, \dots, y_N .

Definition 2.3.6 *The marginal distributions of X alone and of Y alone are*

$$g(x) = \sum_y f(x, y) \text{ and } h(y) = \sum_x f(x, y), \quad (2.5)$$

respectively.

Definition 2.3.7 *Let X and Y be two random variables, with joint probability mass function $f(x, y)$ and marginal distributions $g(x)$ and $h(y)$, respectively. The random variables X and Y are independent if and only if*

$$f(x, y) = g(x)h(y), \quad (2.6)$$

for all (x, y) within their range.

Definition 2.3.8 Let X be a discrete random variable with probability distribution $f(x)$. The mean or expected value of X is

$$\mu = E(X) = \sum_x x f(x). \quad (2.7)$$

Definition 2.3.9 Let X be a discrete random variable with probability distribution $f(x)$ and mean μ . The variance of X is

$$\sigma^2 = E[(X - \mu)^2] = \sum_x (x - \mu)^2 f(x). \quad (2.8)$$

Definition 2.3.10 Let X and Y be discrete random variables with joint probability distribution $f(x, y)$. The covariance of X and Y is

$$\sigma_{XY} = E[(X - \mu_X)(Y - \mu_Y)] = \sum_x \sum_y (x - \mu_X)(y - \mu_Y) f(x, y). \quad (2.9)$$

Definition 2.3.11 Let X be a discrete random variable with probability distribution $f(x)$ and mean μ . The positive square root of the variance is the standard deviation of X .

Definition 2.3.12 Let X and Y be random variables with covariance σ_{XY} and standard deviations σ_X and σ_Y , respectively. The correlation coefficient of X and Y is

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}. \quad (2.10)$$

Consider a time series of length N which we assume to be dense and denoised. Then we can create M -dimensional delay vectors, where M is the embedding dimension, $M \geq m > 2D_0$.

Definition 2.3.13 *A trajectory matrix, J , is a matrix whose rows consist of the M -dimensional delay vectors*

$$J = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N-M+1}]^T. \quad (2.11)$$

A trajectory matrix J contains the complete record of observations which have occurred in the dynamical system. The number of linearly independent columns of J determines the dimension $m \leq M$. The embedding may be described by a multivariate distribution whose variables are the M components of the embedding vectors. If (x, y) are independent random variables then $\text{cov}(x, y) = 0$ and $\rho(x, y) = 0$, where $\text{cov}(x, y)$ is the covariance and $\rho(x, y)$ is the correlation coefficient. The sample covariance matrix of the components of the \mathbf{x}_i averaged over the entire trajectory is defined by

$$C = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T. \quad (2.12)$$

The off-diagonal elements of the covariance matrix are the unnormalized correlation coefficients of the distribution. The correlation coefficients measure the linear statistical correlation of the $\mathbf{x}_i, i = 1, \dots, M$, with each other. We can write

$$C = J^T J. \quad (2.13)$$

Represent the $N - M + 1$ by M matrix, J , by its singular value decomposition (SVD),

$$J = U \Sigma V^T, \quad (2.14)$$

where U, V are orthogonal matrices, $\Sigma = \Sigma_{i,j} = 0$ for $i = 1, \dots, N - M + 1, j = 1, \dots, M, i \neq j$, and $\Sigma_{i,i} = \sigma_i$.

Definition 2.3.14 *The σ_i are the singular values of J such that $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_s \geq 0$, such that s is the minimum of $N - M + 1$ and M .*

Definition 2.3.15 *The σ_i^2 are the eigenvalues of $J^T J$.*

It is now obvious that the number of linearly independent row vectors in J is equivalent to the number of nonzero eigenvalues of $J^T J$, the rank of the covariance matrix or the number of nonzero singular values. Hence, we can utilize any of these properties to find the dimension of the smallest subspace of the embedding space that contains the reconstructed trajectory. Systems of real data inevitably contain noise, likely causing all singular values to be nonzero. Hence, we consider the minimum embedding dimension to be the number of “large” singular values of $J^T J$, that is, proportionally greater than the “small” singular values of $J^T J$. The proportion of “large” to “small” is a parameter which is dependent on the data as well as computer precision.

2.4 Cosine Transform

The discrete cosine transform is similar to the discrete Fourier transform: it transforms a signal from the spatial domain to the frequency domain. As the Fourier transform is a sum of sine and cosine waves, the cosine transform comprises the real part of the Fourier transform.

2.5 Wavelet Transform

The wavelet transform is an alternative to the Fourier transform which utilizes localized, rather than long, waves. Like the Fourier transform, the wavelet transform is a linear transformation that is invertible and orthogonal. It is also a transform from the time domain to a frequency domain. Instead of representing the signal as the sum of a finite number of frequencies, such as sine and cosine waves, the wavelet transform comprises waves that vanish outside a finite interval. Thus, the waves are localized. The differences between the infinitely many sets of wavelets are determined by how compactly they are localized as well as their smoothness.

Chapter 3

HIGH LEVEL ALGORITHM DESCRIPTION

This chapter consists of high level algorithms for interpolation, filtering, and delay vectors. Sufficient information is also provided to form an efficient approximation to

$$\mathbf{y}_t = [x_t, x_{t-\tau}, \dots, x_{t-m\tau}]^T, \quad (3.1)$$

or, equivalently, to find $P(\mathbf{b}) : \mathbb{R}^m \rightarrow \mathbb{R}^l$, where $P(\mathbf{b})$ is an estimate for the continuation of \mathbf{b} . Numerous methods for approximating (3.1) exist. Here local and global methods are discussed and their implementations described. Local functional forms have considerable flexibility in fitting functions of arbitrary complexity, but a dense and lengthy data set is required. In contrast, global predictors require much less data in constructing the single prediction function, but demand extensive computational resources. The local method presented here is based on nearest neighbors, whereas radial basis functions form the foundation of the global approach to approximation. To determine prediction accuracy, the mean squared error is employed.

3.1 Interpolation

We begin with a time series of length n , $\{x_1, x_2, \dots, x_n\}$. By linearly interpolating the time series, we can consider the dense time series to have length $N = s(n - 1) + n$ where s is the number of interpolating steps per sample period. Piecewise linear interpolation is chosen because of its simplicity and is sufficient for our applications.

3.2 Delay Coordinates

Consider the dense data set $\{x_1, x_2, \dots, x_N\}$. The delay matrix is formed from time lagged vectors of the original or interpolated time series. We set the time lag to 1, thus reducing the number of free parameters and utilizing all sequential data. If the attractor dimension is known a priori, it can be input into the subroutine directly. Otherwise, the dimension of the attractor can be calculated statistically, see section 2.3, creating a singular system with an upper bound pre-defined to determine the minimum embedding dimension. Define the singular values $sv(i)$, $i = 1, \dots, m$, where m is the upper bound on the dimension. We consider a ratio,

$$r = \frac{sv(i+1)}{sv(1)}, i = 1, \dots, m - 1.$$

If $r > .01$, we consider the singular value to be significant. This increases the minimum embedding dimension to $i+1$ where the value of i is the last value to make $\frac{sv(i+1)}{sv(1)} > .01$ true.

Therefore, for the time series of length N and minimum embedding dimension, m , we can create the delay matrix

$$M = \begin{bmatrix} x_m & x_{m+1} & \dots & x_N \\ x_2 & x_3 & \dots & x_{N-m+2} \\ x_1 & x_2 & \dots & x_{N-m+1} \end{bmatrix}.$$

3.3 Noise Reduction

Real data, regardless of how it is obtained, is likely to contain noise. In order to form a predictive model of the system, one must first recover the underlying signal, otherwise the predictive model may misrepresent the system. Theoretical results [15] show that, if a given set of data is noise free, the attractor A and its reconstruction $D(A)$ will have identical dynamical properties. Hence we must filter our data of noise prior to creating a prediction model. We describe filters based on the Fourier and wavelet transforms. Once the data is filtered, it is subsequently reconstructed into real-valued, filtered delay vectors by an appropriate inverse transform.

3.3.1 Cosine Filtering

The cosine filtering technique consists of three linear operations. First, take the cosine transform of each column of the delay matrix, where each column is of order, $2m$. Secondly, set to zero all but the lowest $m/2$ frequency contributions, and, third, take the inverse cosine transform of order m , using the $m/2$ available frequencies. The

discrete cosine transform is given by

$$y(k) = \sum_{n=1}^m 2x(n) \cos \left[\frac{\pi}{2m} k(2n+1) \right] \quad (3.2)$$

where m is the length of x , and x and y are the same size. Similarly, the inverse discrete cosine transform is given by

$$x(n) = \frac{1}{m} \sum_{k=0}^{m-1} w(k) y(k) \cos \left[\frac{\pi}{2m} k(2n+1) \right], \quad (3.3)$$

where

$$x(n) = \begin{cases} \frac{1}{2}, & k = 0, \\ 1, & 1 \leq k \leq m-1, \end{cases}$$

and m is the length of x , and x is the same size as y .

3.3.2 Wavelet Filtering

Let the function $f(t)$, $t \in [0, 1]$, be the function underlying the noise, and let the data, $x_i = f(t_i) + \sigma z_i$ have length $N = 2^{J+1}$. Here the t_i are equispaced and the z_i are white noise. Specific filters and thresholding techniques can be accessed through the Wavelab subroutine group; see [5, 6, 7] for further details. The wavelet filtering package can be outlined in three steps:

1. Apply a wavelet transform to the data, yielding noisy wavelet coefficients.
2. Apply a filter to the wavelet coefficients.
3. Set all wavelet coefficients at noise level to 0, invert the wavelet transform, thus producing the estimate $\hat{f}(t)$, $t \in [0, 1]$.

In contrast to Fourier filtering, The wavelet filtering technique has the effect of maintaining the height of spikes without broadening the peaks. However, no one filter-

ing technique can be judged superior to another because the ideal filtering technique greatly depends on the type of data to be analyzed.

Regardless of the type of methods used, producing a dense and denoised data set provides us an opportunity to form more accurate predictions than without interpolation or filtering.

3.4 Local Linear Prediction Algorithm

Consider a dense and denoised time series $\{x_1, x_2, \dots, x_N\}$. Let the reconstructed state space be known as the training set, and let $\mathbf{y}_i, i = 1, \dots, N - m$, be a representative state in the training set, where m is the minimum embedding dimension. Then, for each $\mathbf{y}_i, i = 1, \dots, N - m - 1$, we know the value $P(\mathbf{y}_i)$, the predicted value of \mathbf{y}_i , one time unit later. Our goal is to form $P(\mathbf{y}^*)$ accurately, where \mathbf{y}^* is the last value in the time series, yielding a prediction algorithm from the time series. Let \mathbf{y}^* be the state of the system from which we wish to predict. For a deterministic time series, we need only find a previous time for which the state was \mathbf{y}^* . In real world problems, noise, inaccurate data and inexact systems may prevent an exact \mathbf{y}_i from being known. Therefore, we use a linear combination of the k closest state space representations, $\{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_k\}$.

We choose the similar states, or the *neighbors*, with the restriction that only one neighbor be chosen from each trajectory. The failure to describe a sufficient number of trajectories leads to repetitive data representing the nearest neighbors, which may misrepresent the system.

Definition 3.4.1 *A trajectory is a subinterval of the time series.*

To define a trajectory from a time series with no local extrema, we set the i^{th} , $i = 1, \dots, N - M + 1$ trajectory, equal to the i^{th} , $i = 1, \dots, N - M + 1$ state space representation. For a function with local extrema, we define a trajectory to include all the state spaces, from one maxima to another. For example, suppose we have a time series $\{y_1, \dots, y_{25}\}$, with local maxima at $x=5$ and $x=20$. The first trajectory would include the interval $\{y_1, \dots, y_5\}$. The second interval would include the values $\{y_6, \dots, y_{20}\}$. The third trajectory would then include the remaining values of the time series.

Since the $\mathbf{b}_i, i = 1, \dots, k$, nearest neighbors were found from the training set, we know the values $P(\mathbf{b}_i), i = 1, \dots, k$. Using the data, we wish to estimate $P(\mathbf{y}^*)$. To estimate $P(\mathbf{y}^*)$, we wish to find a relationship between the neighbors, $\{\mathbf{b}_1, \dots, \mathbf{b}_k\}$, and $\{P(\mathbf{b}_1), \dots, P(\mathbf{b}_k)\}$. Note that $\mathbf{b}_i \in \mathbb{R}^m$, and $P(\mathbf{b}_i) \in \mathbb{R}^l$, for $i = 1, \dots, k$, and $l \leq m$.

Definition 3.4.2 Define $\Pi(X)$ to be the projection of X .

Since it is possible for these sets to be in different spaces, we project the data in \mathbb{R}^m , that is $\{\Pi(\mathbf{b}_1 - \mathbf{c}), \dots, \Pi(\mathbf{b}_k - \mathbf{c})\}$ onto the space \mathbb{R}^l thus enabling us to compare similar objects, where \mathbf{c} is the center of mass of the nearest neighbors and $\Pi : \mathbb{R}^m \rightarrow \mathbb{R}^l$ is the projection onto the best \mathbb{R}^l .

To find the best \mathbb{R}^l , define the matrix A whose rows consist of the $\mathbf{b}_i - \mathbf{c}$, $i = 1, \dots, k$; thus A is $k \times m$. Represent A by its singular value decomposition, $A = U\Sigma V^T$, where U, V are orthogonal and $\Sigma = \Sigma_{i,j}$ is a $k \times m$ matrix with $\Sigma_{ij} = 0$ for $i \neq j$ and $\Sigma_{ii} = \sigma_i$, with $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_s \geq 0$, where $s = \min\{k, m\}$.

Definition 3.4.3 The \mathbf{u}_i are the associated left singular vectors A .

Definition 3.4.4 *The \mathbf{v}_i are the associated right singular vectors of A .*

Definition 3.4.5 *A vector \mathbf{v} is said to be linearly dependent on the set of vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ if and only if \mathbf{v} can be written as some linear combination of $\mathbf{v}_1, \dots, \mathbf{v}_n$; otherwise, \mathbf{v} is said to be linearly independent of the set of vectors.*

Definition 3.4.6 *A basis for a vector space V is a linearly independent spanning set for V .*

Definition 3.4.7 *A set S of vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ in V is said to span some subspace V_0 of V iff S is a subset of V_0 and every vector \mathbf{v}_0 in V_0 is linearly independent on S ; S is said to be a spanning set for V_0 .*

Lemma 3.4.1 *The first k left singular vectors $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_k$ form an orthonormal basis for the column space of A . In other words, the column space of A is spanned by the first k columns of U .*

Lemma 3.4.2 *The first k right singular vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ form an orthonormal basis for the row space of A . In other words, the row space of A is spanned by the first k columns of V .*

Definition 3.4.8 *Let A be a $p \times q$ matrix. Then*

(a) the column space of A is the subspace of the vector space that is spanned by the set of columns of A .

(b) the row space of A is the subspace of the vector space of $1 \times q$ matrices that is spanned by the rows of A .

Definition 3.4.9 *The leading column for a row of a matrix is the column containing the first nonzero entry in that row.*

Definition 3.4.10 *A $p \times q$ matrix A is said to be in Gauss reduced form when there is an integer l with $0 \leq l \leq p$ for which the following is true*

- (a) the first l rows of A are nonzero, while the remaining $p - l$ rows are zero,*
- (b) the first nonzero entry in each nonzero row equals 1, and the column in which this occurs is a leading column.*
- (c) for the l leading columns, the leading column for each row is farther to the right than is the leading column for the row above.*

Theorem 3.4.1 *Let A have rank l . Then*

- (a) the column space of A has dimension l , and a basis for it consists of the set of columns of A corresponding to the leading columns in any Gauss-reduced form of A .*
- (b) the row space of A has dimension l , and a basis for it consists of the nonzero rows in any Gauss-reduced form of A .*

With $l = \text{rank}(A)$, known, it follows from Theorem 3.4.1 that the row space of A has dimension l . From Lemma 3.4.2, it follows that the first l right singular vectors form a spanning set for \mathbb{R}^l . This spanning set is the best subplane \mathbb{R}^l of \mathbb{R}^m that minimizes the distances to the neighbors. We then project the $\{(\mathbf{b}_1 - \mathbf{c}), \dots, (\mathbf{b}_k - \mathbf{c})\}$ onto the span of \mathbb{R}^l yielding a set in \mathbb{R}^l which we can compare to $\{P(\mathbf{b}_1), \dots, P(\mathbf{b}_k)\}$.

To find a relationship between these two sets, we fit a line to the determined values, $\{(\Pi(\mathbf{b}_1 - \mathbf{c}), P(\mathbf{b}_1)), \dots, (\Pi(\mathbf{b}_k - \mathbf{c}), P(\mathbf{b}_k))\}$. The linear model $L : \mathbb{R}^l \rightarrow \mathbb{R}$ is of the form

$$L(\mathbf{x}) = \mathbf{a} \cdot \mathbf{x} + d, \tag{3.4}$$

where \mathbf{a} is an l -vector and d is a constant scalar.

The prediction of \mathbf{y}^* is found by evaluating

$$X^* = L(\Pi(\mathbf{y}^* - \mathbf{c})), \quad (3.5)$$

where X^* is an estimate for $P(\mathbf{y}^*)$.

3.5 Radial Basis Functions

Consider a dense and denoised time series $\{x_1, x_2, \dots, x_N\}$ and let $\{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{N'}\}$ be the reconstructed state space of the system, where $N' = N - m - 1$. A global method for time series prediction is formed from the model

$$s(\mathbf{y}^*) = \sum_{i=1}^{N'} \lambda_i \phi(\|\mathbf{y}^* - \mathbf{y}_i\|), \quad \mathbf{y}^* \in \mathbb{R}^m. \quad (3.6)$$

where ϕ is a function from \mathbb{R}^+ to \mathbb{R} , $\|\cdot\|$ represents the Euclidean norm on \mathbb{R}^m , and $\phi(\|\mathbf{y}^* - \mathbf{y}_j\|)$ is a *radial basis function* with center \mathbf{y}_j . The model s is thus a linear combination of radial basis functions, $\phi(\|\mathbf{y}^* - \mathbf{y}_i\|)$ with a weighting function, λ_i .

Radial basis functions are a class of functions which are ideal for interpolating multivariate data, and which provide exact representations for spherically symmetric data. Note the value of m has no effect on this approximation. Table 3.5 contains common radial basis functions where $r \geq 0$, and c is a positive constant.

Common Radial Basis Functions	
$\phi(r) = r$	Linear
$\phi(r) = r^3$	Cubic
$\phi(r) = r^2 \log r$	Thin plate spline
$\phi(r) = e^{-r^2}$	Gaussian
$\phi(r) = (r^2 + c^2)^{\frac{1}{2}}$	Multiquadric
$\phi(r) = (r^2 + c^2)^{-\frac{1}{2}}$	Inverse multiquadric

Table 3.1: Radial Basis Functions

To determine the coefficients, $\{\lambda_1, \lambda_2, \dots, \lambda_{N'}\}$ defined by the conditions, $s(\mathbf{y}_i) = x_{m+i} = d_i$, $i = 1, 2, \dots, N'$, we solve the linear system:

$$\begin{bmatrix} R_{1,1} & R_{1,2} & \dots & R_{1,N'} \\ R_{2,1} & R_{2,2} & \dots & R_{2,N'} \\ \vdots & \vdots & \ddots & \vdots \\ R_{N',1} & R_{N',2} & \dots & R_{N',N'} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{N'} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{N'} \end{bmatrix}, \quad (3.7)$$

where $R = (R_{i,j})$, with $R_{i,j} = \phi(\|\mathbf{y}_i - \mathbf{y}_j\|)$, $i, j = 1, 2, \dots, N'$. The strict interpolation approach of using radial basis functions for nonlinear modeling determines all the coefficients of a linear combination of radial basis functions, where the number of coefficients is one less than the number of data points. For nonsingular R , this technique is accurate, but requires $O(N'^3)$ operations. The matrix R is factored using Gaussian elimination and the factors are then used to solve the system of equations. In the

case of a singular R , repeated delay vectors are deleted, as well as their corresponding functions, $s(y_i)$. The nonsquare matrix R is then factored using Householder orthogonalization with column pivoting. The factors are then used to solve the system in a least squares sense.

3.6 Measuring the Error

The methods discussed in the previous sections have outlined techniques for prediction of time series. Since we can apply any of these techniques in a particular situation, we must choose the best prediction. To measure the prediction accuracy, we will utilize the *mean squared error* (MSE) which measures the average squared difference between the predicted value and the true value, and is defined in the following way

Definition 3.6.1 *Let N be the number of training pairs, and N_p be the number of predictions. If x_i , $i = 1, 2, \dots, N, \dots, N_p$, are the real values of the output and \hat{x}_i , $i = 1, 2, \dots, N, \dots, N_p$ are the predictions for x_i , then*

$$MSE = \frac{1}{N_p - N} \sum_{i=N+1}^{N_p} (\hat{x}_i - x_i)^2. \quad (3.8)$$

Chapter 4

NUMERICAL RESULTS

In this chapter, we present the results and analysis of numerical experiments. All the computations were obtained using Matlab on a Silicon Graphics workstation. We begin with a discussion on the results of the filtering process, which is the primary step to understanding the underlying dynamics of a system. This process is independent of the prediction routines and deserves individual attention. We continue with the numerical results, beginning with data sets extracted from simple functions. We progress to more complicated functions and finally apply the algorithms to seismic data.

The individual processes of the prediction algorithm are not highlighted, however, the parameters are detailed in tables. The main comparison is made between the local and global methods of prediction. The parameters which are varied within the local or global prediction routines include the length of the delay, the type of filtering, the number of interpolation steps per sampling period, and the coefficient of the multiquadric radial basis function. Two parameters are not varied in these experiments: in the local routine, the number of nearest neighbors is chosen to be four, and in the global routine, the radial basis function is always chosen to be multiquadric.

When reading the tables, interpolation refers to the number of interpolation steps per sample period; see sections 2.1 and 3.1. Filter refers to the type of filtering. The abbreviation C:win- x ,dim- y refers to a cosine filtering technique with window length, x , and embedding dimension, y ; see section 2.4, and 3.3.1. The abbreviation wvt- x , y refers to the wavelet filtering technique with filter type, x , and threshold type, y ; see section 2.5 and 3.3.2. Dim refers to the embedding dimension, and Model refers to the local or global method for prediction. The abbreviation $4nn$ refers to the local algorithm with four nearest neighbors. The abbreviation $rbf = 6, c = y$, refers to the global prediction algorithm, with radial basis function equal to the 6th on the list, or multiquadric, with the constant c chosen to be y . Finally MSE refers to the mean squared error of the prediction.

4.1 Filtering techniques

In Figure 4.1, we present data set #27, which was obtained from a seismic event, sampled at equal intervals [16]. Figures 4.2 and 4.3 show filtered values of the original time series. The filtered values shown in Figure 4.2 arose from a cosine filtering technique with window length of 16, and embedding dimension of 8. The realized values of the wavelet filtering technique are shown in the Figure 4.3, where a Symmlet-8 transform and Visu thresholding is used. Both filtering techniques retain the general curve of the actual data. In comparing these filtering techniques, note that the wavelet filtering technique produces a smoother curve and the values at the peaks are closer to the original data set, as compared to the cosine filtering technique. Which technique is preferred depends on factors including the error in obtaining the original data and

the purpose of the filtering.

4.2 Linear function

The discrete data set obtained for this set of experiments was created using the function $y(x) = x$, where the domain is given by $x(i) = i, i = 1, \dots, 32$. The minimum embedding dimension needed to represent the state space equals two. We calculate the minimum embedding dimension by first considering the maximum embedding dimension to be four. We then statistically calculate the minimum embedding dimension; see section 2.3. The singular values equal $4.2075e+02$, $7.3655e+00$, $4.5914e-14$, and $8.8752e-15$. Our algorithm considers the ratio, r , of singular values to find the number of “large” singular values, and concludes that the first two singular values are significant compared to the last two values. Thus, the minimum embedding dimension is two. Upon examination of the data and knowledge of the function, we know the data set represents the function well. Therefore, interpolation of the data set is unnecessary. We also know the function creating the data set produces exact, integer values. In other words, the data set does not contain any noise, therefore, filtering is unnecessary.

Figure 4.4 illustrates the actual continuation with a solid line, along with the best predictions obtained using nearest neighbors (*nn*), represented by a dashed line, and radial basis functions (*rbf*), represented by a dotted line. By best we mean the predictions with the smallest mean squared errors. The first trial uses a local prediction algorithm with delay set to two, and four nearest neighbors. The prediction yields an accurate continuation. To choose the nearest neighbors, we let a trajectory be

defined by one time step. A few more trials are conducted with larger embedding dimensions, with the same results. The multiquadric radial basis function prediction is shown with an optimal constant value equal to 25. The first trial of the global prediction algorithm utilizes a small constant. We continue with more trials, increasing the constant each time, until the MSE reaches a minimum. Table 4.1 organizes all predictions using the linear data set.

Using the global prediction algorithm and the multiquadric radial basis function, Figure 4.5 shows how the change in the constant, c , alters the prediction of the linear data set. To optimize the constant c , we predict with a small value for the constant and continue to predict until the mean square error reaches a minimum. The value of c is found to have interesting characteristics. As c increases, the slope of the predicted values decrease, approaching the actual continuation from above. For a given delay value of 4, and $c > 10$, the slope of the predicted values continue to decrease, however, the line now lies below the desired continuation. The optimal c value is valid only for this dimension because as the embedding dimension increases, so does the optimal c value. Setting the embedding dimension to 10, and again utilizing a multiquadric rbf, the $c = 27$ is found to minimize the error.

4.3 Quadratic function

The discrete data set obtained for this set of experiments was created using the function $y(x) = (x/2)^2$, where the domain is given by $x(i), i = 1, \dots, 32$. For similar reasons to those given for the linear data, neither interpolation nor filtering is used. To calculate the minimum embedding dimension, we first consider the embedding

dimension to be four and statistically calculate the minimum embedding dimension. The singular values are $1.2034e+03$, $2.7538e+01$, $6.8878e-01$, and $3.8034e-14$. Using r to determine the “large” singular values, we conclude the minimum embedding dimension is 2. However, due to experimental observations, an embedding dimension of 4 yields better results. The local algorithm uses four nearest neighbors, where only one neighbor can be found per trajectory. For this function, we define a trajectory to be one time step. Results of predicting the quadratic function are shown in Figure 4.6 is shown by the dashed line, and the dotted line represents the multiquadric rbf approach with $c = 19$. The short term predictions for both the local and global methods remain on the actual continuation for approximately 9 iterations when they begin to grow at a faster rate. Mean squared errors obtained are given in Table 4.2. The global prediction algorithm outperforms the local approach for short term prediction, but as the prediction length increases the difference in the errors is greatly reduced.

4.4 Sin(x)

The discrete data set obtained for this set of experiments was created using the function $y(x) = \sin(\pi x/60)$ where the domain is given by $x(i) = i, i = 1, \dots, 120$. We first consider the embedding dimension to be 10, and statistically calculate the minimum embedding dimension. The singular values are calculated to be 62.8399, 7.0361, 0.0000, 0.0000, ... Using r , we conclude the first two singular values are significantly greater than the remaining eight. Therefore, statistically, the embedding dimension is 2. However, numerical experiments show an embedding dimension of 4 yields the best

results. The local algorithm using four nearest neighbors accurately predicts the continuation. The four neighbors were chosen from separate trajectories, where a trajectory was defined by one period. The global algorithm using a multiquadric radial basis function with a constant $c = .2$, misses the peak by approximately $6.323e-03$ and therefore has a small error. No visual difference can be seen between the curves for the predictions of the data produced from the sinusoidal function and the actual continuation. Therefore we do not show either of the predictions. The parameters and mean squared errors appear in Table 4.3.

4.5 $x\sin(x)$

The discrete data set obtained for this set of experiments was created using the function $y(x) = \pi x/8 \sin(x/30)$ where the domain is given by $x(i) = i, i = 0, \dots, 80$. As the data sets increase in complexity, we want to be sure the we have a sufficient, denoised data string. Therefore, we increase the interpolation to two steps per sample period and implement filtering routines. An increase in data and noise reduction allows us to represent the state more accurately. For comparison purposes, we use one type of filtering for both the local and global prediction routines. Because of the increase in complexity, the dimension of the attractor now has a larger effect on the outcome of the prediction. The embedding dimension must be chosen large enough to capture the state space.

Figure 4.7 illustrates the prediction of the function $x \sin x$ where the dimension is chosen to be 27. This dimension is too small to capture the state space. Therefore, this nearest neighbor approach cannot predict the change in amplitude. We increase

the embedding dimension until our results captures the variation of amplitude. The optimal minimum embedding dimension is 72, as is shown in Figure 4.8. The dashed line represents the local prediction with four nearest neighbors, where a trajectory is defined to be the distance from one maximum to another. The global prediction with a multiquadric radial basis function with constant $c = 40$ is represented using a dotted line.

Both prediction routines produce obvious errors for the $x \sin x$ function. Two shifts exist in the local predictions, and as the dimension increases, the problem is amplified. In Figure 4.8, the curve shifts around $x = 520$. On the other hand, the rbf approach accurately reconstructs the period, but fails to reach the proper amplitude. Although each produces obvious errors, with the shift seeming the most simple to explain, the global approach outperforms the local prediction routine, as is shown in Table 4.4.

4.6 Lorenz function

The discrete data set obtained for these experiments was obtained by integrating the system of differential equations,

$$dx/dt = \sigma(y - x), \quad (4.1)$$

$$dy/dt = rx - y - xz,$$

$$dz/dt = -bz + xy,$$

using a fourth and fifth order embedded Runge-Kutta pair, over the interval $t(i) = i, i = 0, .05, .10, \dots, 39$. The coefficients of the differential equations are defined by

sample period, add a cosine filtering technique with window length 32 and embedding dimension of 16 to create a sufficient, denoised data string. The minimum embedding dimension is calculated statistically to be 12, however, a larger embedding dimension yields more accurate results. For the local prediction, we use four nearest neighbors, with only one neighbor being chosen from a trajectory. We define a trajectory to be the distance between maxima. The global method of prediction utilizes a multiquadric radial basis function with constant $c = 5$.

The prediction of the Lorenz data using both the local and global approach is shown in Figure 4.9, with accompanying error analysis in Table 4.5. Note that both predictions reproduce the shape of the actual continuation, but fail to predict the phase and amplitude accurately. The continuation of these curves over 600 steps shows the inherent determinism of these algorithms, where the curves become periodic and stray from the actual continuation.

4.7 Seismic data #4

The data set #4 used for this experiment was obtained from a seismic event, sampled at equal intervals [16]. Piecewise linear interpolation with 2 steps per sampling period is applied to the seismic data of length 1000, and then filtered using a cosine filtering technique with window length 16 and embedding dimension 8. Statistically, the embedding dimension is calculated to be 8, which is sufficient. The data acquired from a real seismic event, even when denoised, still attains many extrema over “small” intervals. Therefore, to describe a trajectory, we consider only “large” maxima.

Predictions are shown in Figure 4.10 with their parameters and errors located in

Table 4.6. The local method, initially predicts the downward trend, but, predicts the values prematurely, and then continues with inaccurate predictions. The global method, which uses a multiquadric radial basis function with $c = 10$ outperforms the local method, predicting many of the peaks and valleys, but with inaccurate amplitude and phase.

4.8 Seismic data #143

The data set #143 used for this experiment was obtained from a seismic event, sampled at equal intervals [16]. Piecewise linear interpolation with 2 steps per sampling period is applied to the seismic data of length 1024, and then filtered using a wavelet filtering technique with a symmlet-8 transform and Visu thresholding. To determine a minimum embedding dimension, we bound the dimension at 30 and calculate the number of “large” singular values to be 8. We utilize this dimension in the local prediction routine, but find a dimension of 16 is better suited to the global prediction algorithm. As with the previous data set, even when this data is denoised, it still attains many extrema over “small” intervals. Therefore, to describe a trajectory, we consider only “large” maxima.

The prediction of seismic data #143 is shown in Figure 4.11, with the parameters and error results shown in Table 4.7. The predictions found using the local method with four nearest neighbors seem to follow an average value of the path. In comparison, the global method using a multiquadric radial basis function with $c = 27$ depicts the actual continuation with a horizontal shift to the left by approximately 30 units. Both the local and global methods have relatively equal errors, although, visually,

the global method appears to be a better prediction.

Function	Interpolation	Filter	Dim	Model	Length	MSE
Linear	No	No	4	4nn	20	0.0
Linear	No	No	4	rbf=6, c=.5	20	2.4426e+02
Linear	No	No	4	rbf=6, c=4	20	3.9385e+01
Linear	No	No	4	rbf=6, c=10	20	3.5595e-01
Linear	No	No	4	rbf=6, c=15	20	4.0488e+00
Linear	No	No	4	rbf=6, c=25	20	1.6350e-05
Linear	No	No	10	rbf=6, c=.1	20	8.3852e+00
Linear	No	No	10	rbf=6, c=27	20	4.6722e-03

Table 4.1: Comparison of MSE for prediction of a linear function.

Function	Interpolation	Filter	Dim	Model	Length	MSE
Quadratic	No	No	4	4nn	10	9.4318e-01
Quadratic	No	No	4	4nn	20	3.0221e+02
Quadratic	No	No	4	4nn	30	2.9670e+03
Quadratic	No	No	4	rbf=6, c=19	10	2.6160e-04
Quadratic	No	No	4	rbf=6, c=19	20	3.3678e+01
Quadratic	No	No	4	rbf=6, c=19	30	6.4546e+02

Table 4.2: Comparison of MSE for prediction of quadratic functions.

Function	Interpolation	Filter	Dim	Model	Length	MSE
$\sin x$	No	No	4	4nn	200	0.0
$\sin x$	No	No	4	4nn	400	0.0
$\sin x$	No	No	4	rbf=6, c=.2	200	1.1207e-05
$\sin x$	No	No	4	rbf=6, c=.2	400	1.7671e-05

Table 4.3: Comparison of MSE for prediction of a sinusoid.

Function	Interpolation	Filter	Dim	Model	Length	MSE
$x \sin x$	2	No	72	4nn	100	4.3183e+01
$x \sin x$	2	No	72	4nn	200	2.7235e+02
$x \sin x$	2	No	72	rbf=6, c=40	100	6.0549e+00
$x \sin x$	2	No	72	rbf=6, c=40	200	1.9387e+01

Table 4.4: Comparison of MSE for prediction of $x \sin x$ function.

Function	Interpolation	Filter	Dim	Model	Length	MSE
Lorenz	2	C: win-32, dim-16	16	4nn	400	2.7199e+00
Lorenz	2	C: win-32, dim-16	16	4nn	600	3.4840e+00
Lorenz	2	C: win-32, dim-16	16	rbf=6, c=5	400	6.3091e+01
Lorenz	2	C: win-32, dim-16	16	rbf=6, c=5	600	5.7444e+01

Table 4.5: Comparison of MSE for prediction of a Lorenz function.

Function	Interpolation	Filter	Dim	Model	Length	MSE
seismic 4	2	C: win-16, dim-8	8	4nn	200	6.6709e+01
seismic 4	2	C: win-16, dim-8	8	4nn	400	1.9152e+02
seismic 4	2	C: win-16, dim-8	8	rbf=6, c=10	200	6.0101e+01
seismic 4	2	C: win-16, dim-8	8	rbf=6, c=10	400	9.6027e+01

Table 4.6: Comparison of MSE for prediction of seismic data no.4.

Function	Interpolation	Filter	Dim	Model	Length	MSE
seismic 143	2	wvt:sym-8, Visu	8	4nn	200	7.1234e+02
seismic 143	2	wvt:sym-8, Visu	8	4nn	400	1.5471e+03
seismic 143	2	wvt:sym-8, Visu	16	rbf=6, c=27	200	7.4968e+02
seismic 143	2	wvt:sym-8, Visu	16	rbf=6, c=27	400	7.7975e+02

Table 4.7: Comparison of MSE for prediction of seismic data no.143.

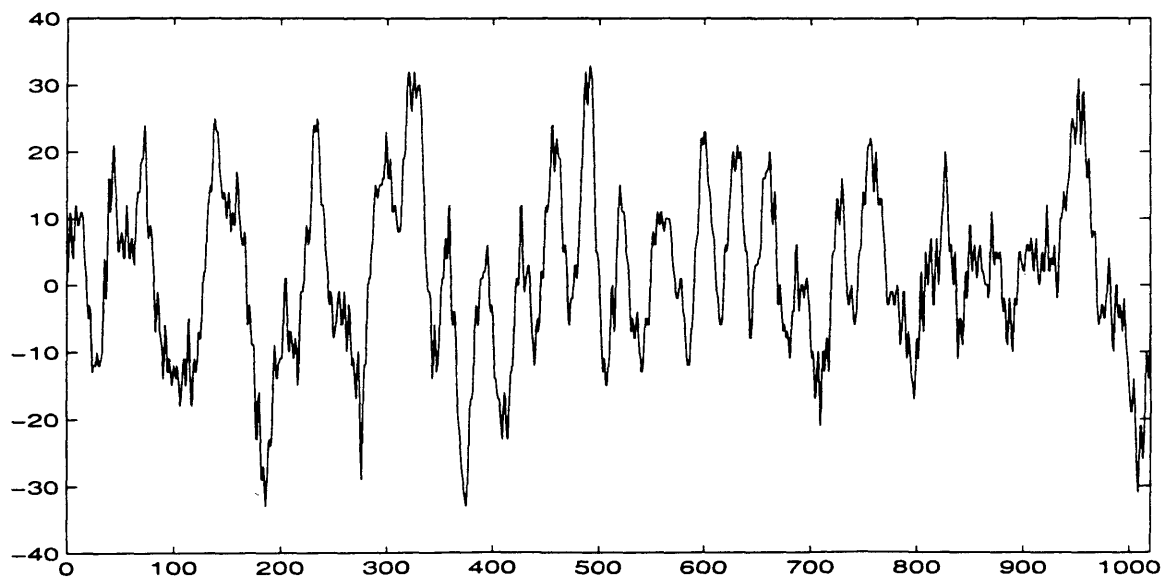


Figure 4.1: Seismic time series #27

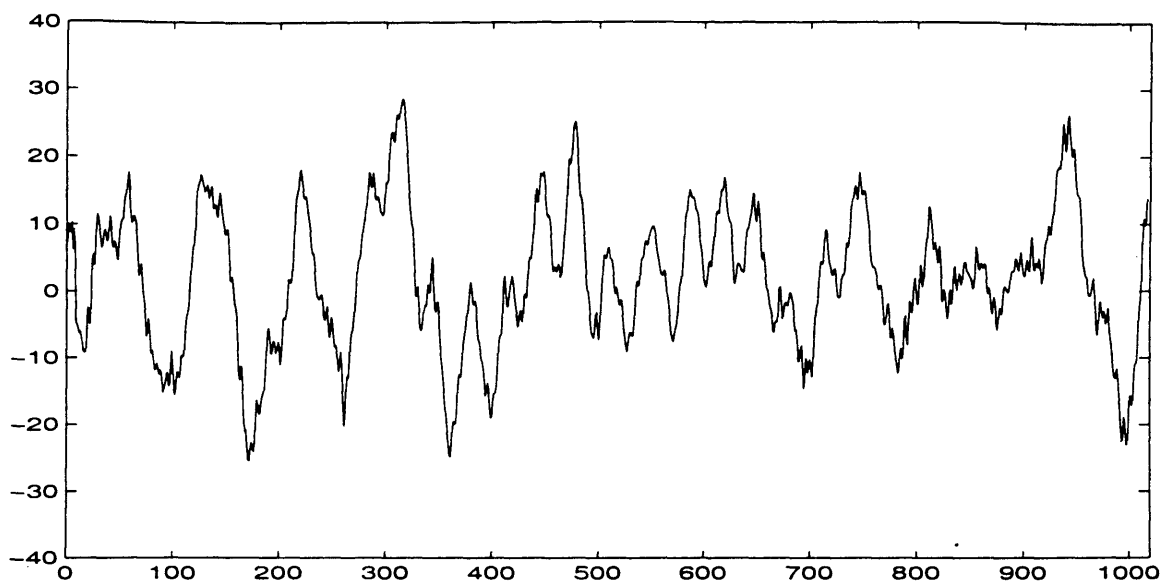


Figure 4.2: Filtered seismic time series #27, using a cosine filtering technique.

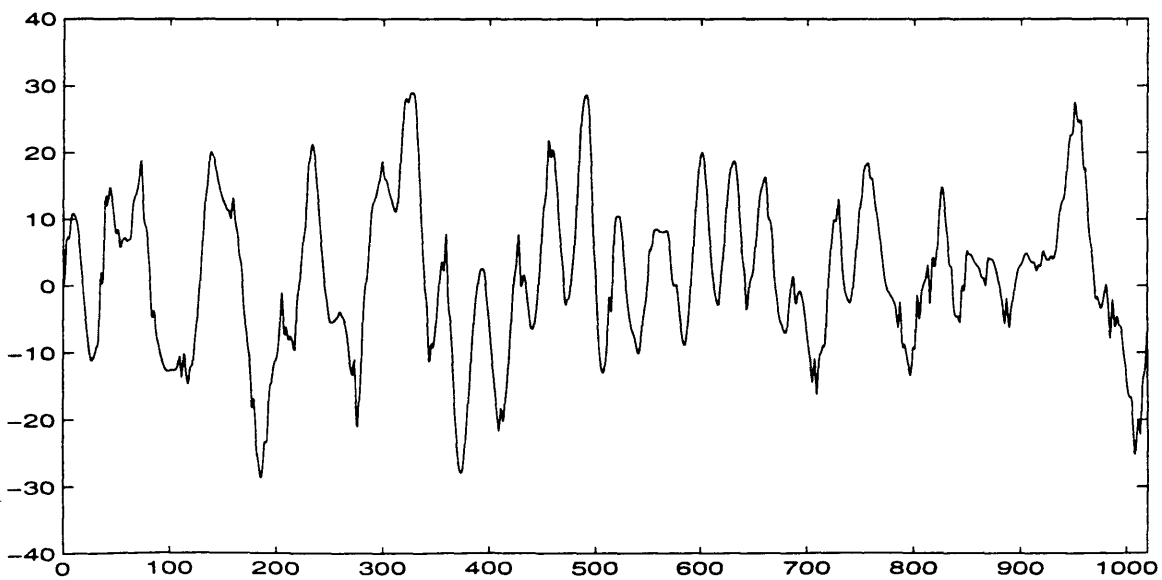


Figure 4.3: Filtered seismic time series #27, using a wavelet filtering technique.

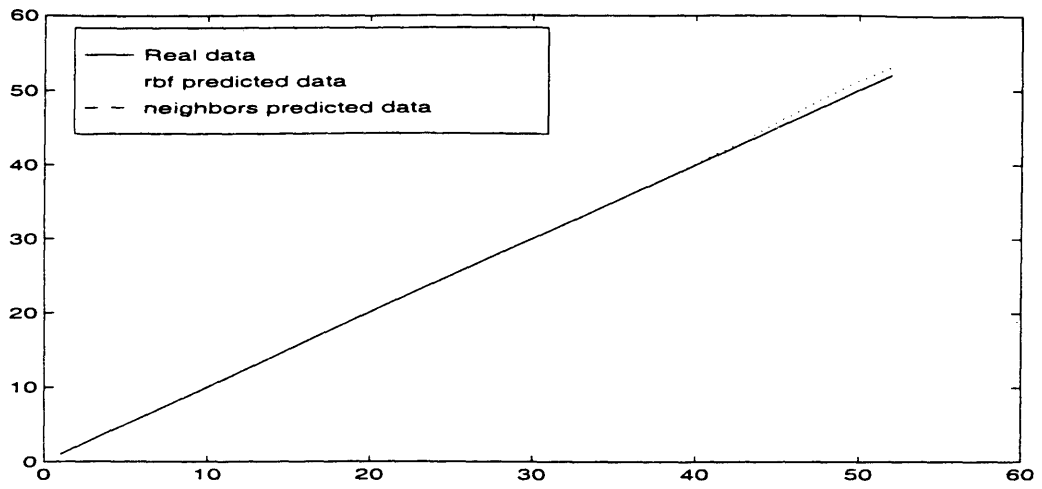


Figure 4.4: Comparison of predictions of a linear function.

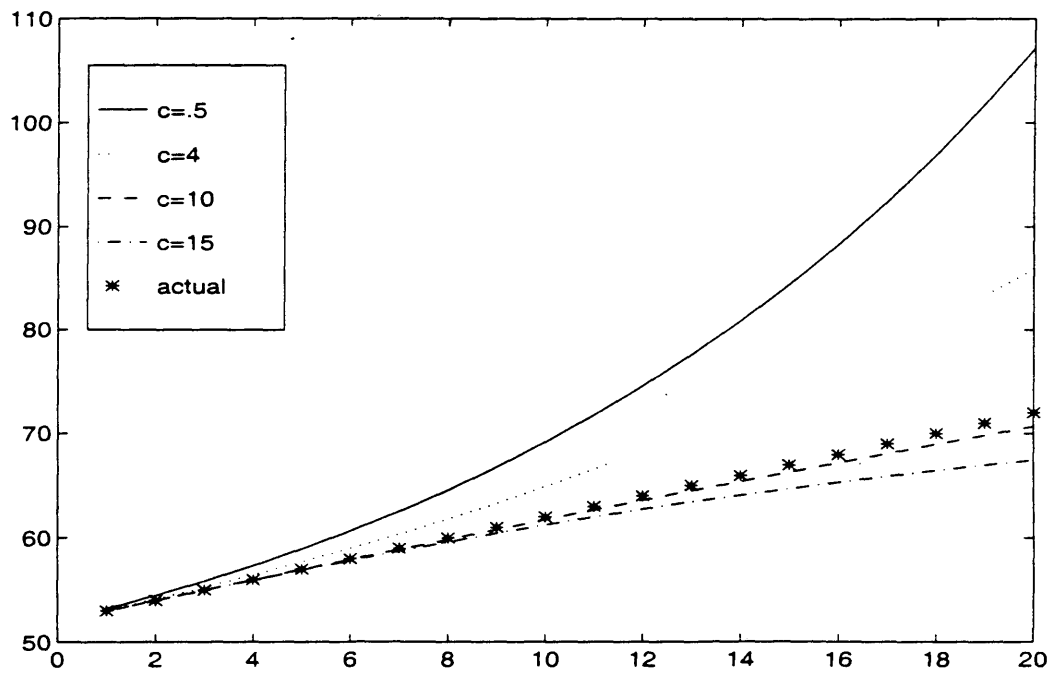


Figure 4.5: Comparison of predictions of a linear function using a multiquadric radial basis function, varying the constant.

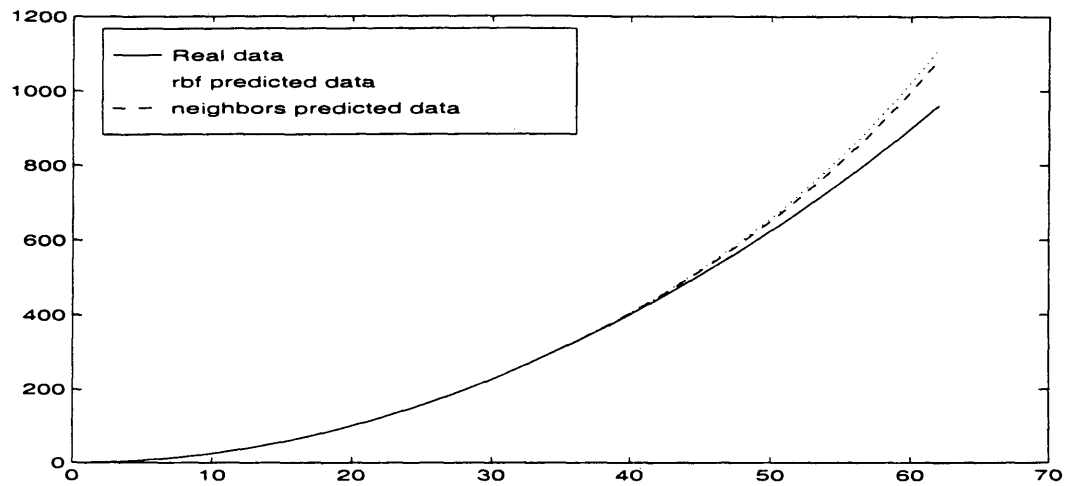


Figure 4.6: Comparisons of predictions of a quadratic function.

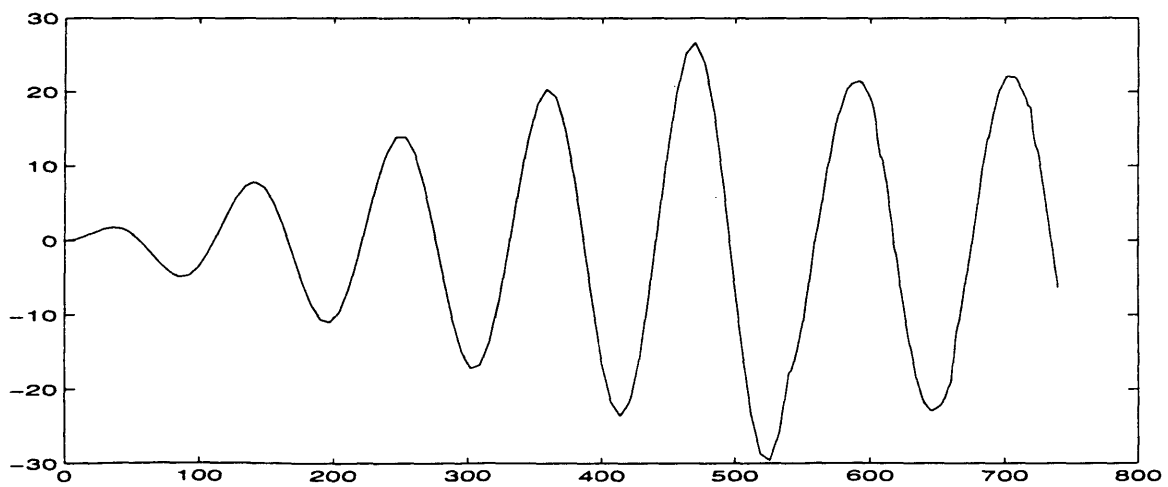


Figure 4.7: Prediction of $x \sin x$ with too small embedding dimension.

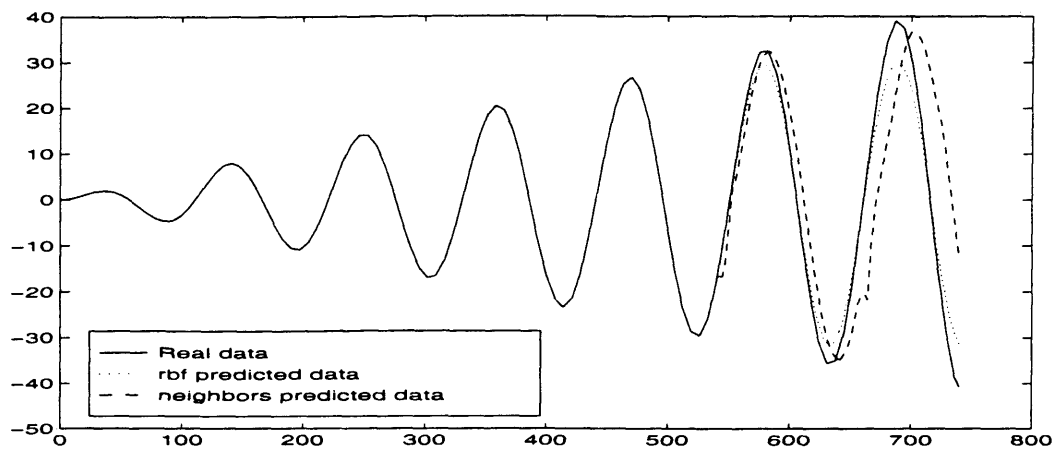


Figure 4.8: Comparison of predictions of the $x \sin x$ function.

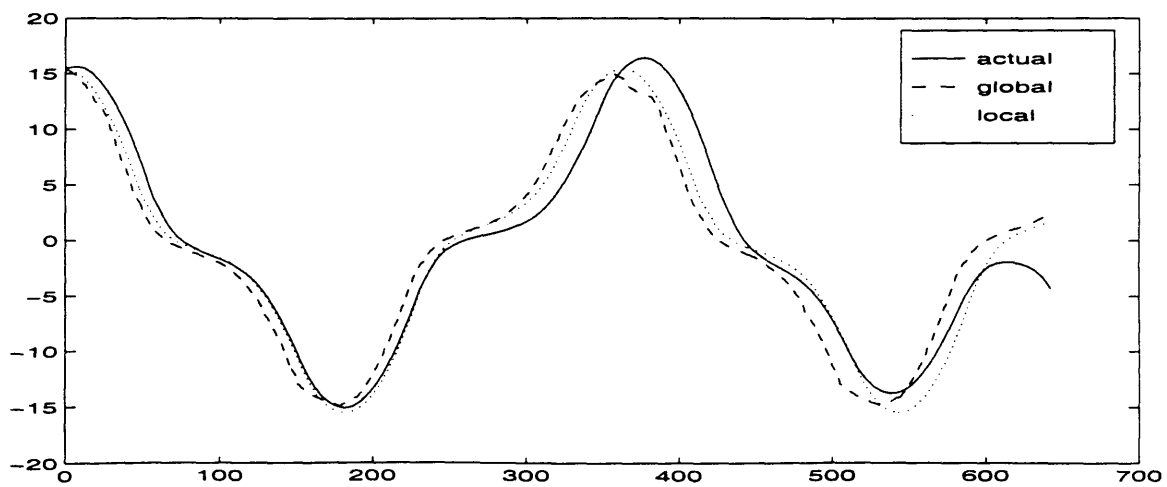


Figure 4.9: Comparison of prediction of the Lorenz function.

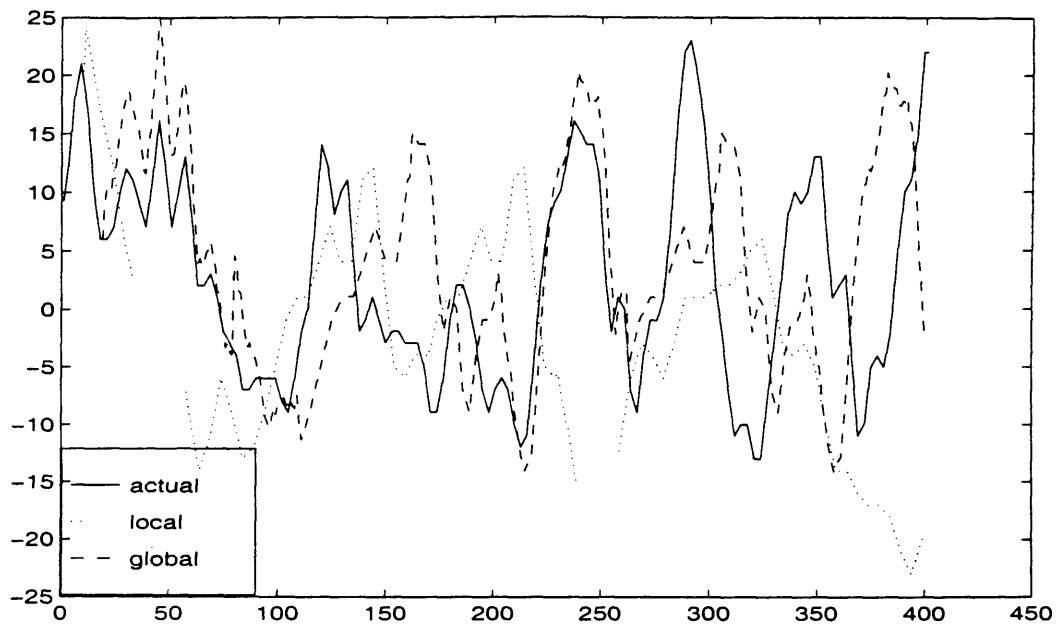


Figure 4.10: Comparison of prediction of seismic data No.4.

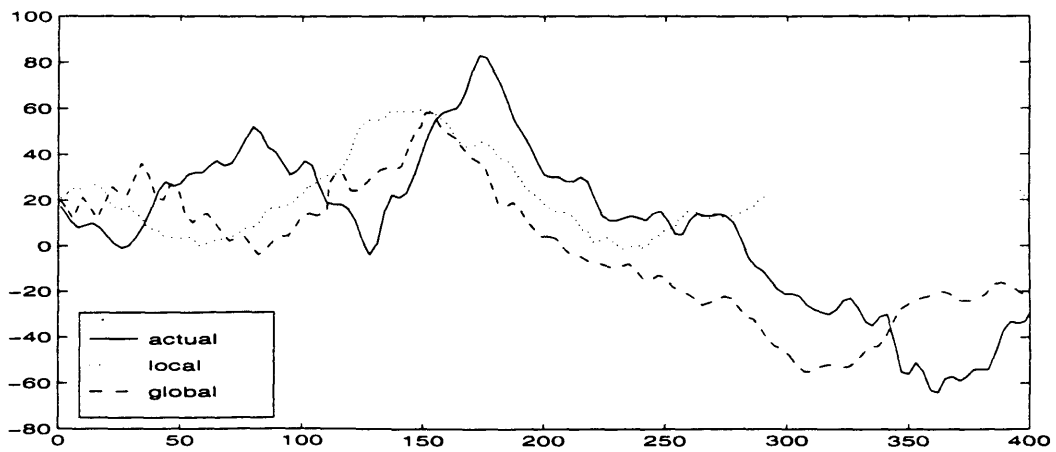


Figure 4.11: Comparison of prediction of seismic data No.143.

Chapter 5

CONCLUSIONS

This chapter concludes the thesis with a summary of what we accomplished, conclusions drawn from experiments, and ideas for future work on this topic.

In this thesis, we have focused on two methods for prediction of chaotic time series. We exploit the deterministic characteristics of these systems by using the observed behavior of a system to predict behavior when similar conditions occur. The methods of a nearest neighbor and radial basis function approach have been programmed in Matlab for user implementation.

We conclude that for simple data strings, both the local and global methods yield near accurate predictions. As the time series increases in complexity, the global method of prediction outperforms the local method. However, the difference in accuracy is small. Overall, the global method outperforms the local method four out of seven times in the sense of mean squared error. However, the successful implementation of either algorithm depends on good choices of the parameters, of which more research is necessary.

The parameter with the greatest influence on both prediction routines is the embedding dimension. For the majority of experiments, a larger embedding dimension

than that is calculated statistically yields more accurate results. For the multiquadric radial basis function, the value of the constant c has a great impact on the outcome of the global prediction method. An algorithm to optimize the constant c would improve the ease of use.

For non periodic data sets, we defined a trajectory to be the functional values from one “large” maxima to another. A general algorithm to describe trajectories would make prediction more systematic. Also, one may consider other measures of distance which may more accurately define a trajectory.

We concentrated on the multiquadric radial basis function, however, five other radial basis function are available for implementation. A correlation between the type of data set and the optimal radial basis function would narrow the number of predictions necessary to find the most accurate prediction.

The interpolation technique used to find the coefficients of the radial basis functions produces accurate results, but requires $O(N^3)$ operations. One could create a problem on linear optimization rather than linear interpolation. This would reduce the accuracy as well as the number of operations.

A suggested method to optimizing prediction would be to first split the time series into two sets. The first $n \leq N$ elements of the time series we consider a training set. It is with this set that we calculate a prediction. The remaining set of length $N - n$ is used to test the predictive ability of the learning set. We then can adjust the parameters to optimize the predicted values with the test set. This optimal method of prediction is then used with the entire time series to predict chaotic time series.

Both the local and global methods to prediction are based on the inherent deter-

minism of the system. However, it is possible that the problem may have too many degrees of freedom to reconstruct the state space. In this case, it may seem that the system is modeled more accurately by a stochastic process. But a model that combines both deterministic and stochastic process may best model the chaotic time series.

BIBLIOGRAPHY

- [1] A.M. Albano, J. Muench, C. Schwartz, A.I. Mees, P.E. Rapp, Singular-Value Decomposition and the Grassberger-Procaccia Algorithm, in *Coping with Chaos*, John Wiley & Sons, New York, 1994, pp. 107-116.
- [2] D.S. Broomhead, G.P. King, Extracting Qualitative Dynamics from Experimental Data, in *Coping with Chaos*, John Wiley & Sons, New York, 1994, pp. 72-91.
- [3] J. Buckheit, *About Wavelab*, Technical report, Department of Statistics, Stanford University, 1995.
- [4] M. Casdagli, Nonlinear Prediction of Chaotic Time Series, in *Coping with Chaos*, John Wiley & Sons, New York, 1994, pp. 212-233.
- [5] D.L. Donoho, Wavelet Shrinkage and W.V.D.: A 10-Minute Tour, *Gifsur-Yvette: Editions Frontieres* pp. 109-128.
- [6] D.L. Donoho, Nonlinear Wavelet Methods for Recovery of Signals, Densities and Spectra from Indirect and Noisy Data, *Proceedings of Symposia in Applied Mathematics* (1993) pp 173-205
- [7] D.L. Donoho, I.M. Johnstone, *Ideal Time-Frequency Denoising*, Technical Report, Department of Statistics, Stanford University, 1995.
- [8] X. He, A. Lapedes, Nonlinear Modeling and Prediction by Successive Approximation using Radial Basis Functions, *Physica D* **70** (1993) pp. 289-301.

- [9] C.A. Micchelli, Interpolation of Scattered Data: Distance Matrices and Positive Definite Functions, *Constructive Approximation* **2** (1986) pp. 11-22.
- [10] N.H. Packard, J.P. Crutchfield, J.D. Farmer, R.S. Shaw, Geometry from a Time Series, *Phys.Rev.Lett.* **45** (1982) pp. 712-715.
- [11] M.J.D. Powell, The Theory of Radial Basis Function Approximation in 1990, *Advances in Numerical Analysis* **2**, (Will Light, ed.), Clarendon Press, Oxford, 1992, pp. 105-210.
- [12] M.J.D. Powell, *Approximation Theories and Methods*, Cambridge University Press, New York, 1981.
- [13] T. Sauer, Time Series Prediction by Using Delay Coordinate Embedding, in *Coping with Chaos*, John Wiley & Sons, New York, 1994, pp. 241-256.
- [14] T. Sauer, J. Yorke, How Many Delay Coordinates Do You Need, *International Journal of Bifurcation and Chaos*, **3** (1993) pp. 737-744.
- [15] T.Sauer, J.A. Yorke, M. Casdagli, Embedology, *J.Stat.Phys.* **65** (1991) pp. 579-616.
- [16] Iris Passcal Instruments Center and 3-D Geophysical.
- [17] F. Takens, Detecting Strange Attractors in Turbulence, *Lecture Notes in Mathematics* **898**, Springer, Berlin, 1981, pp. 366-381.

APPENDIX A: DATA SETS

The data files made available for this thesis, some of which are implemented for prediction, are contained in this appendix, along with the size, description, and source of the data file. Also contained in this appendix are descriptions of the five functions which are options that the user can choose to create a time series. Note that this list is not the only data sets which can be implemented in these routines. The user has the option of entering his/her own data as well as a unique function.

Ascii files

CARUSO:

Size: 50,000 by 1.

Description: The digital signal of Caruso singing.

Source: [5]

ESCA:

Size: 1024 by 1.

Description: It is an ESCA signal.

Source:[5]

Greasy:

Size: 8192 by 1.

Description: The recording of the word "greasy" pronounced by a woman.

Source: [5]

HochNMR:

Size: 1024 by 1

Description: NMR Signal

Source: [5]

Seismic:

Size: 1024 by 1

Description: Seismic Signal

Source:[5]

Laser:

Size: 25,000 by 1

Description: This dataset was part of the time series competition conducted by the Santa Fe Institute.

Source: [5]

RaphaelNMR:

Size: 1024 by 1

Description: NMR Signal

Source: [5]

Sunspots:

Size: 2700 by 1

Description: Monthly sunspot numbers for 225 years

Source: [5]

Transients:

Size: 512 by 1

Description: Mallat/Zhang's artificial signal built from time-frequency transients of different types.

Source: [5]

Tweet:

Size: 8192 by 1

Description: Digital signal of a bird singing

Source: [5]

Common Bird Songs (Cassette), Dover Publications, Inc., New York, 1984.

A:

Size: 4001 by 200

Description: 200 various seismic activities

Source: [16]

Functions

Linear:

Size: 32 by 1

Quadratic:

Size: 32 by 1

Sinusoidal:

Size: 481 by 1

xSinx:

Size: 63 by 1

Lorenz:

Size: 705 by 1

APPENDIX B: IMPLEMENTATION

This appendix contains the practical tools necessary to execute the prediction algorithms in Matlab. The codes are written in a modular format to facilitate accessibility and extensibility. Our goal was to create a user-friendly prediction routine using either a nearest neighbor or radial basis function approach. The algorithms presented for prediction require user input of the data set, as well as many parameters. Therefore, knowledge of the parameters and code are necessary for a successful implementation. Since all variables are global, it is not necessary to define these when calling subroutines. One needs only to call the driver which executes the menu and dialog boxes for user input.

We first display two representative prediction routines and the subroutines on which they depend. The left column represents a typical prediction using the nearest neighbors approach where the data set is from a file and is filtered using the Fourier transform. As a comparison, the right column represents the radial basis function approach to prediction using a predefined function to create the data set and the wavelet transform to filter the data. Secondly, we present the code, beginning with the driver file, **predict**, and following with all remaining code in alphabetical order. Each file contains a summary of the code and an outline of necessary input parameters. Within the summaries, the bold face commands represent subroutines, and the italicized words represent necessary parameters. The parameters may have been created by a prior subroutine, or will be input in the active subroutine.

Representative prediction routines

```
predict
  data
    openascii
    system
  linearinterp
  filters
    dimension
      traj
    fftdelay
    fourierfilter
  nearneighbor
  delayvectors
  close
  centroid
  transform
  runsvd
  nnprefunc
end
end
```

```
predict
  data
    fctn
    lorenz
    system
  linearinterp
  filters
    wvt
    dimension
  radial
  delayvectors
  enorm
  rbf
  rbfprefunc
end
end
```

predict

The driver file, **predict**, activates the local or global method for prediction of chaotic time series. The driver file depends directly on the subroutines **data**, **linearinterp**, **filters**, **nearneighbor** and **radial**. The user must input the *modeltype* as a local parameter.

```

data
linearinterp
filters
modeltype = menu('Type of Model','Nearest Neighbors','Radial Basis Functions');
if modeltype == 1
    nearneighbor
elseif modeltype == 2
    radial
end

```

centroid

The file **centroid** finds the center of mass, *cofm*, of the *neighbormatrix*, a global input parameter. The global output, *cofm* is a vector of length *m*.

```

[f,g]=size(neighbormatrix)
for i=1:f
    cofm(i)=0
    for j=1:g
        cofm(i)=cofm(i)+neighbormatrix(i,j)
    end
    cofm(i)=cofm(i)/g
end
cofm=cofm'

```

close

The file **close** finds the number of nearest neighbors, k , to the the predictor vector, $bstar$, where $bstar$ is a global input parameter. Note that only one neighbor can be chosen from each trajectory, where the trajectory breaks are chosen by the user. The *data* is plotted to aid the user in choosing these trajectory breaks. As more data is produced, additional trajectory breaks may need to be added, thus the **keyboard** function is employed. To find the k closest vectors, we first find the component-wise difference of each vector of the *delaymatrix* to form the *difmatrix*, then use the 2-norm as a measurement function. This is the *normatrix*. We then sort the *normatrix*, finding the *neighbormatrix*, the matrix of nearest neighbors, and the global output of this subroutine.

```

if t==1
    plot(data)
    space=menu('How are the trajectories spaced?','Evenly','Unevenly');
    if space==1
        spacing=input('What is the spacing?');
        if rem((length(data)-1),spacing) == 0
            j=0;
            for i=1:spacing:length(data)
                j=j+1;
                trajbrk(j)=i;
            end
            spacing(j+1)=length(data);
        else
            j=0;
            for i=1:spacing:length(data)+1
                j=j+1;
                trajbrk(j)=i;
            end
        end
    end
end
else

```

```

        trajbrk=input('Enter all trajectory breaks in brackets ');
    end
else
    if space==1
        if rem((length(data)-1),spacing) == 0
            j=0;
            for i=1:spacing:length(data)
                j=j+1;
                trajbrk(j)=i;
            end
            trajbrk(j+1)=length(data);
        else
            j=0;
            for i=1:spacing:length(data)+1
                j=j+1;
                trajbrk(j)=i;
            end
        end
    end
    else
        disp('Please add or change any trajectory breaks. ');
        disp('When complete, type return. ');
        disp('The last two trajectory breaks are ');
        disp(trajbrk(length(trajbrk)-1:length(trajbrk)))
        keyboard
    end
end
end
[f,g]=size(delaymatrix);
for i=1:g
    difmatrix(:,i)=delaymatrix(:,i)-delaymatrix(:,qv) ;
    normatrix(i)=norm(difmatrix(:,i),2) ;
end
[y,s]=sort(normatrix);
for i=1:g
    if s(i) == qv
        for j=i+1:g
            s(j-1)=s(j) ;
        end
    end
end
end
for kk=1:length(s)
    for j=1:length(trajbrk)-1

```

```

        if (s(kk) < trajbrk(j+1) & s(kk) >= trajbrk(j));
            traj(kk)=trajbrk(j) ;
        end
    end
end
i=2;
neighbors(:,1)=delaymatrix(:,s(1));
for j=2:length(s)
    if traj(j) = traj(j-1:-1:1)
        neighbors(:,i)=delaymatrix(:,s(j));
        XX(i)=data(s(j)+w);
        i=i+1;
    end
end
neighbormatrix(:,1:k)=neighbors(:,1:k) ;
bstar=delaymatrix(:,qv);
X(:,1:k)=XX(:,1:k);
X(:,1)=data(s(1)+w);

```

data

The file **data** asks the user to choose the type of input to be used in the prediction algorithm; a function or timeseries and the relative length of the timeseries. Dependent algorithms include **fctn**, **openascii**, and **system**. The global output consists of the vectors x and z , where x is the spacing and z is the vector of data.

```

m=menu('Choose your type of input','function','timeseries');

if m==1
    fctn
    vv=input('Do you want to use the whole vector for prediction?');
    if vv == 1
        z=b ;
    elseif vv == 2
        vvv=input('With what vector entry do you want to start the training set? ');
        vvv=input('What is the last entry for the training set?') ;
        z=b(vvv:v);
        x=x(1:length(z));
    end
    system
    end
elseif m==2
    u=menu('How will you enter data','directly into matlab','from a predefined
    ascii file');

    if u == 1
        z=input('Please enter the data within square brackets. ');
        x=1:length(z) ;
        x=x' ;
        z=z' ;
    elseif u == 2
        openascii ;
        uu=siz(2) ;
        if uu ≠ 1
            v=input('Which vector will you be using?') ;
            z=b(:,v) ;
            x=1:length(z) ;
        end
    end
end

```

```
    x=x' ;
elseif uu ==1
    v = 1 ;
    z=b(:,v) ;
    x=1:length(z) ;
    x=x';
end
vv=input('Do you want to use the whole vector for prediction?') ;
if vv == 2
    vvvv=input('What is the starting point? ');
    vvv=input('What is the length of the vector for the training set? ');
    z=b(vvvv:vvvv+vvv-1,v);
    x=1:length(z) ;
    x=x' ;
end
end
end
system
end
```

delayvectors

The subroutine **delayvectors** creates the *delaymatrix*; the matrix of delays, of length w ; from the given time series. This routine depends on the global parameters, w and *data*.

```

if t == 1
  for i=1:length(data)-(w-1)
    k=i
    for j=1:w
      delaymatrix(w-j+1,i)= data(k)
      k=k+1
    end
  end
end
if t > 1
  for i=length(data)-(w-1):length(data)-(w-1)
    k=i
    for j=w:-1:1
      delaymatrix(j,i)= data(k)
    end
  end
end
end

```

dimension

The subroutine **dimension** relies on user input to the method of calculating the dimension of the attractor. If the dimension is known, the user can input it directly. Otherwise, the user can opt for a statistical calculation of the dimension, inputting a maximum value. The algorithm **dimension**, requires **traj** and **svd** as further sub-routines. The global output, *dim*, is the calculated dimension of the attractor.

```

u=menu('How would you like to calculate attractor dimension? ', 'I will enter it', 'Calculate
using statistical Methods');

if u == 1
    dim=input('What is the dimension');
elseif u == 2
    maxdim=input('Choose a number almost sure to be larger than the dimension')
    traj
    sv =svd(tra) ;
    dim = 1 ;
    for i=1:length(sv)-1
        if sv(i+1)/sv(1) > .01
            dim = dim+1 ;
        end
    end
end
end
end

```

enorm

The file **enorm** computes the Euclidean norm of the difference of two vectors from the input parameter, *delaymatrix*. The global output is given by the symmetric matrix, *A*.

```

if t ==1
    for i=1:length(delaymatrix)
        A(i,i)=0.0;
        for j=i+1:length(delaymatrix)
            A(i,j)=norm(delaymatrix(:,i)-delaymatrix(:,j));
            A(j,i)=A(i,j) ;
        end
    end
end
if t ≠ 1
    for i=1:length(delaymatrix)
        for j=length(delaymatrix):length(delaymatrix)
            A(i,j)=norm(delaymatrix(:,i)-delaymatrix(:,j)) ;
            A(j,i)=A(i,j) ;
        end
    end
end
end

```

fctn

The file **fctn** contains five predescribed functions which, once sampled, creates an equispaced time series. The user chooses the type of function to be used from a menu. The function creating the chaotic lorenz time series relies on the subroutines **lorenz** and **interp1**, as well as the built in function **ode45**. If the user desires to use an original function, the menu contains an appropriate option. The global output consists of x and b , where x is the spacing of the data, b .

```

u=menu('What type of function do you want to use',
'My own', 'Linear','Quadratic','Sinusoidal','xsinx','lorenz data') ;
if u == 1
    x=input('How does your independent variable vary?');
    b=input('What is the function?');
elseif u == 2
    x=(.5:.5:16)';
    b=2*x;
elseif u == 3
    x=(.5:.5:16)';
    for i=1:length(x)
        b(i)=x(i)*x(i) ;
    end
    b=b' ;
elseif u == 4
    x=(0:pi/80:6*pi)';
    b=sin(x);
elseif u == 5
    x=(0:.4:25)';
    for i=1:length(x)
        b(i)=x(i)*sin(x(i));
    end
    b=b';
elseif u == 6
    p0=0;
    pf=50;
    x0=[1 1 .9]';

```

```
[p,x]=ode45('lorenz',p0,pf,x0);  
x(1:1700)=x(301:2000);  
t=0:.05:35.2;  
b=interp1(p,x(:,1),t);  
x=t';  
end
```

ftdelay

The routine **ftdelay** finds the delay matrix to be used in Fourier filtering. The length of the delay matrix, *win* is the length of the window considered to contain all necessary information to determine the state space. The input necessary consists of the *data*. The global output is the *delaymatrix* which will be used in **fourierfiltering**.

```
win=input('What is the length of the window? ');
for i=1:length(data)-(win-1)
    k=i
    for j=1:win
        delaymatrix(win-j+1,i)=data(k);
        k=k+1
    end
end
end
```

filters

The routine **filters** asks the user for the type of filtering to be used: Fourier, Wavelet or neither then implements the respective algorithm. The routine **filters** depends on the subroutines: **dimension**, **fftdelay**, **fft1**, **wvt** and **delayvectors**. The global output, *filtered* is the filtered delay matrix.

```

u=menu('How do you want to filter your data?'),
'Fourier Transform','Wavelet Transform','I do not want to filter the data');

if u==1
    dimension
    fftdelay
    fft1
elseif u == 2
    wvt
    dimension
    t=1;
    delayvectors
elseif u == 3 ;
    dimension
    t=1;
    delayvectors
    filtered = delaymatrix ;
end

```

fourierfilter

The algorithm **fourierfilter** performs the filtering of *delaymatrix* using the **fft** and **ifft** built in **Matlab** commands. The global input parameters are *delaymatrix* and *win*, where $win \geq$ the embedding dimension, m . The global output, *filtered*, is the filtered *delaymatrix* and *dim* is the embedding dimension.

```
m=input('What is the length of the embedding dimension? ');  
  
prefiltered=fft(delaymatrix,win);  
[f,g]=size(delaymatrix);  
for i=m/2+1:win  
    for j=1:g  
        fdelaymatrix(i,j)=0;  
    end  
end  
filtered=real(ifft(prefiltered,m));  
dim=m;
```

linearinterp

The routine **linearinterp** uses the input parameter, s , of how many interpolation steps per sample period, and linear interpolation to increase the number of data points in the time series. For vector representation of the data, $data(k)$, for example, Matlab is incapable of representing fractional k . Therefore, the original data set of length n will have new length sn . The routine **linearinterp** relies on the global parameters: $data$ and a . The global output, $data$ now represents the interpolated $data$, and a it's new length.

```

interp=input('How many interpolation steps per sample period? Of power 2 ');
if interp ≠ 0
delta=(a(2)-a(1))/(interp+1);
for i=1:interp*(length(data)-1) +length(data)
    aa(i)=a(1)+(i-1)*delta;
end
i=0; j=1;
for l=1:interp+1:length(aa)-interp;
    i=i+1;
    j=j+1;
    for k= 0:gg
        g(k+1)=data(i)*(aa(k+1)-a(j))/(a(i)-a(j)) + data(j)*(aa(k+1)-a(i))/(a(j)-a(i))
    end
end
g(length(aa))=data(i+1);
a=aa';
data=g';
end

```

Lorenz

The file **Lorenz** defines a system of differential equations to be used within the Matlab routine **ode45**.

```
function xdot = Lorenz(p,x)

xdot = zeros(3,1);
xdot(1) = -10.*x(1)+ 10.*x(2);
xdot(2) = 28.*x(1) -x(2) - x(1).*x(3);
xdot(3) = -(8/3).*x(3) + x(1).*x(2);
```

nearneighbor

The routine **nearneighbor** is the prediction algorithm for the nearest neighbor approach to prediction. The global parameters used include the *data*, *w*, and *pv*. The subroutines **nearneighbor** is dependent on include **delayvectors**, **close**, **centroid**, **transform**, **runsvd**, and **nnprefunc**.

```
k=input('How many neighbors would you like to use to predict?');
l=input('What subspace, less than the order of the filtered matrix,is spanned by the
m-dimensional delay vectors?') ;
h=input('How far would you like to predict?');
pv= length(data)-w+1;
for t=1:h
    qv=pv+t-1;
    delayvectors
    close
    centroid
    transform
    runsvd
    nnprefunc
end
```

nnprefunc

nnprefunc is the nearest neighbors prediction function. This subroutine first calculates the projection of the data points $b_1 - c, \dots, b_k - c$ onto \mathbb{R}^l . The matrix *project* consists of the vectors of projections. Secondly, we use the Matlab function *polyfit* to find the coefficients of a linear equation that fits the data $p(x_i) = y_i$, in a least squares sense. Finally, we estimate $P_t(b^*)$ by evaluating $L(x) = ax + d$. **nnprefunc** is dependent on the global parameters *transformatrix*, *lspace*, *bstar*, and *X*. The global output is the predicted value of the next data point, *L*.

```
[f,g]=size(transformatrix) ;
for o=1:f
    project(o)=transformatrix(o,1:g)*lspace ;
end
projectb=bstar'*lspace;
p=polyfit(project',X',1);
L=p(1)*projectb + p(2) ;
data(length(data)+1)=L
```

openascii

openascii is a subroutine to open an ascii format file which contains data in vector form. The file can contain more than one vector of data, but a maximum of one vector can be used for prediction. User input includes the file name and *siz*, the matrix size of the file to be opened. Built in Matlab functions **fopen** and **fscanf** are utilized. Global output, *b* is the data string, with sampling size assumed to be 1.

```
fnm=input('What is the file name? Enter within apostrophe ');  
fid=fopen(fnm) ;  
siz=input('What is the size of the file? Enter within brackets. ');  
[a,count]=fscanf(fid,'%f',[siz(2),siz(1)]) ;  
b=a' ;
```

radial

radial contains the subroutines to run the radial basis function approach to prediction. This routine depends on the following subroutines; **delayvectors**, **enorm**, **rbf**, and **rbfprefunc**. User input includes the type of radial basis function to be used in the prediction, as well as the length of the prediction.

```
n=menu('What type of rbf do you want to use?'),
'linear','cubic','thin plate spline','gaussian','inverse multiquadric','multiquadric')
if n==5 | 6
    c=input('Choose a positive value for c. ');
end
h=input('How far would you like to predict?');
for t=1:h
    delayvectors
    enorm
    rbf
    rbfprefunc
end
```

rbf

The subroutine **rbf** calculates the matrix R whose elements are the radial basis functions of the Euclidean norm of $(x_i - x_j)$, $i, j = 1, \dots, N$. The global parameters which **rbf** depends include *delaymatrix*, A , c , and the *type of rbf*. The global output is the matrix R .

```

for i=1:length(delaymatrix)
  for j=1:length(delaymatrix)
    if n == 1
      R(i,j) = A(i,j) ;
    elseif n == 2
      R(i,j) = A(i,j)^3 ;
    elseif n == 3
      R(i,j) = A(i,j)^2*log(A(i,j)) ;
    elseif n == 4
      R(i,j) = exp(-(A(i,j)^2)) ;
    end
  end
end
if n == 5
  for i=1:length(delaymatrix)
    for j=1:length(delaymatrix)
      R(i,j) = (A(i,j)^2 + c^2)^-0.5 ;
    end
  end
end
if n == 6
  for i=1:length(delaymatrix)
    for j=1:length(delaymatrix)
      R(i,j) = (A(i,j)^2 + c^2)^.5 ;
    end
  end
end
for i=1:length(delaymatrix)-1
  d(i)=data(i+w) ;
end
d=d';

```

```

[f,g]=size(R);
count=0;
for i=1:(g-1-count)
    for j=g:-1:(i+1)
        if R(:,i)==R(:,j)
            count=count+1;
            for k=j:g-1
                if j =g
                    R(:,k)=R(:,k+1);
                    d(k)=d(k+1);
                end
            end
        end
    end
end
end
extra=0;
for i=1:g-1
    if R(:,i)==R(:,i+1)
        extra=extra+1;
    end
end
for i=1:g-extra
    RR(:,i)=R(:,i);
    dd(:,i)=d(i);
end
extras=0;
[ff,gg]=size(RR);
for i=1:gg-2
    for j=i+1:gg
        if RR(:,i)==RR(:,j)
            extras=1;
        end
    end
end
end
clear RR
clear dd
for k=1:g-extra-extras
    RR(:,k)=R(:,k);
    dd(:,k)=d(k) ;
end
R=R';

```

rbfprefunc

rbfprefunc first solves the linear system $Ra=d$, then creates the function $s(x)$ which is a linear combination of the radial basis functions. The routine **rbfprefunc** depends on the type of rbf, as well as *delaymatrix*, *data*, *n*, *w*, and *R*. The global output is the prediction of the continued time series, *L*.

```

for i=1:length(delaymatrix)
    d(i)=data(i+w-1) ;
end
d=d' ;
a=R\d ;
for i=1:w
    y(i)=(data(length(data)-i+1));
end
y=y';
L=0 ;
r=0 ;
if n == 1
    for i=1:length(R)
        r=norm(y-delaymatrix(:,i)) ;
        phi(i)=r ;
        L=L+a(i)*phi(i) ;
    end
end
if n == 2
    for i=1:length(R)
        r=norm(y-delaymatrix(:,i)) ;
        phi(i)=r3 ;
        L=L+a(i)*phi(i) ;
    end
end
if n == 3
    for i=1:length(R)
        r=norm(y-delaymatrix(:,i)) ;
        phi(i)=r2*log(r) ;
        L=L+a(i)*phi(i) ;
    end
end

```

```
        end
    end
    if n == 4
        for i=1:length(R)
            r=norm(y-delaymatrix(:,i)) ;
            phi(i)=exp(-(r2)) ;
            L=L+a(i)*phi(i) ;
        end
    end
    if n == 5
        for i=1:length(R)
            r=norm(y-delaymatrix(:,i)) ;
            phi(i)=(r2+c2)-5 ;
            L=L+a(i)*phi(i) ;
        end
    end
    if n == 6
        for i=1:length(R)
            r=norm(y-delaymatrix(:,i)) ;
            phi(i)=(r2+c2).5 ;
            L=L+a(i)*phi(i) ;
        end
    end
    data(length(data)+1)=L
```

runsvd

The subroutine **runsvd** depends on the Matlab function **svd** and the global variables *transformatrix* and *bstar*. The local output *lspace* is the subspace spanned by the delay vectors. We then project the rows of *transformatrix* onto *lspace*, yielding *project*. The linear model, *p*, is then formed through these points. The global output, *L*, is the linear model evaluated by the projection of *bstar* on *lspace*.

```
[U,S,V]=svd(transformatrix);
[f,g]=size(V) ;
for j=1:l
    for i=1:f
        lspace(i,j)=V(i,j) ;
    end
end
[f,g]=size(transformatrix) ;
for o=1:f
    project(o)=transformatrix(o,1:g)*lspace ;
end
projectb= bstar'*lspace;
p=polyfit(project',X',1);
L=p(1)*projectb + p(2) ;
```

system

The subroutine **system** forms the data into a working unit, where the global output is the *data* and its spacing is given by *a*. The routine **system** depends on the global parameters *x* and *z*, the spacing and data of the timeseries.

```
Timeseries=[x,z]
[f,g] = size(Timeseries)
data=Timeseries(:,2)
a=Timeseries(1:f,1)
```

traj

The subroutine **traj** finds the trajectory matrix from which we can statistically calculate the dimension of the attractor. This routine uses the global parameters *maxdim* and *data*. The global output, *tra* is the trajectory matrix.

```

for i=1:maxdim
  for j=1:length(data)-(maxdim-1)
    tra(j,i)=data(i+j-1);
  end
end
end

```

transform

The subroutine **transform** depends on the global parameters *neighbormatrix*, *cofm* and *bstar*. This routine creates the matrix *transformatrix* whose rows consist of the vectors $b_1-cofm, \dots, b_k-cofm$, where *cofm* is the center of mass of the *neighbormatrix*. Also, the routine recalculates *bstar* by subtracting off the center of mass of the *neighbormatrix*.

```

[f,g]=size(neighbormatrix);
for i=1:f
  for j=1:g
    transformatrix(j,i)=neighbormatrix(i,j)-cofm(i);
  end
end
for i=1:f
  bstar(i)= bstar(i)-cofm(i);
end
end

```

wvt [5]

The subroutine **wvt** performs the wavelet transform and inverse transform to create a filtered data set. The user must input what type of filter which will be used, the integer parameter of the wavelet, and the type of soft thresholding to be used. This routine is dependent on built in functions **MakeONFilter**, **NormNoise**, and **WaveShrink**. The only global parameters necessary is the *data*. The global output is *fdata*, the filtered data string.

```

fil=menu('What type of filter do you want to use?',
'Haar Filter','Beylkin','Coiflet','Daubechies','Symmlet','Vaidyanathan');
par=input('What integer parameter of the specific wavelet type do you want?');
st=menu('What type of soft thresholding do you want to use?',
'Visu','SURE','Hybrid','MinMix','MAD');
if fil==1
    QMF8=MakeONFilter('Haar',par);
elseif fil==2
    QMF8=MakeONFilter('Beylkin',par);
elseif fil==3
    QMF8=MakeONFilter('Coiflet',par);
elseif fil==4
    QMF8=MakeONFilter('Daubechies',par);
elseif fil==5
    QMF8=MakeONFilter('Symmlet',par);
elseif fil==6
    QMF8=MakeONFilter('Vaidyanathan',par);
end
scaled=NormNoise(data,QMF8);
y= scaled *1.2 ;
if st==1
    [fdata,wcoef]=WaveShrink(y,'Visu',5,QMF8);
elseif st == 2
    [fdata,wcoef]=WaveShrink(y,'SURE',5,QMF8);
elseif st ==3
    [fdata,wcoef]=WaveShrink(y,'Hybrid',5,QMF8);
elseif st ==4

```

```
[fdata,wcoef]=WaveShrink(y,'MinMax',5,QMF8);  
elseif st==5  
    [fdata,wcoef]=WaveShrink(y,'MAD',5,QMF8);  
end  
datalen=1:length(data)
```