# MODELING CHAOTIC SYSTEMS

By

Khaled Marzoug Al-Mughadhawi

Bachelor of Science in Electrical Engineering
King Fahad University of Petroleum and Minerals
Dhahran, Saudi Arabia
1989

Master of Science in Electrical Engineering
Southern Methodist University
Dallas, Texas
1999

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
In Electrical Engineering
July, 2005

MODELING CHAOTIC SYSTEMS

Thesis Approved:

Dr. Martin T. Hagan
Thesis Adviser

Dr. Gary E. Young

Dr. Rao. K. Yarlagadda

Dr. Carl D. Latino

Dr. A. Gordon Emslie
Dean of the Graduate College

## ACKNOWLEDGMENTS

First and foremost, I would like to thank Allah (the God) for helping me and making it possible for me to reach this point of knowledge. Peace and blessing of Allah are on his messenger Muhammad, his last prophet and the seal of all messengers.

My thanks extend to my parents: Marzoug, and Mohelah for their support, patience, and extreme love. My father grew up as an orphan in a small town known as Abar Ali (south of Madina, Saudi Arabia). When I was born, in 1966, he had to stop his education in the Madina teaching institute. He did that in order to support his new born child. He use to say to me “Khaled, I want you to compensate my loss of education by bringing me the highest degree in your major”. “I hope I did that Dad.”

I would like to thank my advisor Dr. Martin Hagan for his support, patience, and help. He use to arrange weakly seminars for me, often lasting for three hours. He listened very carefully to my presentations and corrected me, suggesting new ways to solve the research problems. My thanks extend to all his graduate students who attended these seminars. I would like also to thank Dr. Yong Hu, and Dr. Tagi Menhaj for their valuable suggestions and comments. My thanks extend also to the dissertation committee members for their comments and suggestions.

Three professors in the Mathematics department gave me a much needed help and support. Dr. Saleh Mehdi gave me three lectures in differential geometry which helped me in understanding the embedding theorem. He and I corresponded often regarding the mathematical challenges I faced. Dr. Roger Zierau was very helpful to me. We spent long hours for many weeks discussing the linear Oseledec theorem. His support was very valuable to me. Dr. Leticia Barchini's suggestions and comments made the proof of the theorem possible as well.

My friends in the Islamic society of Stillwater made the difficult task of this degree much easier by their friendly relationship with me. I ask Allah to help them and make their future project of building a new mosque a success. My friends in the Saudi house showed me the true meaning of brotherhood. Our weakly meeting proved to be very important in bring us home far away from home.

I would like also to thank my wife Nawal for her extreme support and patience with me and our five kids. Without her help, my goal could have been much more difficult to accomplish. My kids Abdu-Rahman, Maha, Ahmad, Roba, and Arwa gave me the social stability which was also necessary to achieve this task. My good neighbor Sonya Brewster helped to proofread many chapters of my dissertation.

Finally I would like to thank the Saudi government for their help and financial support of my research.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

## NOMENCLATURE

Basic Concepts

Scalars: small *italic* letters...*a, b,* $\phi$

Vectors: small bold non-italic letters, or under-bar symbol **a**, **b**, $\underline{\phi}$

Matrices: capital Bold non-italic letters ... **A**, **B**, **C**

Language

Vector means a column of numbers.

Vector element

Small italic with an index $a(i)$ or $a_i$

Matrix elements

An element of matrix **A**: small italic showing the row then the column of the matrix:

$a(i, j)$, $a_{ij}$, or $\mathbf{A}_{(i, j)}$

The $k^{th}$ column vector of a matrix **I**

$\mathbf{i}^k$

The $k^{th}$ row vector of a matrix **I**

${}^k\mathbf{i}$

Norm

$\|\mathbf{a}\|$

Covariance matrix

$\mathbf{C}$

Eigen value and eigen vector

$\lambda$ and $\underline{\eta}$ (or bold small non-italic letter)

Set

Empty set

$\varnothing$

Subset of $\Re^n$

A, B, C

Dimension

Space dimension is superscript

$\Re^d$

Box counting dimension

$d_c$

Theoretical minimum embedding dimension

$d_E$

Actual minimum embedding dimension

$d_L$

Side length of a *d*-cube (box-counting dimension)

$\sigma$

Interval in $\Re^1$ (box-counting dimension)

$N$

Number of $\sigma$ $d$-cubes needed to cover a set (box-counting dimension)

$c_\sigma^d$

Lyapunov dimension (Chapter 8)

$d_{Lyp}$

The number of Lyapunov exponents such that $\sum_{j = 1, 2, \ldots, k_L} \lambda_j > 0$

$k_L$

Power

For a vector

$(\mathbf{x})^k$

For a matrix

$(\mathbf{A})^k$

Dynamics

State vector in the original space

$\mathbf{x}(n)$

Equilibrium point

$\mathbf{x}^*$

The map for the state update in the original space

$$\mathbf{x}(k+1) = \boldsymbol{f}(\mathbf{x}(k))$$

Scalar Measurements

$$y(m)$$

Measurement function

$$y(m) = h(\mathbf{x}(m))$$

Delay-time

$$T$$

Delay-coordinate map (for embedding)

$$\mathbf{y}_d(m) = F_d(h, \mathbf{x}(n))$$

Delay-vector in the reconstructed attractor in $\Re^d$

$$\mathbf{y}_d(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d-1)T)\right]^t$$

The map for the state update in the reconstructed space

$$\mathbf{y}_d(m+1) = \underline{\phi}(\mathbf{y}_d(m))$$

Map for the output update in the reconstructed space

$$y(m+1) = \mu(\mathbf{y}_d(m))$$

The sampled measurement (predictive algorithm)

$$y_s(m) = y(1+(m-1)T)\text{, Equation (6.16)}$$

The sampled measurement with zero mean

$$s(m) = y_s(m) - \frac{1}{L}\sum_{k=1}^{L} y_s(k)$$

The delay-vector created from $s(m)$

$$\mathbf{y}_d^s(m) = \left[s(m)\ s(m-1)\ \ldots\ s(m-d+1)\right]^t$$

Vectors distances matrix in the *d*-dimensional space

$\mathbf{Q}_d$, see section 5.2.1 on page 39

Number of neighbors in $\Re^{d+1}$ used to search for the nearest neighbor of $\mathbf{y}_d(m)$ (for the CND method)

$w$

Indices matrix of the *w*-nearest neighbors in the *d*-dimensional space

$\mathbf{I}_d$

Nearest neighbor distances vector in *d*-dimensional space

$\mathbf{r}_d$

Nearest neighbor distances in *d*-dimensional space with distances computed in (*d* + 1)-dimensional space

$\mathbf{e}_{d+1}$

Number of neighbors of $\mathbf{y}_d(m)$

$N_b$

Basis matrix for the PCA projection method

$\mathbf{B}_d = \left[\underline{\eta}_1\ \underline{\eta}_2\ \ldots\ \underline{\eta}_d\right]$; $\underline{\eta}_i$ are the eigen vectors of the covariance matrix **C**

The $N_b$ neighbors of the delay-vector $\mathbf{y}_d(m)$ in $\Re^d$

$\mathbf{Y}_d^{N_b}$: when the delay-coordinate projection method is used.

Its element is $\mathbf{y}_d(m)$

$\mathbf{Y}_d^{N_b, p}$: when the PCA projection method is used. It element

is $\mathbf{y}_d^p(m)$ (see page 62)

Index of the $k^{th}$ neighbor of $\mathbf{y}_d(m)$

$i_d^k(m)$

Average vector in the space $\Re^{d_E}$

$\bar{\mathbf{y}}_{d_E}(m)$, the CDT pseudocode

Average mutual information between two measurements

I

Threshold for significant FNN change (for the CND method)

$\alpha$

Threshold of distance increase (for the CDD method)

$\rho$

Attractor mean value (for the CDT method)

$\beta$

A fraction of attractor mean (for the CDT method)

$\zeta$

Time step for the CDT method

$\Delta$

Distance between the nearest neighbors after $\Delta$ time steps (CDT)

$\sigma_{\Delta}$

Threshold for significant change in prediction error (Predictive technique)

$\gamma$

Signal variance

$(\sigma_s)^2$

Noise variance

$(\sigma_n)^2$

Jacobian matrix at $\mathbf{y}_d(m)$

$\mathbf{J}_{\mathbf{y}_d(m)}$ or $\mathbf{J}$

Sampling time

$\tau_s$

Delay time in Mackey-Glass equation (Chapter 7)

$t_f$

Lyapunov Exponent (global)

$\lambda$

Finite time Lyapunov Exponent (local)

$\lambda^t$

Infinitesimal distance

$\varepsilon$

Infinitesimal perturbation from $\mathbf{x}(t)$ to $\mathbf{x}_\varepsilon(t)$

$$\underline{\delta}(t) = \mathbf{x}_\varepsilon(t) - \mathbf{x}(t)$$

Abbreviations

FNN: False Nearest Neighbor

CND: Change of Neighbor with Dimension

CDD: Change of Distance with Dimension

CDT: Change of Distance with Time

PCA: Principal Component Analysis

SNR: Signal to Noise Ratio

ECG: Electrocardiogram

TDL: Tapped Delay Line

SSE: Sum Squared Errors

LE: Lyapunov Exponent

SVD: Singular Value Decomposition

Linear Algebra

Singular value of a matrix

$\sigma_i$ or $\sigma_i(\mathbf{A})$

Multiplications of the *n* Jacobian matrices

$$\mathbf{J}_\mathbf{y}^n = \mathbf{J}_{\mathbf{y}(n-1)}\mathbf{J}_{\mathbf{y}(n-2)}\cdots\mathbf{J}_{\mathbf{y}(0)}$$

Perturbation vector from time $m$ to time $n$

$$\underline{\delta}(m) = \mathbf{x}(n) - \mathbf{x}(m)$$

Matrix of the $N_b$ perturbation vectors from $\mathbf{y}_d(m)$

$$\mathbf{E}_m$$

Diagonal matrix

$$\mathbf{D}$$

Diagonal matrix with elements magnitude of $x$

$$\mathbf{D}_a \text{; } |x| \in \mathbf{D}_a$$

Linear Oseledec matrix

$$\mathbf{O}_k$$

General Oseledec matrix

$$\Lambda_{\mathbf{x}}$$

Multilinear Algebra

Vector exterior product

$$\mathbf{e}_i \wedge \mathbf{e}_j$$

The $q^{th}$-exterior power of a linear operator $L$

$$L^{\wedge q}$$

Adjoint of a linear operator $L$

$$L^*$$

Matrix representation of the wedge of the linear operator $L$ ($L^{\wedge q}$)

$$\mathbf{B}_q$$

# CHAPTER 1

# OBJECTIVE AND CONTRIBUTIONS

## 1.1 Introduction

In this chapter, we will show the objective of this research and give a brief list of our contributions. In Section 1.4, we give a literature review of some important publications in chaotic modeling. A summary of the remaining chapters is given in Section 1.5.

## 1.2 Objective

The objective of this research is to model chaotic systems. We will build a chaotic model from a set of scalar measurements taken from the system. Evolution of the states in the model follows the evolution of the hidden states in the original system. To build a chaotic model, we start by estimating the model order. The next step is to estimate the set of Lyapunov exponents (the model parameters) that characterize the chaotic system. A good chaotic model depends on both the accuracy of the estimated model order and the estimate of Lyapunov exponent values.

## 1.3 Contributions

Our contributions are:

(1) Four new algorithms for estimating the model order.

a) The Change of Neighbors with Dimension (CND).

b) The Change of Distance with Dimension using the Global neighbor search ($\mathrm{CDD}_G$).

c) The Change of Distance with Time using the Global neighbor search ($\mathrm{CDT}_G$).

d) The predictive.

(2) The proof of a new theorem that relates the poles of a linear system to the set of Lyapunov exponents.

(3) Implementation of the new theorem result to estimate the set of Lyapunov exponents for chaotic systems.

(4) Development of an existing algorithm which uses a neural network to estimate the set of Lyapunov exponents.

In total, we implemented four algorithms to estimate the model order and two algorithms to estimate the set of Lyapunov exponents for a chaotic system. The six algorithms were tested on different chaotic systems. The testing examples include noise free and noisy signals. In addition, the testing systems have different dimensions.

## 1.4 Literature review

There are two primary areas of research in modeling chaotic systems. These areas, which will be discussed briefly, are: estimating the model order and estimating the set of Lyapunov exponents. In this section, we list some key references in these areas. We begin by listing literature dealing with estimating the model order. After that we list the literature that describes estimating the set of Lyapunov exponents.

We begin by reviewing publications that discuss estimating the model order. We found many papers by Abarbanel and his colleagues that talk about different methods for estimating the model order. Kennel, Brown, and Abarbanel [KBA92] introduced estimating the model order by the method of False Nearest Neighbors (FNNs). This method is a geometric one. It depends on measuring the distance between neighbors as the model order increases. The distance between false neighbors increases more than the distance between true neighbors as the model order increases. In 1993, Abarbanel and Kennel [AbKe93] developed the method of FNNs further and introduced a new method that uses time as well as distance to check for the existence of FNNs. In this proposal, we will present new algorithms that improve on the results of Abarbanel's algorithms. In 1997, Cao [Cao97] suggested a new method for estimating the model order. His method is a geometric method and it is closely related to Abarbanel method [KBA92]. Instead of using a threshold value to determine false neighbors, he suggested the use of the mean of distance change as the model order increases. Abarbanel [Aba98] used a geometric method with a predictor to estimate the model order. Since the points inside the attractor have an evolution rule, their neighboring points will evolve according to a certain rule as well. To determine the model order, he used a predictor to estimate the evolution rule. When the model order is not large enough, prediction errors will increase. As the model order increases, the prediction errors drop. At certain point, further increase of the model order does not improve the prediction errors. At this point, the model order is found. In 2002, Kennel and Abarbanel [KeAb02] used the method of FNNs with a global neighbor search method. The projection method used in this paper is the delay-coordinate method. (Our second algorithm ($\mathrm{CDD}_G$) used the method of

FNNs, with a global neighbor search, but the $\mathrm{CDD}_G$ used the Principal Component Analysis projection method.)

Now we review some papers that talk about estimating the set of Lyapunov exponents for a chaotic model. The most important paper in this field is the one by Oseledec [Ose68]. He proved that the set of Lyapunov exponents can be estimated from the product of an infinite number of Jacobian matrices along the attractor of the system. (In this proposal, we will present a new theorem that provides insight into the Oseledec theorem by relating Lyapunov exponents to the poles of a linear system.) In 1985, Eckmann and Ruelle [EcRu85] and Sano and Sawada [SnSd85] applied the Oseledec theorem to measurements taken from a chaotic system. Both papers use orthogonalization methods to overcome the numerical problem of multiplying a large number of matrices. Parlitz [Par92] introduced the identification of spurious Lyapunov exponents. He did that by reestimating the set of Lyapunov exponents from measurements backward in time. By doing that, he found that true exponents change their signs, while spurious ones change their values as well as signs. Darbyshire and Broomhead [DaBr96] estimated the set of Lyapunov exponents by the methods of least squares and total least squares. Their method uses the pseudo-inverse to estimate the Jacobian matrices. After that, they applied the orthogonalization method proposed by Eckmann. Djamai and Coirault [DjCo02] introduced the use of neural networks to estimate the set of Jacobian matrices. After estimating the Jacobian matrices, they used the same method proposed by Eckmann to reorthogonalize the product of these matrices.

## 1.5 Remaining chapters summary

In Chapter 2, we will present an introduction to chaotic dynamical systems. In Chapter 3, we introduce modeling chaotic systems and discuss modeling by embedding. Examples are given to illustrate the idea of embedding functions.

In Chapter 4, we will present an introduction to four methods used to find the minimum dimension (model order) required for embedding. In Chapter 5, we implement the methods found in Chapter 4 to find the minimum embedding dimension of the Henon map. Six algorithms based on the four previous methods are presented in Chapter 6. (Four of the six algorithms represent our original work.) Full details of the algorithms are given, and more practical issues are discussed in Chapter 6. The results of implementing the six algorithms on nine chaotic systems are found in Chapter 7. The minimum embedding dimensions of the nine systems are listed in this chapter, and a comparison is made among the six algorithms.

In Chapter 8, we will explore the theory of Lyapunov exponents (LEs) and prove a new theorem that relates the poles of a linear system to the set of LEs. In Chapter 9, we will discuss the estimation of the LE values and apply the result of the new theorem to estimate the LEs. In this chapter, we will also improve an existing algorithm that uses a neural network to estimate the LEs. We tested the two algorithms by estimating the LEs of different chaotic systems. We also compare the two algorithms to an existing algorithm using the test systems.

In Chapter 10, we will give a conclusion of the dissertation and give some recommendations on possible future research.

# CHAPTER 2

# INTRODUCTION TO CHAOTIC SYSTEMS

## 2.1 History of dynamical systems

Isaac Newton (1642-1727) introduced the idea of modeling the motion of objects by equations. Position, velocity, and acceleration were the fundamental parameters of his equations. From position, velocity, and acceleration, he could describe the state of a moving object at any given time. Later scientists modelled dynamical systems by using a set of differential equations. The solution of these equations describes the state of the system at any given time.

If the solution of the set of differential equations remains in a bounded region, the sequence of states reduces to either i) a steady state, generally because of loss of energy by friction, or ii) a periodic or quasi periodic motion. An example of case (ii) is the motion of the moon around the earth and the motion of the earth around the sun. Case (i) and case (ii) remained the only two known bounded solutions until the use of modern computers made possible the numerical solution of sets of differential equations.

In 1963 Edward Lorenz published a paper entitled “Deterministic non Periodic Flow” [Lo63]. He discovered a new bounded attractor that is not periodic but was filling a region in space. This was the first time the world knew about the third possible bounded attractor: iii) the chaotic attractor. The new solution (chaotic) is not simply periodic nor

quasi periodic with a large number of periods. Chaotic motion is possible with simple one dimensional systems. Although chaotic motion can be produced from simple systems, it is very complicated and becomes unpredictable after a short time. This happens because of the sensitivity of chaotic systems to changes in the initial conditions [ASY96].

In the next section, we define some terms related to dynamical systems. Three characteristics of chaotic systems are presented in Section 2.3. In Section 2.4, we present some examples of chaotic systems. Finally, Section 2.5 provides a summary of the chapter.

## 2.2 Dynamical definitions

In the previous section, we gave a brief history of dynamical systems. In this section, we define some important dynamical terms that will be used in the remaining chapters. We start by defining the dynamical system.

### 2.2.1 Dynamical System

The dynamical system is a system that consists of a sequence of states which are governed by a certain rule that determines the next state given the previous one. A dynamical system in $\Re^d$ can be described by either $d$ first order ordinary differential equations (*flow*) or $d$ difference equations (*map*). In the first case, the time is a continuous variable; $t \in \Re^1$, and the system is called a continuous dynamical system:

$$\frac{d(\mathbf{x}(t))}{dt} = \boldsymbol{f}(\mathbf{x}(t)). \tag{2.1}$$

In the second case, the time is a discrete variable; $n \in \aleph$, and the system is called a discrete dynamical system:

$$\mathbf{x}(n+1) = \boldsymbol{f}(\mathbf{x}(n)). \tag{2.2}$$

### 2.2.2 Equilibrium point

A point $\mathbf{x}(t) = \mathbf{x}^*$ in the state space is said to be an *equilibrium point* of a continuous dynamical system where $\left.\frac{d\mathbf{x}(t)}{dt}\right|_{\mathbf{x}(t)=\mathbf{x}^*} = \mathbf{0}$. The *equilibrium point* of a discrete dynamical system, on the other hand, happens when $\boldsymbol{f}(\mathbf{x}^*) = \mathbf{x}^*$.

### 2.2.3 Orbit of a dynamical system

The sequence of points $\mathbf{x}(t)$ or $\mathbf{x}(n)$ that results from the solution of the set of equations representing the system is called the *orbit* (*trajectory*) of the dynamical system. The point $\mathbf{x}_0$ or $\mathbf{x}(0)$ is called the *initial condition* of the system.

### 2.2.4 Attractor

The attractor of a dynamical system is set containing the limits of all orbits that start sufficiently close to it.

## 2.3 Characteristics of Chaotic Systems

In this section, we explore three characteristics of chaotic systems. These characteristics will help us to understand how to model these systems.

### 2.3.1 Sensitivity of chaotic systems to initial conditions

Chaotic systems are known for their sensitivity to initial conditions. To illustrate this idea, let's look into the tent map, which is a first order chaotic system:

$$x(n+1) = f(x(n)) = (3/4)(1 - |1 - 2x(n)|). \tag{2.3}$$

Figure 2.1 shows two curves (solid, and dotted with 'x') representing the responses of the system when two sightly different initial conditions are used. The solid curve was produced from the initial condition $x_1(0) = 0.23$ and the dotted curve with 'x' was produced from

$x_2(0) = 0.2301$. The dotted curve in the bottom of the figure is the difference between the two responses. We can see that the difference starts to grow after approximately 15 iterations. Notice that the responses of the system due to different initial conditions remain within a bounded region.

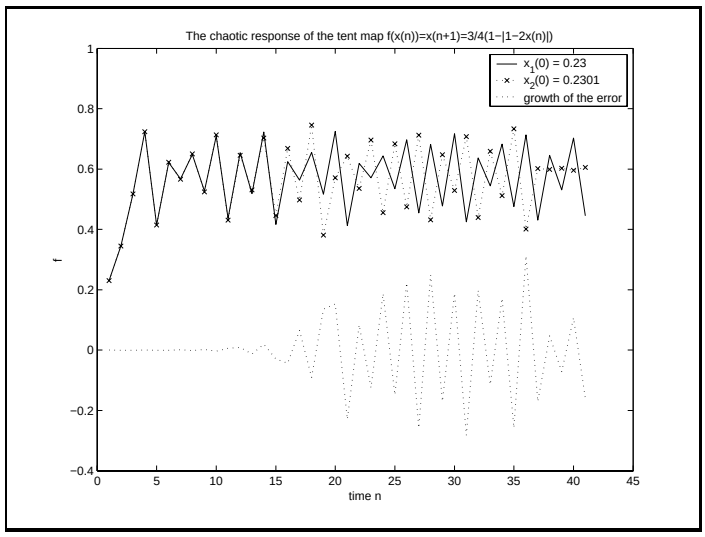



Figure 2.1 Sensitivity of the chaotic tent map to initial conditions.

### 2.3.2 Chaotic signals look random but they are deterministic

Before the discovery of chaotic systems, scientists thought of chaos as a random signal (noise). To illustrate this idea, let's look into the frequency responses (Fourier spectrums [KaSc00]) of a periodic signal and a random signal, and then compare them to a chaotic signal. The left hand side of Figure 2.2 shows a 60 Hz periodic sinusoidal wave. The plot on the right hand side is its frequency response. As we can see, it has only one component at 60 Hz.

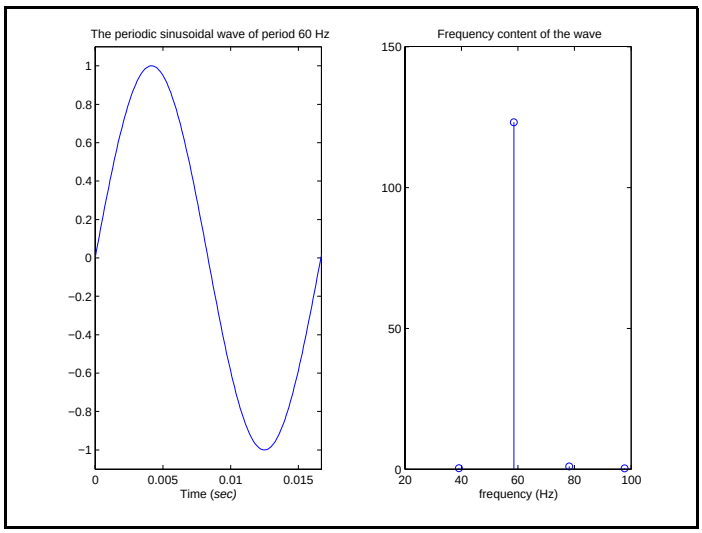



Figure 2.2 The 60 Hz periodic wave and its frequency response

If we look into the frequency response of a random signal, shown in the right hand side of Figure 2.3, we can see that it has a component at every frequency from 0 through 120 Hz.

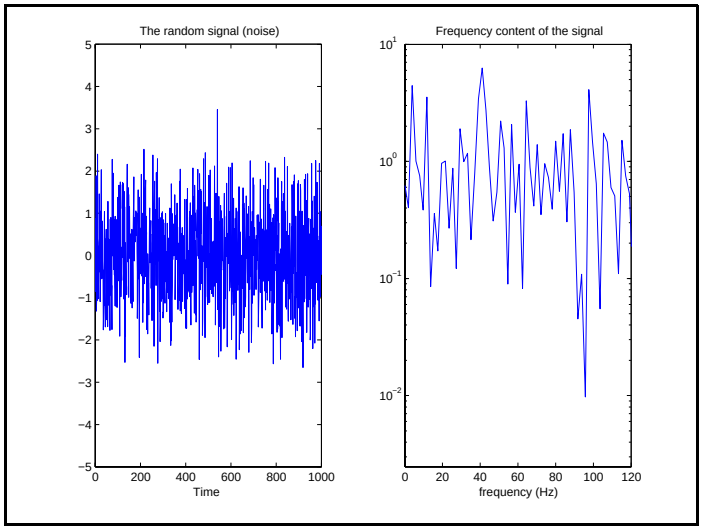



Figure 2.3 The random signal and its frequency response

We will now compare the frequency responses of the two signals above to the frequency response of a chaotic signal. The left hand side of Figure 2.4 shows the time domain plot of a chaotic signal (*x* component of the Lorenz model). Its frequency response is shown on the right hand side of the same figure. We can see that the chaotic signal has a component at every frequency from 0 through 120 Hz, which is similar to the random signal.

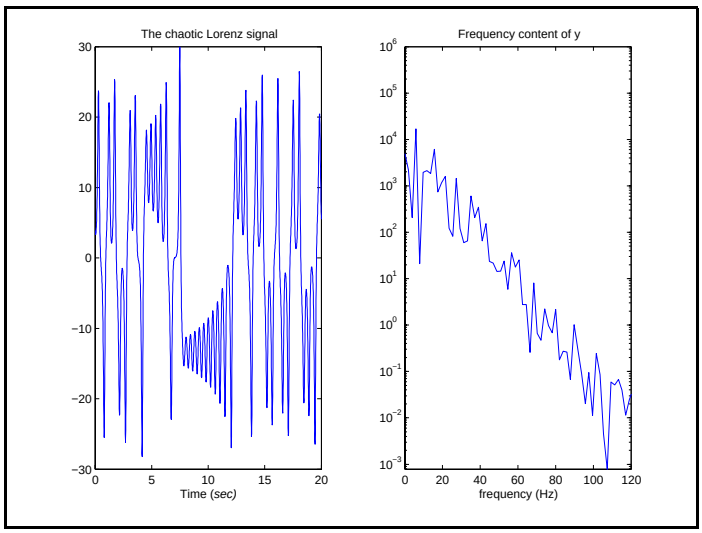



Figure 2.4 The chaotic signal and its frequency response

Although the frequency response of the chaotic system looks random, we know that this system is deterministic. This is true since it is produced from a known set of differential equations.

### 2.3.3 Chaotic systems have attractors with fractal dimension

To see that chaotic attractors have fractal dimensions, let's look into the Henon map:

$$x_1(n+1) = 1 - a(x_1(n))^2 + x_2(n), \tag{2.4}$$

$$x_2(n+1) = bx_1(n), \tag{2.5}$$

where $a = 1.4$, $b = 0.3$, and the initial conditions are $x_1(0) = x_2(0) = 0.5$ [Hen76]. In Figure 2.5, we can see that the attractor of the Henon map is not a simple line of dimension 1, nor it is a closed curve. It is also not a plane of dimension 2. It is an object that fills a region in the plane.

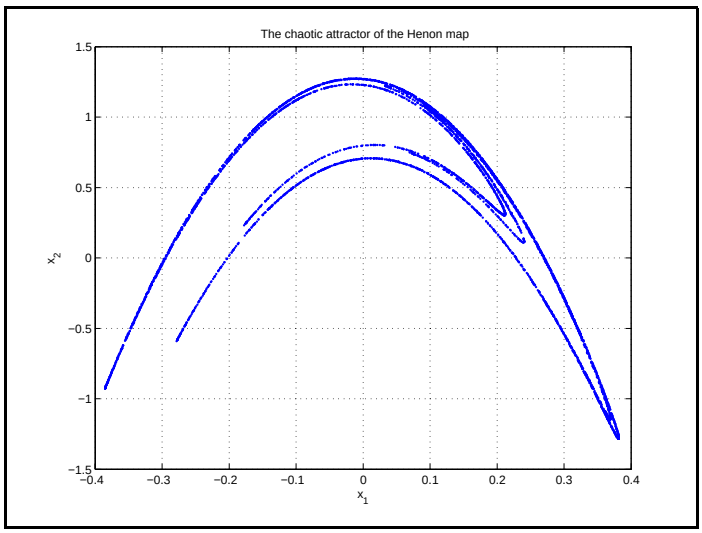



Figure 2.5 The chaotic attractor of the Henon map.

As a matter of fact, its dimension is not an integer, but rather is fractal. The dimension of the attractor of the Henon map can be found by the box-counting dimension.

2.3.3.1 The box-counting dimension:

We can find the dimension of an interval $N = [0, 1]$, in $\Re^1$, by dividing it into subintervals $N_j$ of length $\sigma = 0.1$ such that $N_j \cap N_{j+1} = \varnothing$, where $j \in \aleph$. Then the minimum number of $N_j$ subintervals needed to cover $N$ is $c_\sigma^N = 10$, which can be written as $c_\sigma^N = (1/\sigma)^1$. Notice that the exponent in the expression is 1, which is also the dimension of $N$. For the case of a unit square in $\Re^2$, $S = \left\{ \begin{bmatrix} x \\ y \end{bmatrix} \in \Re^2 \middle| 0 \le x \le 1, 0 \le y \le 1 \right\}$, we can find the dimension of $S$ by dividing it into small squares $S_j$ of side length $\sigma = 0.1$ (called $\sigma$-squares), such that $S_j \cap S_{j+1} = \varnothing$. Then the minimum number of $S_j$ squares needed to cover $S$ is $c_\sigma^S = 100$, which can be written as $c_\sigma^S = (1/\sigma)^2$. Notice that the exponent in the expression is 2 which is also the dimension of $S$. The method used to find the

dimension of the sets in the previous two examples is called the box-counting dimension. It is used to find the dimension of more complicated sets in $\Re^d$, like fractal sets. The box-counting dimension is denoted by $d_c$ which is the exponent in the relation:

$$c_\sigma^d \approx (1/\sigma)^{d_c} \text{ as } \sigma \to 0, \tag{2.6}$$

where $c_\sigma^d$ is the number of $\sigma$ $d$-cubes needed to cover the set in $\Re^d$. Taking the limit of the log of Equation (2.6) as $\sigma \to 0$, we have

$$d_c = \lim_{\sigma \to 0} \frac{\ln c_\sigma^d}{\ln(1/\sigma)}. \tag{2.7}$$

To evaluate the box-counting dimension ($d_c$) in Equation (2.7), we draw a *log-log* plot of $c_\sigma^d$ versus $1/\sigma$ as $\sigma \to 0$. The value of $d_c$ is the slope of the resulting curve.

Now we can apply the box-counting dimension to the attractor of the Henon map. We start by choosing the side length of the $\sigma$-squares to be some value $\sigma < 1$. Then we count the minimum number of squares needed to cover the set of points in the attractor (which results from iterating equations (2.4) and (2.5) $10^5$ times). Next we decrease $\sigma$, and count the minimum number needed to cover the attractor for the new $\sigma$. Figure 2.6 shows the number of $\sigma$-squares ($c_\sigma^d$) versus $\sigma$, for $\sigma = \{0.1, 0.05, 0.01, 0.004, 0.002, 0.001\}$.

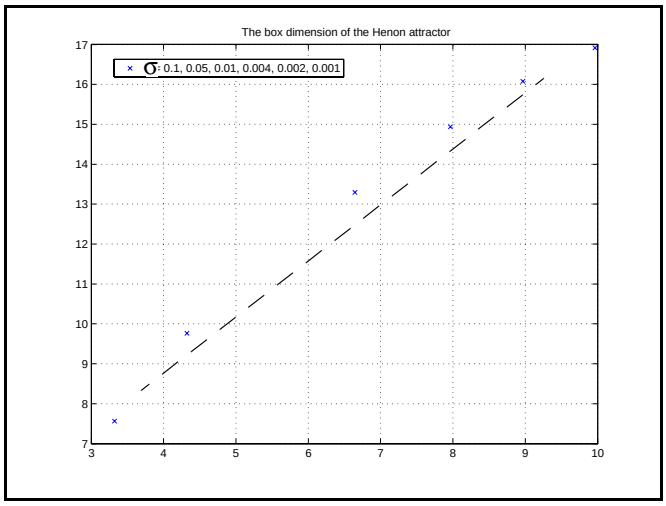



Figure 2.6 The slope of the line is the box-counting dimension of the Henon map attractor.

The resulting box-counting dimension of the Henon map was found to be $d_c \approx 1.189$, which is a fractal. After presenting three characteristics of the chaotic systems, in the next section we list some examples of chaotic systems.

## 2.4 Examples of Chaotic systems

### 2.4.1 Weather system

We said in Section 2.1 that E. Lorenz was the first scientist to quantify chaos. He modelled the heat convection phenomena in fluids by using a set of three differential equations. His model represents earth's atmosphere. He used it to forecast weather.

### 2.4.2 Biological models

Many biological activities exhibit chaotic behaviors. One example is the Epileptic seizure. Epilepsy causes patients suffering from this disease to experience bouts of unconsciousness. Epileptic seizures result from an abnormal neuronal discharge from the central nervous system [Zyl01]. Another example of chaotic systems is the red blood cell production. Mackey and Glass [MG77] modelled red blood cell production and found that it ex-

hibits chaotic behavior at some parameter values of the model, as will be shown in Chapter 7. A third example of chaotic systems is the Electrocardiogram (ECG). In 1983, Glass *et al* [GGS83] experimented on the spontaneous beating of cells from embryonic chick hearts. They found that when these cells were stimulated by external periodic pulses, they showed chaotic motion (see also [Moo92]).

### 2.4.3 Laser signals

Measurements taken from a laser output can be represented by a chaotic model. In Chapter 7, we will give more detail regarding the modeling of two data sets of laser outputs.

### 2.4.4 Chaotic circuits

Chaos can be observed in electrical circuits as well. An example of this is the RLC circuit designed by Rulkov *et al* [RVRDV92].

### 2.4.5 Discrete chaotic systems:

In the previous section, we have shown two discrete chaotic systems: the tent and the Henon maps. Discrete chaotic systems are mainly used for analysis. In Chapter 7, we will show more discrete chaotic systems used to analyze chaos of different dimensions.

## 2.5 Chapter Summary

In this chapter, we presented a historical background for dynamical systems. We defined some dynamical terms that will be used in subsequent chapters. Three characteristics of chaotic systems were illustrated with examples. We also explored a few examples of dynamical systems that show chaotic behaviors. In the next chapter, we will define dy-

namical modeling, and show how to implement modeling of chaotic systems by using measurements taken from these systems.

# CHAPTER 3

# INTRODUCTION TO DYNAMICAL MODELING

## 3.1 Introduction

In Chapter 2, we introduced dynamical chaotic systems. We have seen three characteristics of chaotic systems as well. We mentioned that these characteristics will help us to understand modeling of chaotic systems. In this chapter, we explore the modeling process. We also show some of its applications in real life. A theoretical background of the method used for modeling chaotic systems is illustrated with examples.

In the next section, we present a brief introduction to modeling chaotic systems. Some applications of the modeling process are presented in Section 3.3. We do modeling by a technique called embedding. Some mathematical background on embedding is contained in Section 3.4. The use of the delay-coordinate map for modeling is illustrated in Section 3.5 with two examples.

## 3.2 Modeling

We said in Chapter 2 that dynamical systems can be represented as a set of differential equations (for a continuous system), or a set of difference equations (for a discrete system). We also said that these equations determine the evolution of states, which converges to an attractor of the system. Knowing the evolution of the states is important for understanding the behavior of the system at a future time. Unfortunately these equations,

in most cases, are not known. All we can see from a chaotic system (like those listed in Chapter 2) is a set of scalar measurements. Modeling chaotic systems entails describing the hidden states of the system from these measurements only. From the set of measurements, the modeling process builds a new set of states, which are in some sense equivalent to the original hidden states. In Chapter 8, we will give more detail regarding the meaning of equivalent chaotic systems.

## 3.3 Applications of modeling

### 3.3.1 Detection of chaos

In Section 2.3.2, we stated that, before the quantification of chaos, scientists thought of chaos as a random signal (noise). Given a set of scalar measurements, the modeling process enables us to distinguish between chaotic systems (deterministic) and random signals (noise). Not only that, but modeling can also extract the hidden deterministic part from a noisy signal. This application of the modeling process is important because, in most cases, the set of measurements is contaminated with noise from different sources.

### 3.3.2 Prediction

The ability to predict the future has fascinated scientists for a long time. Modeling chaotic systems enables us to predict the hidden future states of the system. It does so using only the set of measurements. However, as we have said in Section 2.3.1, chaotic systems have sensitive dependence on the initial conditions. This problem limits the ability of chaotic modeling to make long term predictions of future states. In Chapter 8, we will give more detail.

### 3.3.3 Diagnosis

Abarbanel [Aba98] conducted an experiment on the ECG of subjects undergoing a stress test for a specific pathology. He found that in the extreme pathology of ventricular fibrillation, characteristics of the model are different from those of a healthy person. This means the modeling process can be used to diagnose life threatening diseases (for more examples, see [Hol86 Chapters 9 and 11]).

In the next two sections, we present some mathematical definitions, then we present the embedding method which is used to build models of chaotic systems.

## 3.4 Definitions

Before we present the embedding method, let's give a mathematical background of embedding functions. We begin by defining an injection function and an immersion function, then we define the embedding function.

### 3.4.1 Injection (1-to-1) function

Let $M$ and $N$ be two sets, a function $\boldsymbol{g}: M \to N$ is an injection (1-to-1) if

$$\boldsymbol{g}(\mathbf{x}^1) = \boldsymbol{g}(\mathbf{x}^2) \Rightarrow \mathbf{x}^1 = \mathbf{x}^2, \tag{3.1}$$

where $\mathbf{x}^1, \mathbf{x}^2 \in M$. To understand the injection function, consider the next example.

3.4.1.1 Example: An injection

Let the function $\boldsymbol{g}: \Re^2 \to \Re^2$ be

$$\boldsymbol{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) & g_2(\mathbf{x}) \end{bmatrix}^t = \begin{bmatrix} x_1 & (x_2)^3 \end{bmatrix}^t, \tag{3.2}$$

where $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \in \Re^2$. By applying the condition for an injection, we have

$$\boldsymbol{g}(\mathbf{x}^1) = \boldsymbol{g}(\mathbf{x}^2) = \left[x_1^1 \ (x_2^1)^3\right]^t = \left[x_1^2 \ (x_2^2)^3\right]^t. \tag{3.3}$$

From the first element of the vectors on the right hand side we have

$$x_1^1 = x_1^2, \tag{3.4}$$

while from the second element of the vectors we have

$$(x_2^1)^3 = (x_2^2)^3 \Rightarrow x_2^1 = x_2^2. \tag{3.5}$$

In conclusion, we can see that $\mathbf{x}^1 = \mathbf{x}^2$, so the function $\boldsymbol{g}$ is an injection. On the other hand, let's look at an example of a function which is not an injection.

3.4.1.2 Example: A function that is not injection

Let the function $\boldsymbol{g}$: $\Re^2 \to \Re^2$ be

$$\boldsymbol{g}(\mathbf{x}) = \left[x_1 \ (x_2)^2\right]^t. \tag{3.6}$$

To test the condition for injection, we have

$$\boldsymbol{g}(\mathbf{x}^1) = \boldsymbol{g}(\mathbf{x}^2) = \left[x_1^1 \ (x_2^1)^2\right]^t = \left[x_1^2 \ (x_2^2)^2\right]^t. \tag{3.7}$$

From the first element of the vectors on the right hand side we have

$$x_1^1 = x_1^2. \tag{3.8}$$

From the second element of the vectors we have

$$(x_2^1)^2 = (x_2^2)^2 \Rightarrow x_2^1 = \pm x_2^2. \tag{3.9}$$

Which means that $\mathbf{x}^1$ does not have to equal $\mathbf{x}^2$, so the function $\boldsymbol{g}$ is not an injection.

### 3.4.2 Immersion function

Let $M$ and $N$ be two sets, a function $\boldsymbol{g}: M \to N$ is an immersion if its Jacobian matrix is an injection (full rank) [Nak90 p.149].

3.4.2.1 Example: An immersion

Let the function $\boldsymbol{g}: \Re^2 \to \Re^2$ be

$$\boldsymbol{g}(\mathbf{x}) = \left[ g_1(\mathbf{x}) \; g_2(\mathbf{x}) \right]^t = \left[ x_1 \;\; (x_2)^3 + x_2 \right]^t, \tag{3.10}$$

where $\mathbf{x} \in \Re^2$. The Jacobian matrix of $\boldsymbol{g}$ is

$$\mathbf{J} = \left[ \begin{matrix} 1 & 0 \\ 0 & 3(x_2)^2 + 1 \end{matrix} \right] \Bigg|_{\mathbf{x} = \mathbf{x}_0}. \tag{3.11}$$

The determinant of $\mathbf{J}$ is $3(x_2)^2 + 1 \neq 0$, therefore $\mathbf{J}$ is full rank. Hence, the function $\boldsymbol{g}$ is an immersion.

3.4.2.2 Example: A function that is not immersion

Let the function $\boldsymbol{g}: \Re^2 \to \Re^2$ be

$$\boldsymbol{g}(\mathbf{x}) = \left[ x_1 \;\; (x_2)^3 \right]^t, \tag{3.12}$$

where $\mathbf{x} \in \Re^2$. The Jacobian matrix of $\boldsymbol{g}$ is

$$\mathbf{J} = \left[ \begin{matrix} 1 & 0 \\ 0 & 3(x_2)^2 \end{matrix} \right] \Bigg|_{\mathbf{x} = \mathbf{x}_0}. \tag{3.13}$$

The determinant of **J** is $3(x_2)^2$, which zero for $x_2 = 0$. Therefore **J** is not full rank. Hence $\boldsymbol{g}$ is not an immersion. Now that we have defined the injection and the immersion functions, we will define the embedding function.

### 3.4.3 Embedding function

Let *M* and *N* be two sets, a function $\boldsymbol{g}: M \rightarrow N$ is an embedding if it is an injection and an immersion [Nak90].

3.4.3.1 Example: An embedding

Let's look at the function $\boldsymbol{g}$ in Example 3.4.2.1 ($\boldsymbol{g}(\mathbf{x}) = \begin{bmatrix} x_1 & (x_2)^3 + x_2 \end{bmatrix}^t$). In that example, we have seen that $\boldsymbol{g}$ is an immersion. To prove that it is an embedding, we need to show that it is also an injection. To do this, we start by assuming that

$$\boldsymbol{g}(\mathbf{x}^1) = \boldsymbol{g}(\mathbf{x}^2) = \begin{bmatrix} x_1^1 & (x_2^1)^3 + x_2^1 \end{bmatrix}^t = \begin{bmatrix} x_1^2 & (x_2^2)^3 + x_2^2 \end{bmatrix}^t. \tag{3.14}$$

If we look at the first element of the vectors on the right hand side of Equation (3.14), we can see that $x_1^1 = x_1^2$. While the second elements give

$$(x_2^1)^3 + x_2^1 = (x_2^2)^3 + x_2^2. \tag{3.15}$$

By rearranging the above equation, we have

$$(x_2^1)^3 + x_2^1 - (x_2^2)^3 - x_2^2 = 0. \tag{3.16}$$

Which can be written as

$$(x_2^1 - x_2^2)((x_2^1)^2 + (x_2^2)^2 + x_2^1 x_2^2 + 1) = 0. \tag{3.17}$$

By looking at the second parenthesis of the left hand side of Equation (3.17), we can see that it can be simplified as follows

$$\left(x_2^1 + \frac{1}{2}x_2^2\right)^2 + \frac{3}{4}(x_2^2)^2 + 1 \neq 0 \ \forall \mathbf{x}. \tag{3.18}$$

So the only solution of Equation (3.17) is $x_2^1 = x_2^2$. As a result, we see that $\mathbf{x}^1 = \mathbf{x}^2$. Which means the function $\boldsymbol{g}$ is an injection. Since $\boldsymbol{g}$ is an injection and an immersion (Example 3.4.2.1), we see that $\boldsymbol{g}$ is an embedding.

3.4.3.2 Example: A non-embedding

From Example 3.4.1.1, we have seen that $\boldsymbol{g}(\mathbf{x}) = \left[x_1 \ (x_2)^3\right]^t$ is an injection. But from Example 3.4.2.2, we have seen that this function is not an immersion. So the function $\boldsymbol{g}$ is not an embedding.

Notice that the main feature of the embedding is that it is an injection, and its linearization (Jacobian matrix) at every point along the attractor is also an injection. After presenting the embedding, let's see now how we can use it for modeling. As we said above, we have only a set of scalar measurements from the system we want to model. To build a model for the system, we will use the delay-coordinate map, which is explained below.

## 3.5 Modeling by Embedding

### 3.5.1 The delay-coordinate map

Let the state of the original system that we want to model be $\mathbf{x} \in \Re^k$, and let the measurements taken from this system be governed by the map $h: \Re^k \rightarrow \Re^1$ such that

$$y(n) = h(\mathbf{x}(n)). \tag{3.19}$$

The delay-coordinate map $F_d$: $\Re^k \rightarrow \Re^d$ [SYC91] is represented by

$$\mathbf{y}_d(n) = F_d(h, \mathbf{x}(n)) = \left[y(n)\ y(n-T)\ \ldots\ y(n-(d-1)T)\right]^t, \tag{3.20}$$

where $T \in \aleph$ is the delay-time, and $\mathbf{y}_d(n) \in \Re^d$ is the delay-vector. The attractor built from the delay-vectors is called the reconstructed attractor.

According to the embedding theorem by Sauer *at al* [SYC91], the function $F_d$ is an embedding if $d \geq 2d_c + 1$, where $d_c$ is the box-counting dimension (see Section 2.3.3.1) of the attractor of the original system. We will call the minimum dimension *d* that satisfies the embedding condition, the theoretical minimum embedding dimension: $d_E$. The embedding map guarantees that evolution of the states in the original unknown attractor is equivalent to the evolution of the delay-vectors in the reconstructed attractor (Chapter 8). In the next two examples, we illustrate the modeling process by using the delay-coordinate map.

### 3.5.2 Example: The delay-coordinate map

Let the set $C = \{\mathbf{y} \in \Re^3 \,|\, y_1 = \cos t, y_2 = \sin 2t, y_3 = \sin t \cos 2t,\ t \in \Re^1\}$ represent an attractor in $\Re^3$. Further, let the measurement function $h$:$\Re^3 \rightarrow \Re^1$, for the purpose of explanation, be $y = h(\mathbf{x}) = x_1 + x_2 + x_3$, where $\mathbf{x} \in \Re^3$. The delay-coordinate map operating on *C* will be: $\mathbf{y}_3(n) = F_3(h, \mathbf{x}(n)) = \left[y(n)\ y(n-1)\ y(n-2)\right]^t$ $(T = 1)$. The original attractor *C* is shown in Figure 3.1*a*, the measurement function *h* is shown in Figure 3.1*b*, and the reconstructed attractor using $F_3$ is shown in Figure 3.1*c*.

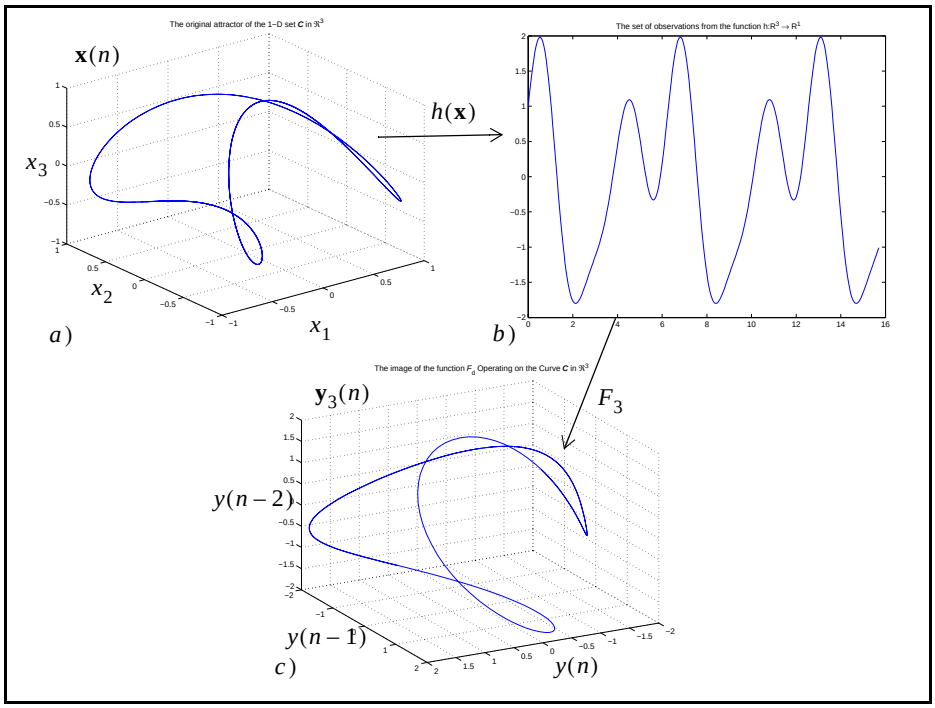



Figure 3.1 *a*) The attractor *C* in $\Re^3$, *b*) the set of measurements, *c*) The reconstructed attractor by the delay-coordinate map.

As we can see, this is a curve of dimension 1 in a three dimensional space. According to the embedding condition; $d_E = 2x1 + 1 = 3$. Which is the same as the dimension of the space required to see the curve *C* without ambiguity.

### 3.5.3 Example: Sufficient but not necessary condition for embedding

Let the circle; $S^1 = \{\mathbf{y} \in \Re^3 \,|\, y_1 = \cos t, y_2 = \sin t, y_3 = 0.5, t \in \Re^1\}$ represent an attractor in $\Re^3$. By using the delay-coordinate map, we can reconstruct $S^1$ in $\Re^3$. The attractor $S^1$ is shown in Figure 3.2 as a solid curve. The reconstructed attractor using the delay-coordinate map ($F_3$) is shown in the same figure as a dashed curve, the delay-time (*T*) is 1.

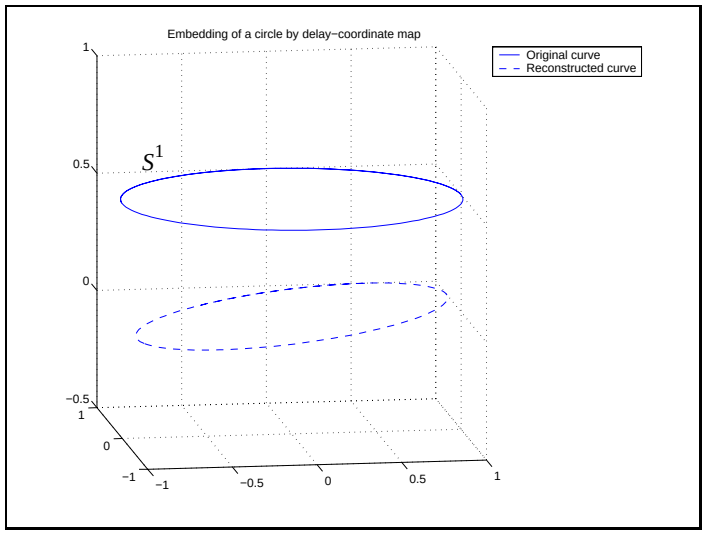



Figure 3.2 The circle $S^1$ in $\Re^3$ and its reconstruction in $\Re^2$.

In Figure 3.2, we can see that $S^1$ is a curve of dimension 1 in a 3D space. Notice that we needed a 2D space to reconstruct this curve. This means the dimension required for embedding is 2 rather than 3, as suggested by the embedding condition ($d = 2 < d_E = 3$). In conclusion, we can see that the embedding condition gives a sufficient but not a necessary condition for embedding. In other words, it is possible that we can embed an attractor in a space of dimension less than $d_E$. In the next chapter, the delay-coordinate map will be used to model chaotic systems. We will also present four methods used to find the actual, rather than the theoretical, minimum dimension required for embedding. The choice of $T$ (the second parameter in the delay-coordinate map), will be shown in Chapter 6.

## 3.6 Chapter summary

In this chapter, we presented an introduction to modeling chaotic systems. Some applications of modeling chaotic systems were given, mathematical definitions of the embedding functions were illustrated with examples, and we showed two examples of modeling

using the embedding method. We found that the embedding condition is a sufficient but not necessary condition. In the next chapter, we show different methods used to find the minimum dimension required for embedding.

# CHAPTER 4

# INTRODUCTION TO THE MINIMUM EMBEDDING DIMENSION ESTIMATION

## 4.1 Introduction

In Chapter 3, we talked about modeling by the embedding method. Modeling is done from a time series of measurements taken from the system. We said that two parameters have to be found in order to model the system by embedding. Those parameters are the dimension of the delay-vector ($d$) and the delay-time ($T$). They will be used to build the delay-vectors.

In this chapter we introduce four methods for estimating the value of $d$. Three of these methods are geometric and one method is predictive. This chapter will provide only a simple overview of the methods. Chapters 5 and 6 will provide more detail. In the next section, we present modeling of chaotic systems by using the delay-vectors. In Section 4.3, we introduce two different approaches for estimating $d$: geometric and predictive. Then, in Section 4.4, we introduce equivalent chaotic systems. A chapter summary is given in Section 4.5. In Chapter 6, we will discuss the selection of the delay-time $T$.

## 4.2 Modeling chaotic systems

The state evolution of a chaotic system in the space $\Re^k$ can be written in the form of the difference equation:

$$\mathbf{x}(m) = \boldsymbol{f}(\mathbf{x}(m-1)), \tag{4.1}$$

where $\mathbf{x}(m) \in \Re^k$ is the state of the system, the time index $m = 1, 2, \ldots, N$, and $N$ is the total number of points. In the steady state condition, the evolution of the state $\mathbf{x}(m)$ follows an attractor with a fractal dimension $d_c$ (chaotic attractor). Practically, the state $\mathbf{x}(m)$ and the function $\boldsymbol{f}$ of the system are invisible to us, and all we can see is a set of scalar measurements $y(m)$. We can write these measurements as $y(m) = h(\mathbf{x}(m))$ (see Equations (3.19). To model a chaotic system, we need to build a system model which uses the delay-vector

$$\mathbf{y}_d(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d-1)T)\right]^t. \tag{4.2}$$

The attractor of the $\mathbf{y}_d(m)$ system should be equivalent to the original attractor, and the state evolution from $\mathbf{y}_d(m) \rightarrow \mathbf{y}_d(m+1)$ should follow that of the original attractor from $\mathbf{x}(m) \rightarrow \mathbf{x}(m+1)$ [Hak98]. (A more careful definition of equivalence is contained in Chapter 8.) The key idea is to find *d* and *T* such that the two systems are equivalent. The embedding theorem guarantees this equivalence if $d \geq 2d_c + 1$. But in practice, we do not generally know the value of $d_c$, so *d* has to be estimated. The projection from the original state $\mathbf{x}$ to the delay-vector $\mathbf{y}_d$ is called the delay-coordinate map. As we said in the previous chapter, the embedding theorem gives a sufficient, but not a necessary, condition for embedding. In other words, it could be possible to find an embedding map at $d \leq 2d_c$. Our goal is to estimate the minimum embedding dimension. We will label this dimension $d_L$. In the next section, we will talk about estimating $d_L$ by using two techniques: a geometric and a predictive technique. In Chapter 6, we will discuss the delay-time *T* to complete the modeling process.

## 4.3 Estimating the minimum embedding dimension

### 4.3.1 The geometric technique

To understand the geometric technique, let the circle in Figure 4.1 represent an attractor of a dynamical system in $\Re^2$, and the line below it represent its projection into $\Re^1$.

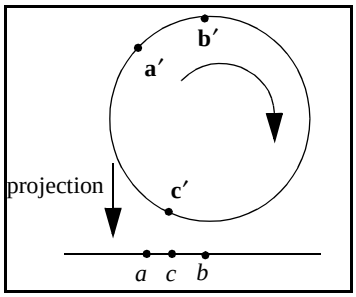



Figure 4.1 Projection artifacts

From the above figure, we can see that the points on the horizontal line (*a*, *b*, and *c*) do not occur in the same sequence as those on the circle (**a′**, **b′**, and **c′**). That happened because the circle was projected into a space with insufficient dimension. That caused the distances between points to shrink, and the original order of points to change. We call this effect the projection artifact. We can see from this example that, if the dimension of the space is too small to represent the attractor of the system, the projection of the attractor to this space will have artifacts. Therefore, we need to increase the dimension of the space in order to get rid of these artifacts.

One technique of estimating the minimum embedding dimension $d_L$ is the geometric technique. To estimate $d_L$ by using the geometric technique, we start from $d = 1$ and find for every point $\mathbf{y}_d(m)$ its nearest neighbor. This neighbor has to be tested to see if it is a true neighbor or if it became a neighbor because of some projection artifacts. After that, *d* is increased and the previous steps are repeated. The value of *d* where all the neighbors are true neighbors, is the minimum embedding dimension $d_L$. Under the geometric tech-

nique, three methods can be used to detect the existence of projection artifacts. These methods are 1) the Change of Neighbors with Dimension method, or CND, 2) the Change of Distance with Dimension method, or CDD, and 3) the Change of Distance with Time method, or CDT.

4.3.1.1 The change of neighbors with dimension method (CND)

One can perform a visual test using Figure 4.1 to determine that the horizontal line does not have enough dimension to represent the structure of the circle, and that we need to have a two dimensional space (meaning that $d_L = 2$). However, we need to automate this test using an algorithm. To estimate $d_L$ using the first geometric method, the change of neighbors with dimension method (CND), the algorithm starts from the scalar measurements *a*, *b*, and *c*, and computes the distances between point *a* and the other two points. It will find that the nearest neighbor to *a* is *c*, while on the circle, the nearest neighbor to $\mathbf{a}'$ is $\mathbf{b}'$. The algorithm concludes that the neighbors have changed, and that the points *a* and *c* became neighbors because of the projection artifact and not because they are true neighbors. That means the 1-D space is not large enough to represent the structure of the circle since it has this projection artifact. By repeating the above steps on the 2-D space, the algorithm would find that there are no projection artifacts left on the attractor in this space and concludes that $d_L = 2$.

4.3.1.2 The change of distance with dimension (CDD) method

The second geometric method that can be used to estimate $d_L$ is the change of distance with dimension method (CDD). The CDD method is similar to the CND method except that it detects the existence of projection artifacts by comparing the distances between neighbors rather than comparing the neighbors themselves. To understand this method, let

the curve in Figure 4.2 represent an attractor of a dynamical system in $\Re^2$, and the line below it represent its projection into $\Re^1$.

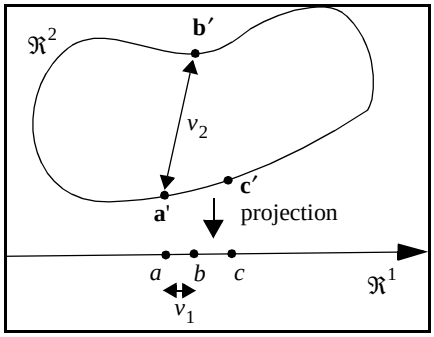



Figure 4.2 The CDD method

We can see the points **a'**, **b'**, and **c'** on the curve in $\Re^2$ and their projection into the horizontal line *a*, *b*, and *c* respectively. As in the CND method, we want an algorithm to find that the 1-D space is not large enough to represent the structure of the curve. From the horizontal line, the algorithm can find that *b* is the nearest neighbor of *a* and the distance between them in $\Re^1$ is $v_1$. In $\Re^2$, along the curve, the distance between **a′** and **b′** is $v_2$. The algorithm can now compare the two distances to find that $v_1 \ll v_2$, therefore, *b* became a neighbor of *a* because of the projection artifact and not because they are true neighbors. As a result, the algorithm will find that the 1-D space is not large enough to represent the structure of the curve since it has this projection artifact. By repeating the above steps on the 2-D space, the algorithm would find that there are no projection artifacts left on the attractor in this space and concludes that $d_L = 2$.

4.3.1.3 The change of distance with time method (CDT)

The third geometric method used to estimate $d_L$ is the change of distance with time method (CDT). This method is different from the previous two in that it tries to detect the existence of projection artifacts by moving neighbors forward in time and checking to see

if they will remain neighbors. To understand this method, let the non-intersecting solid curve in $\Re^3$ shown in Figure 4.3 represent an attractor of a dynamical system, and let the dotted curve with the "+" sign represent its projection into the X-Y plane ($\Re^2$ space).

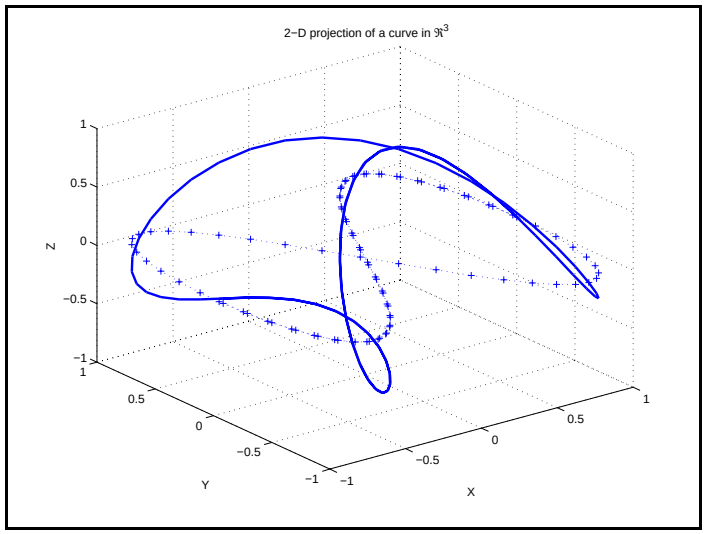



Figure 4.3 The projection of a 3-D curve (solid line) into 2-D curve (dotted line)

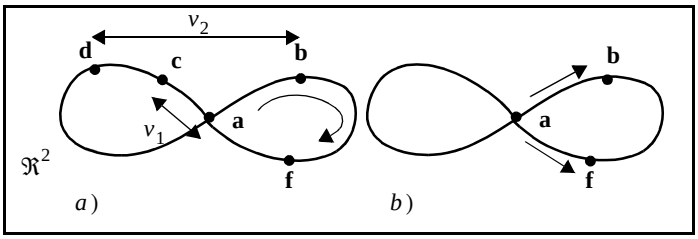



Figure 4.4 The CDT method: *a*) Distances between false neighbors increase with time, *b*) The intersection in the middle of the curve is a projection artifact

The projection of the systems's attractor shown in Figure 4.3 is redrawn in Figure 4.4 for further discussion. Notice that the middle of the curve in Figure 4.4*a* has a projection artifact, which can be found by looking into Figure 4.4*b*. If we start from point **a**, we can't tell whether the next correct point in time is **b** or **c**. In other words, the curve intersection makes it impossible to determine the correct sequence of points.

By using the CDT method, we want an algorithm to be able to find that the 2-D space is not large enough to represent the structure of the system's attractor (which is in $\Re^3$: $d_L = 3$). This can be done by detecting the projection artifact in the middle of the curve. To do that, the algorithm should start by calculating the distance between point **a** and the other points on the curve. It will find that **c** is closest to **a**; let the distance between them be $v_1$. Next it should move forward in time to find that **a** moves to **b**, and **c** moves to **d**. It should then compute the distance between **b** and **d**; let the distance be $v_2$. Now it can compares the two distances and find that $v_1 \ll v_2$, which means that **a** and **c** are neighbors because of some projection artifacts and not because they are true neighbors. (See Figure 4.4*a*) Hence the algorithm concludes that the 2-D space is not large enough to represent the structure of the system's attractor since the curve has this projection artifact. By repeating this for the 3-D space, the algorithm would find no projection artifacts left on the curve in this space and would conclude that $d_L = 3$.

### 4.3.2 The predictive technique:

Now that we have talked about the three geometric methods, we will discuss the predictive technique. The predictive technique is also used to estimate the minimum embedding dimension $d_L$ such that the system of $\mathbf{y}_{d_L}(m)$ (using delay embedding) is equivalent to the system of $\mathbf{x}(m)$ (original system). For simplicity, we will set $T = 1$ in the delay-vector $\mathbf{y}_d(m)$ (see Equations (4.2)). By using the delay-coordinate map; $F_d$, we can write $\mathbf{y}_d(m) = F_d(\mathbf{x}(m))$ (see Equations (3.20)). From the embedding definition (see Section 3.4.3), we know that if the map $F_d$ is an embedding, then it is an injection. Hence, the inverse map $F_d^{-1}: \Re^d \to \Re^k$ exists:

$$\mathbf{x}(m) = F_d^{-1}(\mathbf{y}_d(m)). \tag{4.3}$$

We can substitute Equations (4.1) for the measurement function $h$ at Equations (3.19) ($y(m) = h(\mathbf{x}(m))$) to write $h$ as $y(m) = h(\boldsymbol{f}(\mathbf{x}(m-1)))$. If we let the composite function $h \circ \boldsymbol{f} = q$, we can write

$$y(m) = q(\mathbf{x}(m-1)). \tag{4.4}$$

By substituting Equations (4.3) for Equations (4.4), we can write

$$y(m) = q(F_d^{-1}(\mathbf{y}_d(m-1))). \tag{4.5}$$

Now, if we let the composite function $q \circ F_d^{-1} = \mu$, we can write

$$y(m) = \mu(\mathbf{y}_d(m-1)) = \mu\left(\left[y(m-1)\ y(m-2)\ \ldots\ y(m-d)\right]^t\right). \tag{4.6}$$

We conclude from Equations (4.6) that if the delay-vector at time $m-1$ is known to us, we can predict the current measurement $y(m)$ by approximating the unknown function $\mu$. The predictive technique depends on the idea that when the system of $\mathbf{y}_d(m)$ is equivalent to the system of $\mathbf{x}(m)$, we can use the delay-vector $\mathbf{y}_d(m-1)$ to predict the current measurement $y(m)$. To be able to do that, we need to approximate the unknown function $\mu$ such that,

$$\hat{y}(m) = \hat{\mu}(\mathbf{y}_d(m-1)). \tag{4.7}$$

To approximate $\mu$ we will use a neural network with a Tapped Delay Line (TDL) connected to its input. The TDL is used to produce the delay-vector $\mathbf{y}_d(m-1)$ from the current measurement $y(m)$, as seen in Figure 4.6. Each tap of the TDL is a delayed version of $y(m)$ and the total number of taps is *d*. The network is trained to predict $y(m)$ when it is presented with *d* previous measurements of $y(m)$ (coordinates of $\mathbf{y}_d(m-1)$). The resulting prediction error is recorded as a function of *d*. As the predictor order *d* increases, the prediction error will decrease. However, after a certain point, further increase of *d* results in only a very small decrease in the error. The minimum dimension where the error

does not improve any further is the minimum embedding dimension $d_L$. At this point, the approximation of $\mu$ is accurate and the systems of $\mathbf{y}_d(m)$ and $\mathbf{x}(m)$ are equivalent to each other. Figures 4.5 and 4.6 show the block diagram of the function $\mu$ approximation and the neural network model used to create the approximation.

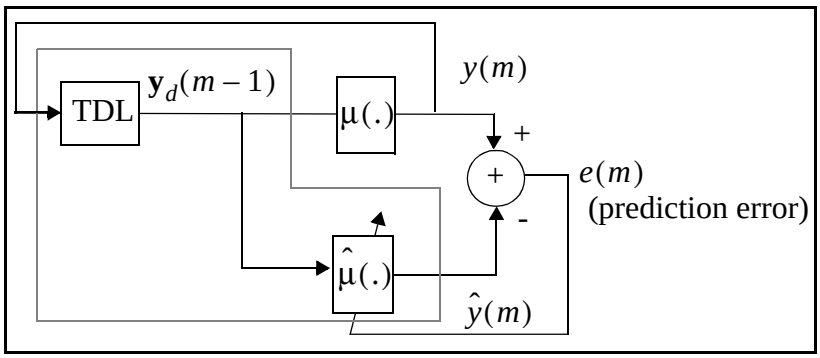



Figure 4.5 The function $\mu$ approximation in the predictive technique

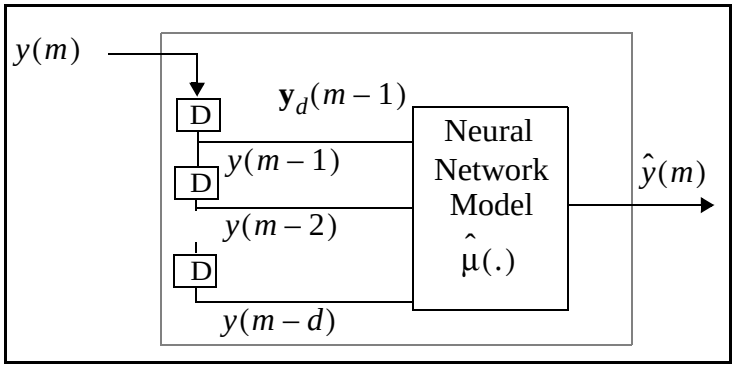



Figure 4.6 A neural network model with a TDL used to approximate the function $\mu$, D is a one step delay-time

## 4.4 Dynamic equivalence

The goal of modeling by embedding is to find a chaotic model that is equivalent to the original unknown system. (The notion of equivalent chaotic systems will be covered in detail in later chapters.) To understand this idea, we know that in the original system the hidden state evolution is governed by the map $\boldsymbol{f}$

$$\mathbf{x}(m+1) = \boldsymbol{f}(\mathbf{x}(m)), \tag{4.8}$$

(see Equations (4.1)), while all we can see from the original system is a set of measurements which are governed by the map *h*:

$$y(m) = h(\mathbf{x}(m)), \tag{4.9}$$

(see Equations (3.19)).

On the other hand, the states in the reconstructed space are the delay-vectors $\mathbf{y}_d(m)$ (see Equations (4.2)), while the output update in this space is governed by the map $\mu$:

$$y(m) = \mu(\mathbf{y}_d(m-1)), \tag{4.10}$$

(see Equations (4.6)). So the delay-vector can be written as

$$\mathbf{y}_d(m) = \left[\mu(\mathbf{y}_d(m-1))\ y(m-1)\ \ldots\ y(m-(d-1))\right]^t, \tag{4.11}$$

($T = 1$). If the reconstructed model given by Equations (4.10) and (4.11) is equivalent to the original model given by Equations (4.8) and (4.9), we can use the evolution of $\mathbf{y}_d(m)$ in place of the evolution of $\mathbf{x}(m)$ to gain an understanding of the system characteristics. In other words, if the two systems are equivalent, we will be able to use $\mathbf{y}_d(m)$ to estimate the original system dimension and parameters. In the coming chapters, we will find the delay-vector dimension *d* and the delay-time *T* which guarantee this dynamic equivalence.

## 4.5 Chapter summary

In this chapter, we provided an introduction to modeling chaotic systems. We introduced two techniques (geometric and predictive) that can be used to estimate the minimum embedding dimension that is required for chaotic systems modeling. We also gave a brief introduction to equivalent chaotic systems. In Chapter 5, we will see the application of both techniques to the chaotic Henon map. In Chapter 6, we will provide complete and detailed algorithms for estimating *d*. We will also discuss practical considerations for implementing the algorithms, including the choice of the delay-time (*T*).

**CHAPTER 5**

# Examples of the minimum embedding dimension estimation

## 5.1 Introduction

In Chapter 3, we introduced modeling of chaotic systems. We used delay-vectors created from measurements to model these systems. We also introduced four methods to estimate the minimum embedding dimension: three geometric methods one predictive method. In this chapter, we will use the four methods to estimate the minimum embedding dimension of the Henon map.

The Henon map is a chaotic system created from a set of two difference equations (see Section 2.3.3). By using the embedding method (see Section 3.5), the theoretical minimum embedding dimension is $d_E = \lceil 2d_c \rceil = \lceil 2\text{x}1.189 \rceil = 3$. But we suspect that the actual minimum embedding dimension is $d_L = 2$ since the system's dynamics are generated from a set of two difference equations. In Sections 5.2 and 5.3, we show the estimated minimum embedding dimension of the Henon map by using the geometric and the predictive techniques, respectively. In Section 5.4, we provide a chapter summary.

## 5.2 The geometric technique

To show how we can estimate $d_L$ by using the geometric technique, we take 15 points from the $X_1$-coordinate of the Henon map to represent the measurements from this

system. That means the measurements $y(m) = x_1(m)$ where $m = 1, 2, \ldots, 15$. In the next step, we use $y(m)$ to construct $\mathbf{y}_d(m) = \left[y(m)\ y(m-1)\ \ldots\ y(m-(d-1))\right]^t$ which is the delay-vector (where $T = 1$).

We said in Chapter 4 that the idea of estimating $d_L$ by using the geometric technique depends on detecting the existence of the projection artifacts on the system's attractor. We can do that by using any of the three methods that we mentioned before: the CND, the CDD, or the CDT method. We need first to compute the distance between the reference vector; $\mathbf{y}_d(m)$ and every other vector in the space $\Re^d$. The nearest neighbor of $\mathbf{y}_d(m)$ is the vector with the shortest distance.

### 5.2.1 Using the change of neighbors with dimension method (CND)

Before we get into the details of the CND method, we need to introduce some notation. First, the nearest neighbor of $\mathbf{y}_d(m)$ will be denoted $\overset{\frown}{\mathbf{y}}{}^1_d(m)$. The $k^{th}$ nearest neighbor of $\mathbf{y}_d(m)$ will be denoted $\overset{\frown}{\mathbf{y}}{}^k_d(m)$. (This means there are $k-1$ vectors that are closer to $\mathbf{y}_d(m)$.) We will indicate the time index of the $k^{th}$ nearest neighbor as

$$i_d^k(m) = index[\overset{\frown}{\mathbf{y}}{}^k_d(m)]. \tag{5.1}$$

For example, if the nearest neighbor of $\mathbf{y}_d(5)$ is $\overset{\frown}{\mathbf{y}}{}^1_d(5) = \mathbf{y}_d(9)$, then the nearest neighbor index is $i_d^1(5) = index[\overset{\frown}{\mathbf{y}}{}^1_d(5)] = index[\mathbf{y}_d(9)] = 9$.

To estimate $d_L$ by using the CND method, we need to check if the nearest neighbor of $\mathbf{y}_d(m)$ will remain a neighbor as the dimension $d$ grows to $d+1$. In other words, we need to check if the nearest neighbor of $\mathbf{y}_d(m)$ will appear as the first, second, …, or $w^{th}$ neighbor of $\mathbf{y}_{d+1}(m)$. To do that, we need to build the matrix $\mathbf{Q}_d$ which has the elements $q_d(i,j) = \|\mathbf{y}_d(i) - \mathbf{y}_d(j)\|$ where $i, j = 1, 2, \ldots, M$ and $i \neq j$. If $i = j$, we label $q_d(i,j)$ as

not a number (NaN) since we are not interested in the distance between a vector and itself. We need also to find the vector $\mathbf{i}_d^1$ whose $m^{th}$ element $i_d^1(m)$ is the time index of the nearest neighbor of $\mathbf{y}_d(m)$. Further, we need to compute the $1^{st}$ through the $w^{th}$ neighbors of each $\mathbf{y}_{d+1}(m)$ and save their indices in the $m^{th}$ row of the matrix $\mathbf{I}_{d+1}$. In other words, $\mathbf{Q}_d$, $\mathbf{i}_d^k$ and $\mathbf{I}_d$ are defined as follows:

$$\mathbf{Q}_d = \begin{bmatrix} q_d(1,1) & q_d(1,2) & \dots & q_d(1,M) \\ q_d(2,1) & q_d(2,2) & \dots & q_d(2,M) \\ \vdots & & & \\ q_d(M,1) & q_d(M,2) & \dots & q_d(M,M) \end{bmatrix}, \tag{5.2}$$

$$\mathbf{Q}_d = \begin{bmatrix} \text{NaN} & \|\mathbf{y}_d(1)-\mathbf{y}_d(2)\| & \dots & \|\mathbf{y}_d(1)-\mathbf{y}_d(M)\| \\ \|\mathbf{y}_d(2)-\mathbf{y}_d(1)\| & \text{NaN} & \dots & \|\mathbf{y}_d(2)-\mathbf{y}_d(M)\| \\ \vdots & & & \vdots \\ \|\mathbf{y}_d(M)-\mathbf{y}_d(1)\| & \|\mathbf{y}_d(M)-\mathbf{y}_d(2)\| & \dots & \text{NaN} \end{bmatrix}, \tag{5.3}$$

$$\mathbf{i}_d^k = \begin{bmatrix} i_d^k(1) \\ i_d^k(2) \\ \vdots \\ i_d^k(M) \end{bmatrix} = \begin{bmatrix} index[\,\overset{\frown}{\mathbf{y}}{}_d^k(1)] \\ index[\,\overset{\frown}{\mathbf{y}}{}_d^k(2)] \\ \vdots \\ index[\,\overset{\frown}{\mathbf{y}}{}_d^k(M)] \end{bmatrix} \text{ where } k = 1, 2, \dots, w, \tag{5.4}$$

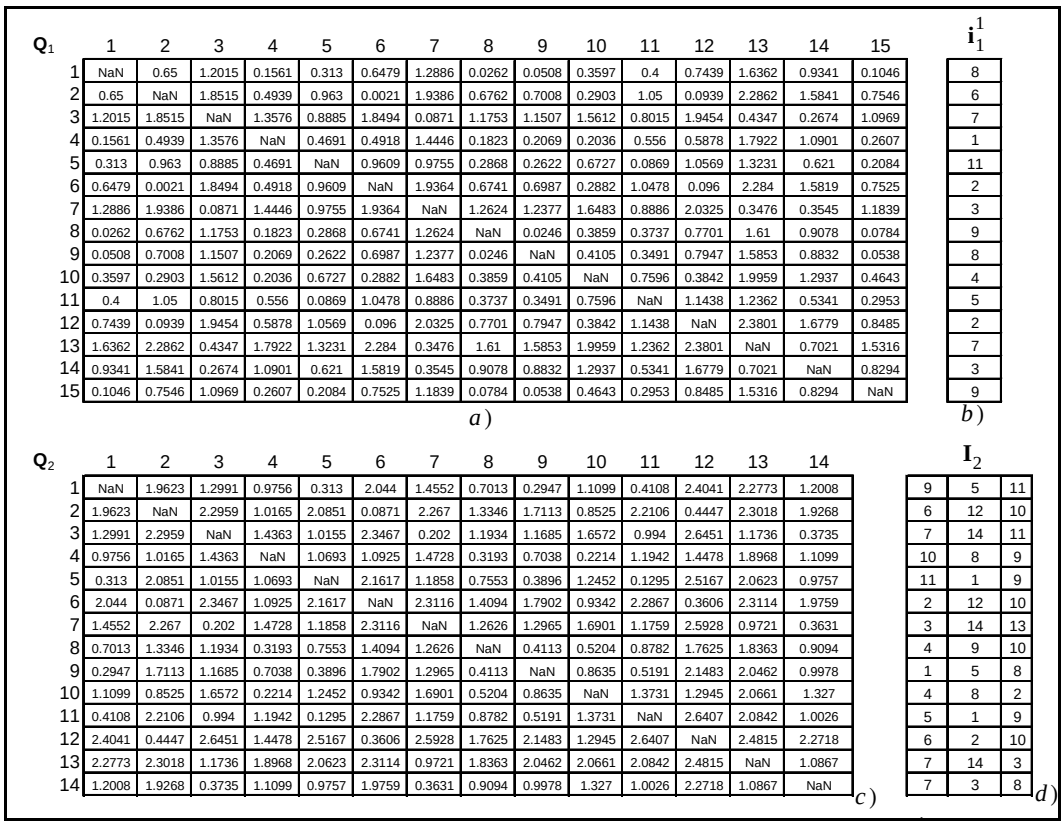
$$\text{and } \mathbf{I}_d = \begin{bmatrix} \mathbf{i}_d^1 & \mathbf{i}_d^2 & \dots & \mathbf{i}_d^w \end{bmatrix} = \begin{bmatrix} i_d^1(1) & i_d^2(1) & \dots & i_d^w(1) \\ i_d^1(2) & i_d^2(2) & \dots & i_d^w(2) \\ \vdots & & & \vdots \\ i_d^1(M) & i_d^2(M) & \dots & i_d^w(M) \end{bmatrix}. \tag{5.5}$$

Each row of the matrix $\mathbf{I}_d$ will be labeled by ${}^m\mathbf{i}_d = \begin{bmatrix} i_d^1(m) & i_d^2(m) & \dots & i_d^w(m) \end{bmatrix}$. So, the ma-

trix $\mathbf{I}_d$ can be written as: $\mathbf{I}_d = \begin{bmatrix} {}^{1}\mathbf{i}_d \\ {}^{2}\mathbf{i}_d \\ \vdots \\ {}^{M}\mathbf{i}_d \end{bmatrix}$.

In Tables 5.1*a* and 5.1*c* and Tables 5.2*a* and 5.2*c*, we can see $\mathbf{Q}_d$ listed for $d = 1$ through 4. In Tables 5.1*d*, 5.2*b* and 5.2*d*, we can see $\mathbf{I}_2$ through $\mathbf{I}_4$ showing the indices of the three neighbors ($w = 3$) of each $\mathbf{y}_{d+1}(m)$. Tables 5.3*a*, 5.3*b*, and 5.3*c* summarize the neighbors time indices found for $d = 1$ through 4. For example, in Table 5.3*a* we can see that the nearest neighbor of $y_1(1)$ is $y_1(8)$ and it fails to appear as the first, second, or third neighbor of $\mathbf{y}_2(1)$, so we label $y_1(8)$ as a FNN. On the other hand, we can see that the nearest neighbor of $y_1(8)$ is $y_1(9)$ and when the dimension increases to 2, it appears as the second neighbor of $\mathbf{y}_2(8)$, so we label $y_1(9)$ as a true neighbor. As a conclusion, we can see in Table 5.3*d* that the total number of FNNs drops from 2 at $d = 1$ to 0 at $d = 2$ and remains 0 at $d = 3$. That means the minimum embedding dimension is $d_L = 2$, where there are no projection artifacts left on the attractor of the system.

Notice that in this method we could have chosen $w = 1$, so that only the nearest neighbor is considered. We have found through experimentation that using *w* greater than 1 provides a more robust algorithm. **This is a new result**. In Chapter 6, we will provide more detail on the practical implementation aspects of the various algorithms.



$\mathbf{Q}_1$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 0.65 | 1.2015 | 0.1561 | 0.313 | 0.6479 | 1.2886 | 0.0262 | 0.0508 | 0.3597 | 0.4 | 0.7439 | 1.6362 | 0.9341 | 0.1046 |
| 2 | 0.65 | NaN | 1.8515 | 0.4939 | 0.963 | 0.0021 | 1.9386 | 0.6762 | 0.7008 | 0.2903 | 1.05 | 0.0939 | 2.2862 | 1.5841 | 0.7546 |
| 3 | 1.2015 | 1.8515 | NaN | 1.3576 | 0.8885 | 1.8494 | 0.0871 | 1.1753 | 1.1507 | 1.5612 | 0.8015 | 1.9454 | 0.4347 | 0.2674 | 1.0969 |
| 4 | 0.1561 | 0.4939 | 1.3576 | NaN | 0.4691 | 0.4918 | 1.4446 | 0.1823 | 0.2069 | 0.2036 | 0.556 | 0.5878 | 1.7922 | 1.0901 | 0.2607 |
| 5 | 0.313 | 0.963 | 0.8885 | 0.4691 | NaN | 0.9609 | 0.9755 | 0.2868 | 0.2622 | 0.6727 | 0.0869 | 1.0569 | 1.3231 | 0.621 | 0.2084 |
| 6 | 0.6479 | 0.0021 | 1.8494 | 0.4918 | 0.9609 | NaN | 1.9364 | 0.6741 | 0.6987 | 0.2882 | 1.0478 | 0.096 | 2.284 | 1.5819 | 0.7525 |
| 7 | 1.2886 | 1.9386 | 0.0871 | 1.4446 | 0.9755 | 1.9364 | NaN | 1.2624 | 1.2377 | 1.6483 | 0.8886 | 2.0325 | 0.3476 | 0.3545 | 1.1839 |
| 8 | 0.0262 | 0.6762 | 1.1753 | 0.1823 | 0.2868 | 0.6741 | 1.2624 | NaN | 0.0246 | 0.3859 | 0.3737 | 0.7701 | 1.61 | 0.9078 | 0.0784 |
| 9 | 0.0508 | 0.7008 | 1.1507 | 0.2069 | 0.2622 | 0.6987 | 1.2377 | 0.0246 | NaN | 0.4105 | 0.3491 | 0.7947 | 1.5853 | 0.8832 | 0.0538 |
| 10 | 0.3597 | 0.2903 | 1.5612 | 0.2036 | 0.6727 | 0.2882 | 1.6483 | 0.3859 | 0.4105 | NaN | 0.7596 | 0.3842 | 1.9959 | 1.2937 | 0.4643 |
| 11 | 0.4 | 1.05 | 0.8015 | 0.556 | 0.0869 | 1.0478 | 0.8886 | 0.3737 | 0.3491 | 0.7596 | NaN | 1.1438 | 1.2362 | 0.5341 | 0.2953 |
| 12 | 0.7439 | 0.0939 | 1.9454 | 0.5878 | 1.0569 | 0.096 | 2.0325 | 0.7701 | 0.7947 | 0.3842 | 1.1438 | NaN | 2.3801 | 1.6779 | 0.8485 |
| 13 | 1.6362 | 2.2862 | 0.4347 | 1.7922 | 1.3231 | 2.284 | 0.3476 | 1.61 | 1.5853 | 1.9959 | 1.2362 | 2.3801 | NaN | 0.7021 | 1.5316 |
| 14 | 0.9341 | 1.5841 | 0.2674 | 1.0901 | 0.621 | 1.5819 | 0.3545 | 0.9078 | 0.8832 | 1.2937 | 0.5341 | 1.6779 | 0.7021 | NaN | 0.8294 |
| 15 | 0.1046 | 0.7546 | 1.0969 | 0.2607 | 0.2084 | 0.7525 | 1.1839 | 0.0784 | 0.0538 | 0.4643 | 0.2953 | 0.8485 | 1.5316 | 0.8294 | NaN |

*a*)

$\mathbf{i}_1^1$

| |
|---|
| 8 |
| 6 |
| 7 |
| 1 |
| 11 |
| 2 |
| 3 |
| 9 |
| 8 |
| 4 |
| 5 |
| 2 |
| 7 |
| 3 |
| 9 |

*b*)

$\mathbf{Q}_2$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 1.9623 | 1.2991 | 0.9756 | 0.313 | 2.044 | 1.4552 | 0.7013 | 0.2947 | 1.1099 | 0.4108 | 2.4041 | 2.2773 | 1.2008 |
| 2 | 1.9623 | NaN | 2.2959 | 1.0165 | 2.0851 | 0.0871 | 2.267 | 1.3346 | 1.7113 | 0.8525 | 2.2106 | 0.4447 | 2.3018 | 1.9268 |
| 3 | 1.2991 | 2.2959 | NaN | 1.4363 | 1.0155 | 2.3467 | 0.202 | 1.1934 | 1.1685 | 1.6572 | 0.994 | 2.6451 | 1.1736 | 0.3735 |
| 4 | 0.9756 | 1.0165 | 1.4363 | NaN | 1.0693 | 1.0925 | 1.4728 | 0.3193 | 0.7038 | 0.2214 | 1.1942 | 1.4478 | 1.8968 | 1.1099 |
| 5 | 0.313 | 2.0851 | 1.0155 | 1.0693 | NaN | 2.1617 | 1.1858 | 0.7553 | 0.3896 | 1.2452 | 0.1295 | 2.5167 | 2.0623 | 0.9757 |
| 6 | 2.044 | 0.0871 | 2.3467 | 1.0925 | 2.1617 | NaN | 2.3116 | 1.4094 | 1.7902 | 0.9342 | 2.2867 | 0.3606 | 2.3114 | 1.9759 |
| 7 | 1.4552 | 2.267 | 0.202 | 1.4728 | 1.1858 | 2.3116 | NaN | 1.2626 | 1.2965 | 1.6901 | 1.1759 | 2.5928 | 0.9721 | 0.3631 |
| 8 | 0.7013 | 1.3346 | 1.1934 | 0.3193 | 0.7553 | 1.4094 | 1.2626 | NaN | 0.4113 | 0.5204 | 0.8782 | 1.7625 | 1.8363 | 0.9094 |
| 9 | 0.2947 | 1.7113 | 1.1685 | 0.7038 | 0.3896 | 1.7902 | 1.2965 | 0.4113 | NaN | 0.8635 | 0.5191 | 2.1483 | 2.0462 | 0.9978 |
| 10 | 1.1099 | 0.8525 | 1.6572 | 0.2214 | 1.2452 | 0.9342 | 1.6901 | 0.5204 | 0.8635 | NaN | 1.3731 | 1.2945 | 2.0661 | 1.327 |
| 11 | 0.4108 | 2.2106 | 0.994 | 1.1942 | 0.1295 | 2.2867 | 1.1759 | 0.8782 | 0.5191 | 1.3731 | NaN | 2.6407 | 2.0842 | 1.0026 |
| 12 | 2.4041 | 0.4447 | 2.6451 | 1.4478 | 2.5167 | 0.3606 | 2.5928 | 1.7625 | 2.1483 | 1.2945 | 2.6407 | NaN | 2.4815 | 2.2718 |
| 13 | 2.2773 | 2.3018 | 1.1736 | 1.8968 | 2.0623 | 2.3114 | 0.9721 | 1.8363 | 2.0462 | 2.0661 | 2.0842 | 2.4815 | NaN | 1.0867 |
| 14 | 1.2008 | 1.9268 | 0.3735 | 1.1099 | 0.9757 | 1.9759 | 0.3631 | 0.9094 | 0.9978 | 1.327 | 1.0026 | 2.2718 | 1.0867 | NaN |

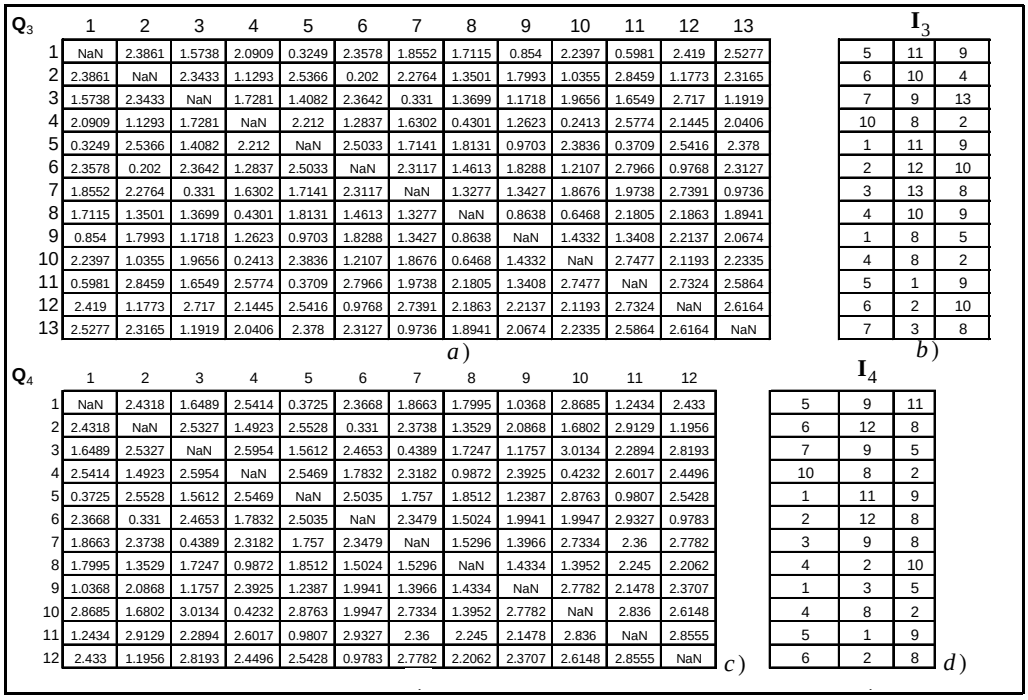
*c*)

$\mathbf{I}_2$

| | | |
|---|---|---|
| 9 | 5 | 11 |
| 6 | 12 | 10 |
| 7 | 14 | 11 |
| 10 | 8 | 9 |
| 11 | 1 | 9 |
| 2 | 12 | 10 |
| 3 | 14 | 13 |
| 4 | 9 | 10 |
| 1 | 5 | 8 |
| 4 | 8 | 2 |
| 5 | 1 | 9 |
| 6 | 2 | 10 |
| 7 | 14 | 3 |
| 7 | 3 | 8 |

*d*)

Table 5.1 The CND method (1/3)



$\mathbf{Q}_3$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 2.3861 | 1.5738 | 2.0909 | 0.3249 | 2.3578 | 1.8552 | 1.7115 | 0.854 | 2.2397 | 0.5981 | 2.419 | 2.5277 |
| 2 | 2.3861 | NaN | 2.3433 | 1.1293 | 2.5366 | 0.202 | 2.2764 | 1.3501 | 1.7993 | 1.0355 | 2.8459 | 1.1773 | 2.3165 |
| 3 | 1.5738 | 2.3433 | NaN | 1.7281 | 1.4082 | 2.3642 | 0.331 | 1.3699 | 1.1718 | 1.9656 | 1.6549 | 2.717 | 1.1919 |
| 4 | 2.0909 | 1.1293 | 1.7281 | NaN | 2.212 | 1.2837 | 1.6302 | 0.4301 | 1.2623 | 0.2413 | 2.5774 | 2.1445 | 2.0406 |
| 5 | 0.3249 | 2.5366 | 1.4082 | 2.212 | NaN | 2.5033 | 1.7141 | 1.8131 | 0.9703 | 2.3836 | 0.3709 | 2.5416 | 2.378 |
| 6 | 2.3578 | 0.202 | 2.3642 | 1.2837 | 2.5033 | NaN | 2.3117 | 1.4613 | 1.8288 | 1.2107 | 2.7966 | 0.9768 | 2.3127 |
| 7 | 1.8552 | 2.2764 | 0.331 | 1.6302 | 1.7141 | 2.3117 | NaN | 1.3277 | 1.3427 | 1.8676 | 1.9738 | 2.7391 | 0.9736 |
| 8 | 1.7115 | 1.3501 | 1.3699 | 0.4301 | 1.8131 | 1.4613 | 1.3277 | NaN | 0.8638 | 0.6468 | 2.1805 | 2.1863 | 1.8941 |
| 9 | 0.854 | 1.7993 | 1.1718 | 1.2623 | 0.9703 | 1.8288 | 1.3427 | 0.8638 | NaN | 1.4332 | 1.3408 | 2.2137 | 2.0674 |
| 10 | 2.2397 | 1.0355 | 1.9656 | 0.2413 | 2.3836 | 1.2107 | 1.8676 | 0.6468 | 1.4332 | NaN | 2.7477 | 2.1193 | 2.2335 |
| 11 | 0.5981 | 2.8459 | 1.6549 | 2.5774 | 0.3709 | 2.7966 | 1.9738 | 2.1805 | 1.3408 | 2.7477 | NaN | 2.7324 | 2.5864 |
| 12 | 2.419 | 1.1773 | 2.717 | 2.1445 | 2.5416 | 0.9768 | 2.7391 | 2.1863 | 2.2137 | 2.1193 | 2.7324 | NaN | 2.6164 |
| 13 | 2.5277 | 2.3165 | 1.1919 | 2.0406 | 2.378 | 2.3127 | 0.9736 | 1.8941 | 2.0674 | 2.2335 | 2.5864 | 2.6164 | NaN |

*a*)

$\mathbf{I}_3$

| | | |
|---|---|---|
| 5 | 11 | 9 |
| 6 | 10 | 4 |
| 7 | 9 | 13 |
| 10 | 8 | 2 |
| 1 | 11 | 9 |
| 2 | 12 | 10 |
| 3 | 13 | 8 |
| 4 | 10 | 9 |
| 1 | 8 | 5 |
| 4 | 8 | 2 |
| 5 | 1 | 9 |
| 6 | 2 | 10 |
| 7 | 3 | 8 |

*b*)

$\mathbf{Q}_4$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 2.4318 | 1.6489 | 2.5414 | 0.3725 | 2.3668 | 1.8663 | 1.7995 | 1.0368 | 2.8685 | 1.2434 | 2.433 |
| 2 | 2.4318 | NaN | 2.5327 | 1.4923 | 2.5528 | 0.331 | 2.3738 | 1.3529 | 2.0868 | 1.6802 | 2.9129 | 1.1956 |
| 3 | 1.6489 | 2.5327 | NaN | 2.5954 | 1.5612 | 2.4653 | 0.4389 | 1.7247 | 1.1757 | 3.0134 | 2.2894 | 2.8193 |
| 4 | 2.5414 | 1.4923 | 2.5954 | NaN | 2.5469 | 1.7832 | 2.3182 | 0.9872 | 2.3925 | 0.4232 | 2.6017 | 2.4496 |
| 5 | 0.3725 | 2.5528 | 1.5612 | 2.5469 | NaN | 2.5035 | 1.757 | 1.8512 | 1.2387 | 2.8763 | 0.9807 | 2.5428 |
| 6 | 2.3668 | 0.331 | 2.4653 | 1.7832 | 2.5035 | NaN | 2.3479 | 1.5024 | 1.9941 | 1.9947 | 2.9327 | 0.9783 |
| 7 | 1.8663 | 2.3738 | 0.4389 | 2.3182 | 1.757 | 2.3479 | NaN | 1.5296 | 1.3966 | 2.7334 | 2.36 | 2.7782 |
| 8 | 1.7995 | 1.3529 | 1.7247 | 0.9872 | 1.8512 | 1.5024 | 1.5296 | NaN | 1.4334 | 1.3952 | 2.245 | 2.2062 |
| 9 | 1.0368 | 2.0868 | 1.1757 | 2.3925 | 1.2387 | 1.9941 | 1.3966 | 1.4334 | NaN | 2.7782 | 2.1478 | 2.3707 |
| 10 | 2.8685 | 1.6802 | 3.0134 | 0.4232 | 2.8763 | 1.9947 | 2.7334 | 1.3952 | 2.7782 | NaN | 2.836 | 2.6148 |
| 11 | 1.2434 | 2.9129 | 2.2894 | 2.6017 | 0.9807 | 2.9327 | 2.36 | 2.245 | 2.1478 | 2.836 | NaN | 2.8555 |
| 12 | 2.433 | 1.1956 | 2.8193 | 2.4496 | 2.5428 | 0.9783 | 2.7782 | 2.2062 | 2.3707 | 2.6148 | 2.8555 | NaN |

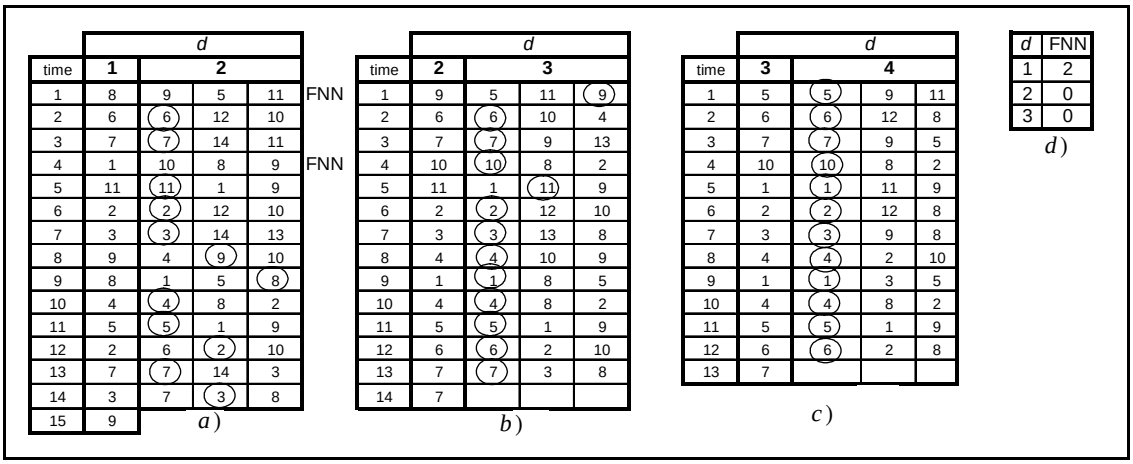
*c*)

$\mathbf{I}_4$

| | | |
|---|---|---|
| 5 | 9 | 11 |
| 6 | 12 | 8 |
| 7 | 9 | 5 |
| 10 | 8 | 2 |
| 1 | 11 | 9 |
| 2 | 12 | 8 |
| 3 | 9 | 8 |
| 4 | 2 | 10 |
| 1 | 3 | 5 |
| 4 | 8 | 2 |
| 5 | 1 | 9 |
| 6 | 2 | 8 |

*d*)

Table 5.2 The CND method (2/3)



| time | d = 1 | d = 2 | | | |
|---|---|---|---|---|---|
| 1 | 8 | 9 | 5 | 11 | FNN |
| 2 | 6 | 6 | 12 | 10 | |
| 3 | 7 | 7 | 14 | 11 | |
| 4 | 1 | 10 | 8 | 9 | FNN |
| 5 | 11 | 11 | 1 | 9 | |
| 6 | 2 | 2 | 12 | 10 | |
| 7 | 3 | 3 | 14 | 13 | |
| 8 | 9 | 4 | 9 | 10 | |
| 9 | 8 | 1 | 5 | 8 | |
| 10 | 4 | 4 | 8 | 2 | |
| 11 | 5 | 5 | 1 | 9 | |
| 12 | 2 | 6 | 2 | 10 | |
| 13 | 7 | 7 | 14 | 3 | |
| 14 | 3 | 7 | 3 | 8 | |
| 15 | 9 | | | | |

*a*)

| time | d = 2 | d = 3 | | | |
|---|---|---|---|---|---|
| 1 | 9 | 5 | 11 | 9 | |
| 2 | 6 | 6 | 10 | 4 | |
| 3 | 7 | 7 | 9 | 13 | |
| 4 | 10 | 10 | 8 | 2 | |
| 5 | 11 | 1 | 11 | 9 | |
| 6 | 2 | 2 | 12 | 10 | |
| 7 | 3 | 3 | 13 | 8 | |
| 8 | 4 | 4 | 10 | 9 | |
| 9 | 1 | 1 | 8 | 5 | |
| 10 | 4 | 4 | 8 | 2 | |
| 11 | 5 | 5 | 1 | 9 | |
| 12 | 6 | 6 | 2 | 10 | |
| 13 | 7 | 7 | 3 | 8 | |
| 14 | 7 | | | | |

*b*)

| time | d = 3 | d = 4 | | | |
|---|---|---|---|---|---|
| 1 | 5 | 5 | 9 | 11 | |
| 2 | 6 | 6 | 12 | 8 | |
| 3 | 7 | 7 | 9 | 5 | |
| 4 | 10 | 10 | 8 | 2 | |
| 5 | 1 | 1 | 11 | 9 | |
| 6 | 2 | 2 | 12 | 8 | |
| 7 | 3 | 3 | 9 | 8 | |
| 8 | 4 | 4 | 2 | 10 | |
| 9 | 1 | 1 | 3 | 5 | |
| 10 | 4 | 4 | 8 | 2 | |
| 11 | 5 | 5 | 1 | 9 | |
| 12 | 6 | 6 | 2 | 8 | |
| 13 | 7 | | | | |

*c*)

| d | FNN |
|---|---|
| 1 | 2 |
| 2 | 0 |
| 3 | 0 |

*d*)

Table 5.3 The CND method (3/3)

### 5.2.2 Using the change of distance with dimension method (CDD)

As we said in Chapter 4, the CDD method can also be used to estimate $d_L$. This method depends on the idea that false nearest neighbor distances increase significantly as the dimension of the space increases.

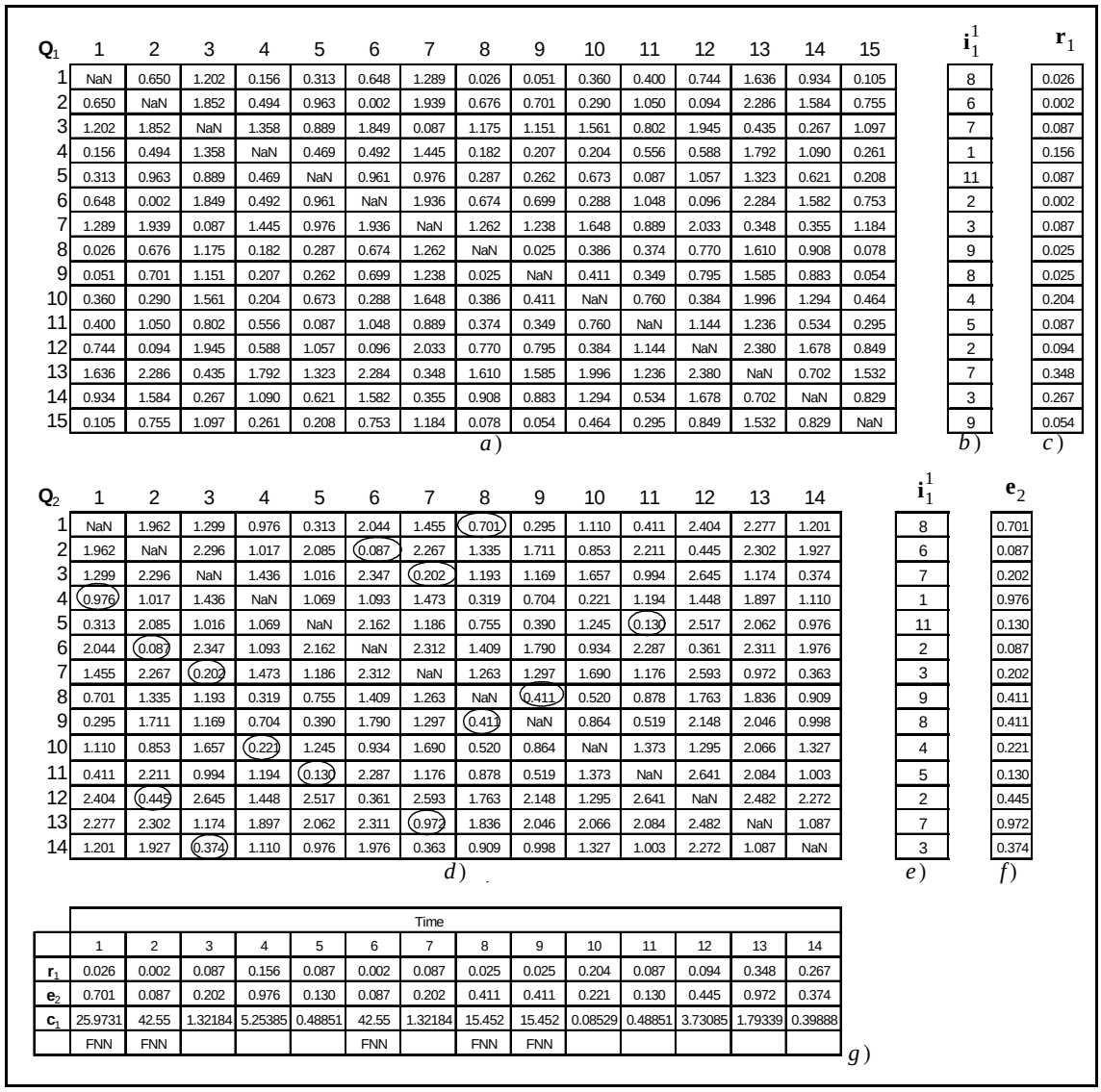
To estimate $d_L$ of the Henon map using the CDD method, we need to compute the matrix $\mathbf{Q}_d$ and find the vector $\mathbf{i}_d^1$ as in the CND method. In addition, we need to find the vector $\mathbf{r}_d$ whose element $r_d(m)$ is the distance between $\mathbf{y}_d(m)$ and its nearest neighbor $\widehat{\mathbf{y}}_d^1(m) = \mathbf{y}_d(i_d^1(m)) : r_d(m) = \left\| \mathbf{y}_d(m) - \widehat{\mathbf{y}}_d^1(m) \right\|$. We also need to find out how much the distances to the nearest neighbors grow as the dimension increases. For these new distances we define the vector $\mathbf{e}_{d+1}$ whose elements $e_{d+1}(m)$ represent the distance between $\mathbf{y}_{d+1}(m)$ and $\mathbf{y}_{d+1}(i_d^1(m))$. (Notice that $\mathbf{y}_{d+1}(i_d^1(m))$ is not the same as $\widehat{\mathbf{y}}_{d+1}^1(m)$. It is $\widehat{\mathbf{y}}_d^1(m)$ with one component added.)

For example, when $d = 1$ in Table 5.4*a*, the nearest neighbor of $y_1(1)$ is $y_1(8)$ and the distance between them is $r_1(1) = 0.026$. At $d = 2$ in Table 5.4*d*, the distance between $\mathbf{y}_2(1)$ and $\mathbf{y}_2(8)$ is $e_2(1) = 0.701$. If $\mathbf{y}_d(i_d^1(m))$ is a true nearest neighbor of $\mathbf{y}_d(m)$, $\mathbf{y}_{d+1}(i_d^1(m))$ will be close to $\mathbf{y}_{d+1}(m)$. To apply this idea, we need to see how

much the distance between the nearest neighbors grows as $d$ increases to $d + 1$. We can do that by forming the vector $\mathbf{c}_d$ which has the elements

$$c_d(m) = \frac{e_{d+1}(m) - r_d(m)}{r_d(m)}. \tag{5.6}$$

If $c_d(m) > \rho$ where $\rho$ is some predefined threshold, we label the nearest neighbor of $\mathbf{y}_d(m)$ as a false nearest neighbor (FNN). For instance, let $\rho = 10$. We can see from our example that $c_1(1) = 25.9731$ as seen at the second column of Table 5.4*g*. Since $c_1(1) > \rho = 10$, we label $y_1(8)$ as a FNN. The results in Tables 5.4*g*, 5.5*g*, and 5.6*g* show $\mathbf{c}_d$ for $d = 1$ through 3. In Table 5.6*h*, we summarize the results of the previous tables. In it, we can see that at $d = 1$, the number of FNNs is 5, while at $d = 2$, the number of FNNs drops to 0 and remains 0 at $d = 3$. Therefore, the minimum embedding dimension is $d_L = 2$.



| $\mathbf{Q}_1$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 0.650 | 1.202 | 0.156 | 0.313 | 0.648 | 1.289 | 0.026 | 0.051 | 0.360 | 0.400 | 0.744 | 1.636 | 0.934 | 0.105 |
| 2 | 0.650 | NaN | 1.852 | 0.494 | 0.963 | 0.002 | 1.939 | 0.676 | 0.701 | 0.290 | 1.050 | 0.094 | 2.286 | 1.584 | 0.755 |
| 3 | 1.202 | 1.852 | NaN | 1.358 | 0.889 | 1.849 | 0.087 | 1.175 | 1.151 | 1.561 | 0.802 | 1.945 | 0.435 | 0.267 | 1.097 |
| 4 | 0.156 | 0.494 | 1.358 | NaN | 0.469 | 0.492 | 1.445 | 0.182 | 0.207 | 0.204 | 0.556 | 0.588 | 1.792 | 1.090 | 0.261 |
| 5 | 0.313 | 0.963 | 0.889 | 0.469 | NaN | 0.961 | 0.976 | 0.287 | 0.262 | 0.673 | 0.087 | 1.057 | 1.323 | 0.621 | 0.208 |
| 6 | 0.648 | 0.002 | 1.849 | 0.492 | 0.961 | NaN | 1.936 | 0.674 | 0.699 | 0.288 | 1.048 | 0.096 | 2.284 | 1.582 | 0.753 |
| 7 | 1.289 | 1.939 | 0.087 | 1.445 | 0.976 | 1.936 | NaN | 1.262 | 1.238 | 1.648 | 0.889 | 2.033 | 0.348 | 0.355 | 1.184 |
| 8 | 0.026 | 0.676 | 1.175 | 0.182 | 0.287 | 0.674 | 1.262 | NaN | 0.025 | 0.386 | 0.374 | 0.770 | 1.610 | 0.908 | 0.078 |
| 9 | 0.051 | 0.701 | 1.151 | 0.207 | 0.262 | 0.699 | 1.238 | 0.025 | NaN | 0.411 | 0.349 | 0.795 | 1.585 | 0.883 | 0.054 |
| 10 | 0.360 | 0.290 | 1.561 | 0.204 | 0.673 | 0.288 | 1.648 | 0.386 | 0.411 | NaN | 0.760 | 0.384 | 1.996 | 1.294 | 0.464 |
| 11 | 0.400 | 1.050 | 0.802 | 0.556 | 0.087 | 1.048 | 0.889 | 0.374 | 0.349 | 0.760 | NaN | 1.144 | 1.236 | 0.534 | 0.295 |
| 12 | 0.744 | 0.094 | 1.945 | 0.588 | 1.057 | 0.096 | 2.033 | 0.770 | 0.795 | 0.384 | 1.144 | NaN | 2.380 | 1.678 | 0.849 |
| 13 | 1.636 | 2.286 | 0.435 | 1.792 | 1.323 | 2.284 | 0.348 | 1.610 | 1.585 | 1.996 | 1.236 | 2.380 | NaN | 0.702 | 1.532 |
| 14 | 0.934 | 1.584 | 0.267 | 1.090 | 0.621 | 1.582 | 0.355 | 0.908 | 0.883 | 1.294 | 0.534 | 1.678 | 0.702 | NaN | 0.829 |
| 15 | 0.105 | 0.755 | 1.097 | 0.261 | 0.208 | 0.753 | 1.184 | 0.078 | 0.054 | 0.464 | 0.295 | 0.849 | 1.532 | 0.829 | NaN |

*a*)

| $\mathbf{i}_1^1$ |
|---|
| 8 |
| 6 |
| 7 |
| 1 |
| 11 |
| 2 |
| 3 |
| 9 |
| 8 |
| 4 |
| 5 |
| 2 |
| 7 |
| 3 |
| 9 |

*b*)

| $\mathbf{r}_1$ |
|---|
| 0.026 |
| 0.002 |
| 0.087 |
| 0.156 |
| 0.087 |
| 0.002 |
| 0.087 |
| 0.025 |
| 0.025 |
| 0.204 |
| 0.087 |
| 0.094 |
| 0.348 |
| 0.267 |
| 0.054 |

*c*)

| $\mathbf{Q}_2$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 1.962 | 1.299 | 0.976 | 0.313 | 2.044 | 1.455 | 0.701 | 0.295 | 1.110 | 0.411 | 2.404 | 2.277 | 1.201 |
| 2 | 1.962 | NaN | 2.296 | 1.017 | 2.085 | 0.087 | 2.267 | 1.335 | 1.711 | 0.853 | 2.211 | 0.445 | 2.302 | 1.927 |
| 3 | 1.299 | 2.296 | NaN | 1.436 | 1.016 | 2.347 | 0.202 | 1.193 | 1.169 | 1.657 | 0.994 | 2.645 | 1.174 | 0.374 |
| 4 | 0.976 | 1.017 | 1.436 | NaN | 1.069 | 1.093 | 1.473 | 0.319 | 0.704 | 0.221 | 1.194 | 1.448 | 1.897 | 1.110 |
| 5 | 0.313 | 2.085 | 1.016 | 1.069 | NaN | 2.162 | 1.186 | 0.755 | 0.390 | 1.245 | 0.130 | 2.517 | 2.062 | 0.976 |
| 6 | 2.044 | 0.087 | 2.347 | 1.093 | 2.162 | NaN | 2.312 | 1.409 | 1.790 | 0.934 | 2.287 | 0.361 | 2.311 | 1.976 |
| 7 | 1.455 | 2.267 | 0.202 | 1.473 | 1.186 | 2.312 | NaN | 1.263 | 1.297 | 1.690 | 1.176 | 2.593 | 0.972 | 0.363 |
| 8 | 0.701 | 1.335 | 1.193 | 0.319 | 0.755 | 1.409 | 1.263 | NaN | 0.411 | 0.520 | 0.878 | 1.763 | 1.836 | 0.909 |
| 9 | 0.295 | 1.711 | 1.169 | 0.704 | 0.390 | 1.790 | 1.297 | 0.411 | NaN | 0.864 | 0.519 | 2.148 | 2.046 | 0.998 |
| 10 | 1.110 | 0.853 | 1.657 | 0.221 | 1.245 | 0.934 | 1.690 | 0.520 | 0.864 | NaN | 1.373 | 1.295 | 2.066 | 1.327 |
| 11 | 0.411 | 2.211 | 0.994 | 1.194 | 0.130 | 2.287 | 1.176 | 0.878 | 0.519 | 1.373 | NaN | 2.641 | 2.084 | 1.003 |
| 12 | 2.404 | 0.445 | 2.645 | 1.448 | 2.517 | 0.361 | 2.593 | 1.763 | 2.148 | 1.295 | 2.641 | NaN | 2.482 | 2.272 |
| 13 | 2.277 | 2.302 | 1.174 | 1.897 | 2.062 | 2.311 | 0.972 | 1.836 | 2.046 | 2.066 | 2.084 | 2.482 | NaN | 1.087 |
| 14 | 1.201 | 1.927 | 0.374 | 1.110 | 0.976 | 1.976 | 0.363 | 0.909 | 0.998 | 1.327 | 1.003 | 2.272 | 1.087 | NaN |

*d*)

| $\mathbf{i}_1^1$ |
|---|
| 8 |
| 6 |
| 7 |
| 1 |
| 11 |
| 2 |
| 3 |
| 9 |
| 8 |
| 4 |
| 5 |
| 2 |
| 7 |
| 3 |

*e*)

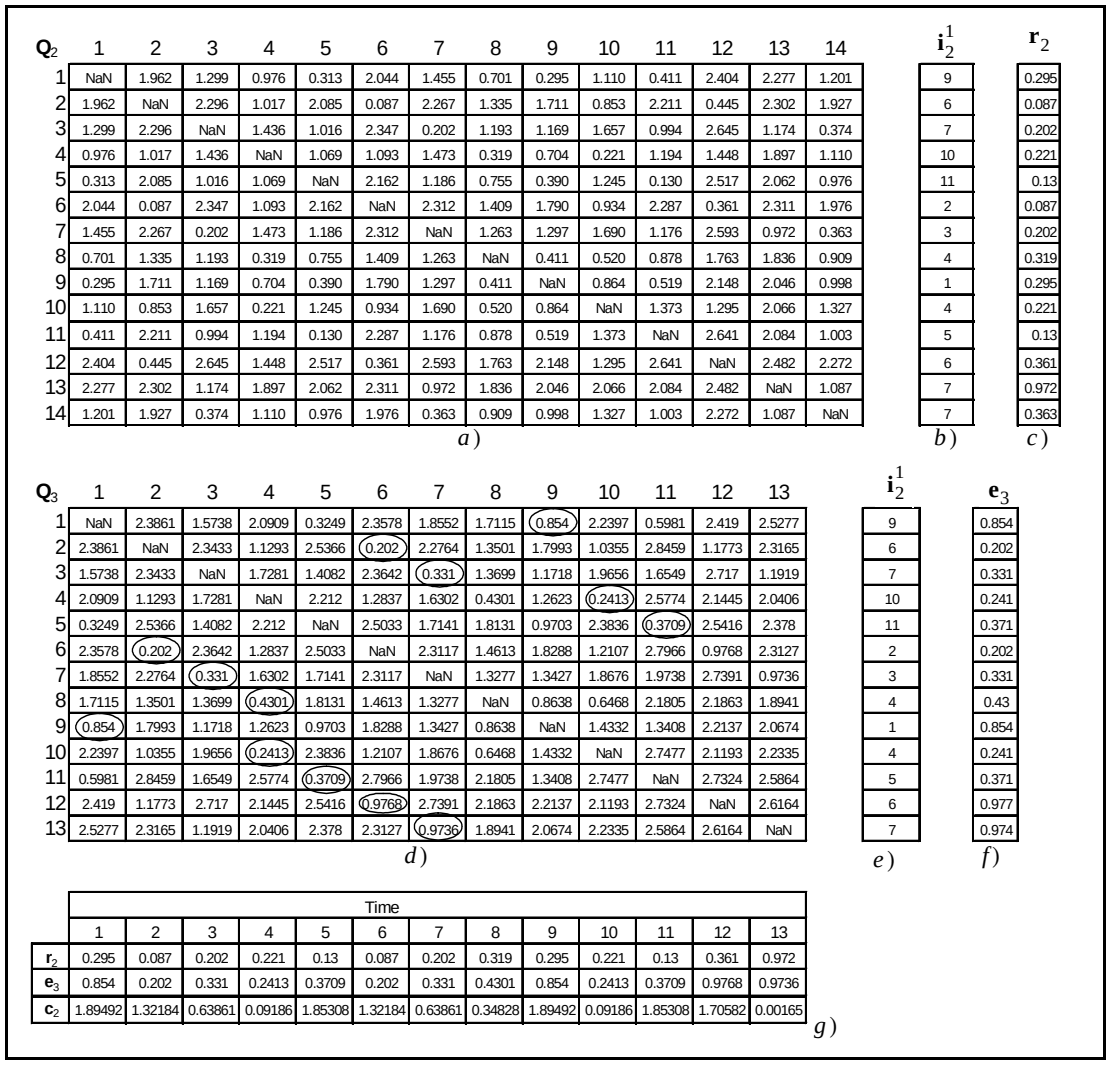
| $\mathbf{e}_2$ |
|---|
| 0.701 |
| 0.087 |
| 0.202 |
| 0.976 |
| 0.130 |
| 0.087 |
| 0.202 |
| 0.411 |
| 0.411 |
| 0.221 |
| 0.130 |
| 0.445 |
| 0.972 |
| 0.374 |

*f*)

| | Time | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| $\mathbf{r}_1$ | 0.026 | 0.002 | 0.087 | 0.156 | 0.087 | 0.002 | 0.087 | 0.025 | 0.025 | 0.204 | 0.087 | 0.094 | 0.348 | 0.267 |
| $\mathbf{e}_2$ | 0.701 | 0.087 | 0.202 | 0.976 | 0.130 | 0.087 | 0.202 | 0.411 | 0.411 | 0.221 | 0.130 | 0.445 | 0.972 | 0.374 |
| $\mathbf{c}_1$ | 25.9731 | 42.55 | 1.32184 | 5.25385 | 0.48851 | 42.55 | 1.32184 | 15.452 | 15.452 | 0.08529 | 0.48851 | 3.73085 | 1.79339 | 0.39888 |
| | FNN | FNN | | | | FNN | | FNN | FNN | | | | | |

*g*)

Table 5.4 The CDD method tables for $d = 1$



| $\mathbf{Q}_2$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 1.962 | 1.299 | 0.976 | 0.313 | 2.044 | 1.455 | 0.701 | 0.295 | 1.110 | 0.411 | 2.404 | 2.277 | 1.201 |
| 2 | 1.962 | NaN | 2.296 | 1.017 | 2.085 | 0.087 | 2.267 | 1.335 | 1.711 | 0.853 | 2.211 | 0.445 | 2.302 | 1.927 |
| 3 | 1.299 | 2.296 | NaN | 1.436 | 1.016 | 2.347 | 0.202 | 1.193 | 1.169 | 1.657 | 0.994 | 2.645 | 1.174 | 0.374 |
| 4 | 0.976 | 1.017 | 1.436 | NaN | 1.069 | 1.093 | 1.473 | 0.319 | 0.704 | 0.221 | 1.194 | 1.448 | 1.897 | 1.110 |
| 5 | 0.313 | 2.085 | 1.016 | 1.069 | NaN | 2.162 | 1.186 | 0.755 | 0.390 | 1.245 | 0.130 | 2.517 | 2.062 | 0.976 |
| 6 | 2.044 | 0.087 | 2.347 | 1.093 | 2.162 | NaN | 2.312 | 1.409 | 1.790 | 0.934 | 2.287 | 0.361 | 2.311 | 1.976 |
| 7 | 1.455 | 2.267 | 0.202 | 1.473 | 1.186 | 2.312 | NaN | 1.263 | 1.297 | 1.690 | 1.176 | 2.593 | 0.972 | 0.363 |
| 8 | 0.701 | 1.335 | 1.193 | 0.319 | 0.755 | 1.409 | 1.263 | NaN | 0.411 | 0.520 | 0.878 | 1.763 | 1.836 | 0.909 |
| 9 | 0.295 | 1.711 | 1.169 | 0.704 | 0.390 | 1.790 | 1.297 | 0.411 | NaN | 0.864 | 0.519 | 2.148 | 2.046 | 0.998 |
| 10 | 1.110 | 0.853 | 1.657 | 0.221 | 1.245 | 0.934 | 1.690 | 0.520 | 0.864 | NaN | 1.373 | 1.295 | 2.066 | 1.327 |
| 11 | 0.411 | 2.211 | 0.994 | 1.194 | 0.130 | 2.287 | 1.176 | 0.878 | 0.519 | 1.373 | NaN | 2.641 | 2.084 | 1.003 |
| 12 | 2.404 | 0.445 | 2.645 | 1.448 | 2.517 | 0.361 | 2.593 | 1.763 | 2.148 | 1.295 | 2.641 | NaN | 2.482 | 2.272 |
| 13 | 2.277 | 2.302 | 1.174 | 1.897 | 2.062 | 2.311 | 0.972 | 1.836 | 2.046 | 2.066 | 2.084 | 2.482 | NaN | 1.087 |
| 14 | 1.201 | 1.927 | 0.374 | 1.110 | 0.976 | 1.976 | 0.363 | 0.909 | 0.998 | 1.327 | 1.003 | 2.272 | 1.087 | NaN |

*a*)

| $\mathbf{i}_2^1$ |
|---|
| 9 |
| 6 |
| 7 |
| 10 |
| 11 |
| 2 |
| 3 |
| 4 |
| 1 |
| 4 |
| 5 |
| 6 |
| 7 |
| 7 |

*b*)

| $\mathbf{r}_2$ |
|---|
| 0.295 |
| 0.087 |
| 0.202 |
| 0.221 |
| 0.13 |
| 0.087 |
| 0.202 |
| 0.319 |
| 0.295 |
| 0.221 |
| 0.13 |
| 0.361 |
| 0.972 |
| 0.363 |

*c*)

| $\mathbf{Q}_3$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 2.3861 | 1.5738 | 2.0909 | 0.3249 | 2.3578 | 1.8552 | 1.7115 | 0.854 | 2.2397 | 0.5981 | 2.419 | 2.5277 |
| 2 | 2.3861 | NaN | 2.3433 | 1.1293 | 2.5366 | 0.202 | 2.2764 | 1.3501 | 1.7993 | 1.0355 | 2.8459 | 1.1773 | 2.3165 |
| 3 | 1.5738 | 2.3433 | NaN | 1.7281 | 1.4082 | 2.3642 | 0.331 | 1.3699 | 1.1718 | 1.9656 | 1.6549 | 2.717 | 1.1919 |
| 4 | 2.0909 | 1.1293 | 1.7281 | NaN | 2.212 | 1.2837 | 1.6302 | 0.4301 | 1.2623 | 0.2413 | 2.5774 | 2.1445 | 2.0406 |
| 5 | 0.3249 | 2.5366 | 1.4082 | 2.212 | NaN | 2.5033 | 1.7141 | 1.8131 | 0.9703 | 2.3836 | 0.3709 | 2.5416 | 2.378 |
| 6 | 2.3578 | 0.202 | 2.3642 | 1.2837 | 2.5033 | NaN | 2.3117 | 1.4613 | 1.8288 | 1.2107 | 2.7966 | 0.9768 | 2.3127 |
| 7 | 1.8552 | 2.2764 | 0.331 | 1.6302 | 1.7141 | 2.3117 | NaN | 1.3277 | 1.3427 | 1.8676 | 1.9738 | 2.7391 | 0.9736 |
| 8 | 1.7115 | 1.3501 | 1.3699 | 0.4301 | 1.8131 | 1.4613 | 1.3277 | NaN | 0.8638 | 0.6468 | 2.1805 | 2.1863 | 1.8941 |
| 9 | 0.854 | 1.7993 | 1.1718 | 1.2623 | 0.9703 | 1.8288 | 1.3427 | 0.8638 | NaN | 1.4332 | 1.3408 | 2.2137 | 2.0674 |
| 10 | 2.2397 | 1.0355 | 1.9656 | 0.2413 | 2.3836 | 1.2107 | 1.8676 | 0.6468 | 1.4332 | NaN | 2.7477 | 2.1193 | 2.2335 |
| 11 | 0.5981 | 2.8459 | 1.6549 | 2.5774 | 0.3709 | 2.7966 | 1.9738 | 2.1805 | 1.3408 | 2.7477 | NaN | 2.7324 | 2.5864 |
| 12 | 2.419 | 1.1773 | 2.717 | 2.1445 | 2.5416 | 0.9768 | 2.7391 | 2.1863 | 2.2137 | 2.1193 | 2.7324 | NaN | 2.6164 |
| 13 | 2.5277 | 2.3165 | 1.1919 | 2.0406 | 2.378 | 2.3127 | 0.9736 | 1.8941 | 2.0674 | 2.2335 | 2.5864 | 2.6164 | NaN |

*d*)

| $\mathbf{i}_2^1$ |
|---|
| 9 |
| 6 |
| 7 |
| 10 |
| 11 |
| 2 |
| 3 |
| 4 |
| 1 |
| 4 |
| 5 |
| 6 |
| 7 |

*e*)

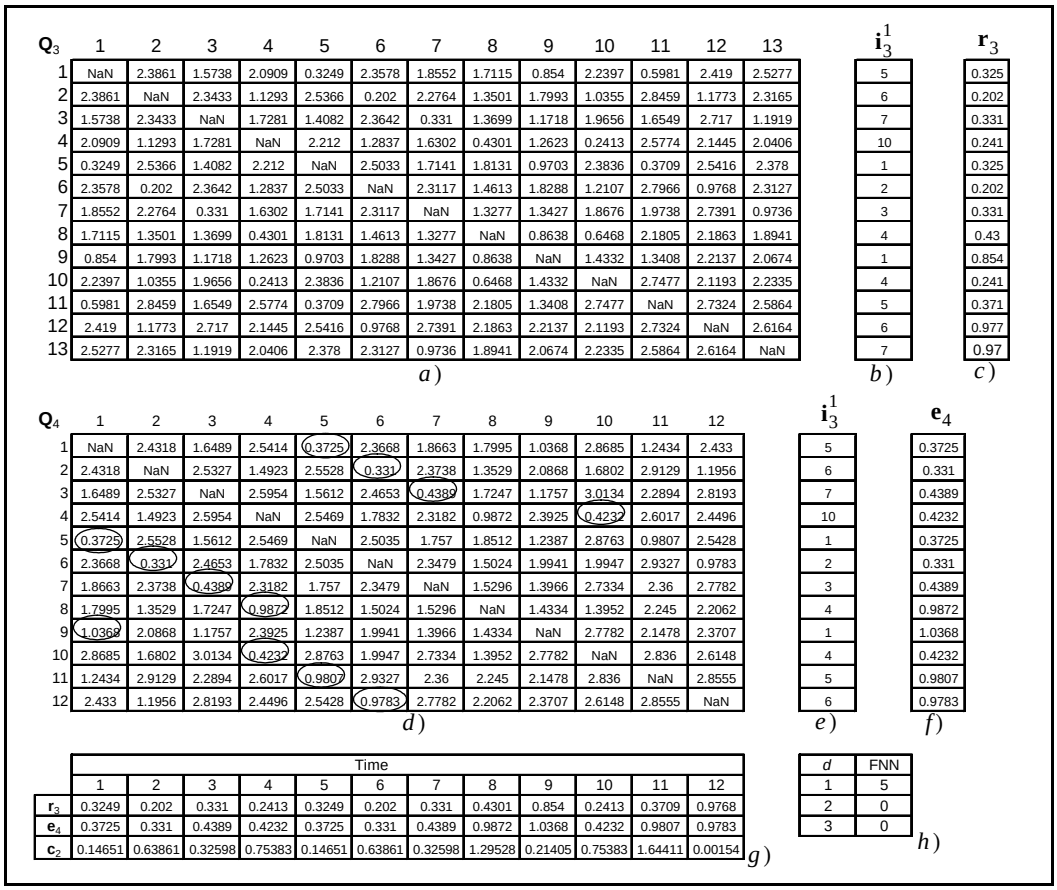
| $\mathbf{e}_3$ |
|---|
| 0.854 |
| 0.202 |
| 0.331 |
| 0.241 |
| 0.371 |
| 0.202 |
| 0.331 |
| 0.43 |
| 0.854 |
| 0.241 |
| 0.371 |
| 0.977 |
| 0.974 |

*f*)

| | Time | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| $\mathbf{r}_2$ | 0.295 | 0.087 | 0.202 | 0.221 | 0.13 | 0.087 | 0.202 | 0.319 | 0.295 | 0.221 | 0.13 | 0.361 | 0.972 |
| $\mathbf{e}_3$ | 0.854 | 0.202 | 0.331 | 0.2413 | 0.3709 | 0.202 | 0.331 | 0.4301 | 0.854 | 0.2413 | 0.3709 | 0.9768 | 0.9736 |
| $\mathbf{c}_2$ | 1.89492 | 1.32184 | 0.63861 | 0.09186 | 1.85308 | 1.32184 | 0.63861 | 0.34828 | 1.89492 | 0.09186 | 1.85308 | 1.70582 | 0.00165 |

*g*)

Table 5.5 The CDD method tables for $d = 2$



| $\mathbf{Q}_3$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 2.3861 | 1.5738 | 2.0909 | 0.3249 | 2.3578 | 1.8552 | 1.7115 | 0.854 | 2.2397 | 0.5981 | 2.419 | 2.5277 |
| 2 | 2.3861 | NaN | 2.3433 | 1.1293 | 2.5366 | 0.202 | 2.2764 | 1.3501 | 1.7993 | 1.0355 | 2.8459 | 1.1773 | 2.3165 |
| 3 | 1.5738 | 2.3433 | NaN | 1.7281 | 1.4082 | 2.3642 | 0.331 | 1.3699 | 1.1718 | 1.9656 | 1.6549 | 2.717 | 1.1919 |
| 4 | 2.0909 | 1.1293 | 1.7281 | NaN | 2.212 | 1.2837 | 1.6302 | 0.4301 | 1.2623 | 0.2413 | 2.5774 | 2.1445 | 2.0406 |
| 5 | 0.3249 | 2.5366 | 1.4082 | 2.212 | NaN | 2.5033 | 1.7141 | 1.8131 | 0.9703 | 2.3836 | 0.3709 | 2.5416 | 2.378 |
| 6 | 2.3578 | 0.202 | 2.3642 | 1.2837 | 2.5033 | NaN | 2.3117 | 1.4613 | 1.8288 | 1.2107 | 2.7966 | 0.9768 | 2.3127 |
| 7 | 1.8552 | 2.2764 | 0.331 | 1.6302 | 1.7141 | 2.3117 | NaN | 1.3277 | 1.3427 | 1.8676 | 1.9738 | 2.7391 | 0.9736 |
| 8 | 1.7115 | 1.3501 | 1.3699 | 0.4301 | 1.8131 | 1.4613 | 1.3277 | NaN | 0.8638 | 0.6468 | 2.1805 | 1.2863 | 1.8941 |
| 9 | 0.854 | 1.7993 | 1.1718 | 1.2623 | 0.9703 | 1.8288 | 1.3427 | 0.8638 | NaN | 1.4332 | 1.3408 | 2.2137 | 2.0674 |
| 10 | 2.2397 | 1.0355 | 1.9656 | 0.2413 | 2.3836 | 1.2107 | 1.8676 | 0.6468 | 1.4332 | NaN | 2.7477 | 2.1193 | 2.2335 |
| 11 | 0.5981 | 2.8459 | 1.6549 | 2.5774 | 0.3709 | 2.7966 | 1.9738 | 2.1805 | 1.3408 | 2.7477 | NaN | 2.7324 | 2.5864 |
| 12 | 2.419 | 1.1773 | 2.717 | 2.1445 | 2.5416 | 0.9768 | 2.7391 | 2.1863 | 2.2137 | 2.1193 | 2.7324 | NaN | 2.6164 |
| 13 | 2.5277 | 2.3165 | 1.1919 | 2.0406 | 2.378 | 2.3127 | 0.9736 | 1.8941 | 2.0674 | 2.2335 | 2.5864 | 2.6164 | NaN |

*a*)

| $\mathbf{i}_3^1$ |
|---|
| 5 |
| 6 |
| 7 |
| 10 |
| 1 |
| 2 |
| 3 |
| 4 |
| 1 |
| 4 |
| 5 |
| 6 |
| 7 |

*b*)

| $\mathbf{r}_3$ |
|---|
| 0.325 |
| 0.202 |
| 0.331 |
| 0.241 |
| 0.325 |
| 0.202 |
| 0.331 |
| 0.43 |
| 0.854 |
| 0.241 |
| 0.371 |
| 0.977 |
| 0.97 |

*c*)

| $\mathbf{Q}_4$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 2.4318 | 1.6489 | 2.5414 | 0.3725 | 2.3668 | 1.8663 | 1.7995 | 1.0368 | 2.8685 | 1.2434 | 2.433 |
| 2 | 2.4318 | NaN | 2.5327 | 1.4923 | 2.5528 | 0.331 | 2.3738 | 1.3529 | 2.0868 | 1.6802 | 2.9129 | 1.1956 |
| 3 | 1.6489 | 2.5327 | NaN | 2.5954 | 1.5612 | 2.4653 | 0.4389 | 1.7247 | 1.1757 | 3.0134 | 2.2894 | 2.8193 |
| 4 | 2.5414 | 1.4923 | 2.5954 | NaN | 2.5469 | 1.7832 | 2.3182 | 0.9872 | 2.3925 | 0.4232 | 2.6017 | 2.4496 |
| 5 | 0.3725 | 2.5528 | 1.5612 | 2.5469 | NaN | 2.5035 | 1.757 | 1.8512 | 1.2387 | 2.8763 | 0.9807 | 2.5428 |
| 6 | 2.3668 | 0.331 | 2.4653 | 1.7832 | 2.5035 | NaN | 2.3479 | 1.5024 | 1.9941 | 1.9947 | 2.9327 | 0.9783 |
| 7 | 1.8663 | 2.3738 | 0.4389 | 2.3182 | 1.757 | 2.3479 | NaN | 1.5296 | 1.3966 | 2.7334 | 2.36 | 2.7782 |
| 8 | 1.7995 | 1.3529 | 1.7247 | 0.9872 | 1.8512 | 1.5024 | 1.5296 | NaN | 1.4334 | 1.3952 | 2.245 | 2.2062 |
| 9 | 1.0368 | 2.0868 | 1.1757 | 2.3925 | 1.2387 | 1.9941 | 1.3966 | 1.4334 | NaN | 2.7782 | 2.1478 | 2.3707 |
| 10 | 2.8685 | 1.6802 | 3.0134 | 0.4232 | 2.8763 | 1.9947 | 2.7334 | 1.3952 | 2.7782 | NaN | 2.836 | 2.6148 |
| 11 | 1.2434 | 2.9129 | 2.2894 | 2.6017 | 0.9807 | 2.9327 | 2.36 | 2.245 | 2.1478 | 2.836 | NaN | 2.8555 |
| 12 | 2.433 | 1.1956 | 2.8193 | 2.4496 | 2.5428 | 0.9783 | 2.7782 | 2.2062 | 2.3707 | 2.6148 | 2.8555 | NaN |

*d*)

| $\mathbf{i}_3^1$ |
|---|
| 5 |
| 6 |
| 7 |
| 10 |
| 1 |
| 2 |
| 3 |
| 4 |
| 1 |
| 4 |
| 5 |
| 6 |

*e*)

| $\mathbf{e}_4$ |
|---|
| 0.3725 |
| 0.331 |
| 0.4389 |
| 0.4232 |
| 0.3725 |
| 0.331 |
| 0.4389 |
| 0.9872 |
| 1.0368 |
| 0.4232 |
| 0.9807 |
| 0.9783 |

*f*)

| | Time | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| $\mathbf{r}_3$ | 0.3249 | 0.202 | 0.331 | 0.2413 | 0.3249 | 0.202 | 0.331 | 0.4301 | 0.854 | 0.2413 | 0.3709 | 0.9768 |
| $\mathbf{e}_4$ | 0.3725 | 0.331 | 0.4389 | 0.4232 | 0.3725 | 0.331 | 0.4389 | 0.9872 | 1.0368 | 0.4232 | 0.9807 | 0.9783 |
| $\mathbf{c}_2$ | 0.14651 | 0.63861 | 0.32598 | 0.75383 | 0.14651 | 0.63861 | 0.32598 | 1.29528 | 0.21405 | 0.75383 | 1.64411 | 0.00154 |

*g*)

| *d* | FNN |
|---|---|
| 1 | 5 |
| 2 | 0 |
| 3 | 0 |

*h*)

Table 5.6 The CDD method tables for $d = 3$

### 5.2.3 Using the change of distance with time method (CDT)

As we said in Chapter 4, the CDT method can also be used to estimate $d_L$. This method depends on the idea that false nearest neighbor distances grow significantly as time increases. As in the previous methods, we begin by computing the nearest neighbors for $d = 1$. Then we track how the distance between the original nearest neighbors increases with time. This method can be applied by computing the distance in dimension $d_E$, rather than in dimension $d$. This has the advantage that the measured distance won't be affected by projection artifacts.

Next, we need to have a method for determining whether or not the distances between two vectors can be considered large. For our purpose, we will use the average vector

length as a bench mark. If the distance between two vectors is larger than the average vector length, we will consider the distance to be large. The average vector length is defined as

$$\beta = \frac{1}{M}\sum_{k=1}^{M} \|\mathbf{y}_d(k)\| . \tag{5.7}$$

If we let $\mathbf{y}_d(n)$ be the nearest neighbor of $\mathbf{y}_d(m)$, we can check if $\mathbf{y}_d(n)$ is a FNN by measuring the distance between $\mathbf{y}_3(m)$ and $\mathbf{y}_3(n)$ as time increases ($d_E = 3$). For example, we can see from the $8^{th}$ row of Table 5.7*a* that the nearest neighbor of $y_1(8)$ is $y_1(9)$. If we look at Table 5.7*b*, we can see that the distance between $\mathbf{y}_3(8)$ and $\mathbf{y}_3(9)$ is $q_3(8, 9) = 0.8638 < \beta = 1.3404$ (see the circled value). But, after one time step ahead, we can see that the distance between $\mathbf{y}_3(9)$ and $\mathbf{y}_3(10)$ is $q_3(9, 10) = 1.4332$ which is greater than $\beta = 1.3404$ as indicated by the arrow. We conclude that $y_1(9)$ is a FNN since the distance between $\mathbf{y}_3(8)$ and $\mathbf{y}_3(9)$ has grown more than $\beta$ as time increases. A second example of the FNNs can be seen from the first row of Table 5.7*a*. We can see that the nearest neighbor of $y_1(1)$ is $y_1(8)$, while in Table 5.7*b*, the distance between $\mathbf{y}_3(1)$ and $\mathbf{y}_3(8)$ is $1.7115 > \beta$ that means $y_1(8)$ is a FNN. On the other hand, the nearest neighbor of $y_1(2)$ is $y_1(6)$, as seen in the second row of the last column in Table 5.7*a*, while if we look at Table 5.7*b*, we can see that the distance between $\mathbf{y}_3(2)$ and $\mathbf{y}_3(6)$ is $0.202 < \beta$. If we move one step ahead, we can see that the distance between $\mathbf{y}_3(3)$ and $\mathbf{y}_3(7)$ is $0.331$ which is still less than $\beta$. A conclusion that can be reached is that $y_1(6)$ is a true nearest neighbor of $y_1(2)$. Notice here that we choose one as the number of forward time steps that is used to check for FNN.

Table 5.9*b* summarizes the number of FNNs found for $d = 1$ through 3. It shows that at $d = 1$, the number of FNNs found is 4, while at $d = 2$, the number of FNNs is 0

and remains 0 at $d = 3$. That means the minimum embedding dimension is $d_L = 2$. Finally, notice that the nearest neighbor of $\mathbf{y}_d(13)$ shown in the last row of the Tables 5.7*b*, 5.8*b*, and 5.9*a* are labeled as not decidable (nd). The reason for this is that at the first instance of time, the distance between the two vectors is less than $\beta$, but we do not have enough data to check if the distance between the two vectors after one time step ahead is greater than $\beta$. So, we have to discard this neighbor from the count of the FNNs and label it as not decidable (nd).

The threshold is $\beta = 1.3404$

$\mathbf{i}_1^1$

| $Q_1$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 0.65 | 1.2015 | 0.1561 | 0.313 | 0.6479 | 1.2886 | 0.0262 | 0.0508 | 0.3597 | 0.4 | 0.7439 | 1.6362 | 0.9341 | 0.1046 | 8 |
| 2 | 0.65 | NaN | 1.8515 | 0.4939 | 0.963 | 0.0021 | 1.9386 | 0.6762 | 0.7008 | 0.2903 | 1.05 | 0.0939 | 2.2862 | 1.5841 | 0.7546 | 6 |
| 3 | 1.2015 | 1.8515 | NaN | 1.3576 | 0.8885 | 1.8494 | 0.0871 | 1.1753 | 1.1507 | 1.5612 | 0.8015 | 1.9454 | 0.4347 | 0.2674 | 1.0969 | 7 |
| 4 | 0.1561 | 0.4939 | 1.3576 | NaN | 0.4691 | 0.4918 | 1.4446 | 0.1823 | 0.2069 | 0.2036 | 0.556 | 0.5878 | 1.7922 | 1.0901 | 0.2607 | 1 |
| 5 | 0.313 | 0.963 | 0.8885 | 0.4691 | NaN | 0.9609 | 0.9755 | 0.2868 | 0.2622 | 0.6727 | 0.0869 | 1.0569 | 1.3231 | 0.621 | 0.2084 | 11 |
| 6 | 0.6479 | 0.0021 | 1.8494 | 0.4918 | 0.9609 | NaN | 1.9364 | 0.6741 | 0.6987 | 0.2882 | 1.0478 | 0.096 | 2.284 | 1.5819 | 0.7525 | 2 |
| 7 | 1.2886 | 1.9386 | 0.0871 | 1.4446 | 0.9755 | 1.9364 | NaN | 1.2624 | 1.2377 | 1.6483 | 0.8886 | 2.0325 | 0.3476 | 0.3545 | 1.1839 | 3 |
| 8 | 0.0262 | 0.6762 | 1.1753 | 0.1823 | 0.2868 | 0.6741 | 1.2624 | NaN | 0.0246 | 0.3859 | 0.3737 | 0.7701 | 1.61 | 0.9078 | 0.0784 | 9 |
| 9 | 0.0508 | 0.7008 | 1.1507 | 0.2069 | 0.2622 | 0.6987 | 1.2377 | 0.0246 | NaN | 0.4105 | 0.3491 | 0.7947 | 1.5853 | 0.8832 | 0.0538 | 8 |
| 10 | 0.3597 | 0.2903 | 1.5612 | 0.2036 | 0.6727 | 0.2882 | 1.6483 | 0.3859 | 0.4105 | NaN | 0.7596 | 0.3842 | 1.9959 | 1.2937 | 0.4643 | 4 |
| 11 | 0.4 | 1.05 | 0.8015 | 0.556 | 0.0869 | 1.0478 | 0.8886 | 0.3737 | 0.3491 | 0.7596 | NaN | 1.1438 | 1.2362 | 0.5341 | 0.2953 | 5 |
| 12 | 0.7439 | 0.0939 | 1.9454 | 0.5878 | 1.0569 | 0.096 | 2.0325 | 0.7701 | 0.7947 | 0.3842 | 1.1438 | NaN | 2.3801 | 1.6779 | 0.8485 | 2 |
| 13 | 1.6362 | 2.2862 | 0.4347 | 1.7922 | 1.3231 | 2.284 | 0.3476 | 1.61 | 1.5853 | 1.9959 | 1.2362 | 2.3801 | NaN | 0.7021 | 1.5316 | 7 |
| 14 | 0.9341 | 1.5841 | 0.2674 | 1.0901 | 0.621 | 1.5819 | 0.3545 | 0.9078 | 0.8832 | 1.2937 | 0.5341 | 1.6779 | 0.7021 | NaN | 0.8294 | 3 |
| 15 | 0.1046 | 0.7546 | 1.0969 | 0.2607 | 0.2084 | 0.7525 | 1.1839 | 0.0784 | 0.0538 | 0.4643 | 0.2953 | 0.8485 | 1.5316 | 0.8294 | NaN | 9 |

*a*)

$\mathbf{i}_1^1$

| $Q_3$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 2.3861 | 1.5738 | 2.0909 | 0.3249 | 2.3578 | 1.8552 | 1.7115 | 0.854 | 2.2397 | 0.5981 | 2.419 | 2.5277 | 8 | FNN |
| 2 | 2.3861 | NaN | 2.3433 | 1.1293 | 2.5366 | 0.202 | 2.2764 | 1.3501 | 1.7993 | 1.0355 | 2.8459 | 1.1773 | 2.3165 | 6 | |
| 3 | 1.5738 | 2.3433 | NaN | 1.7281 | 1.4082 | 2.3642 | 0.331 | 1.3699 | 1.1718 | 1.9656 | 1.6549 | 2.717 | 1.1919 | 7 | |
| 4 | 2.0909 | 1.1293 | 1.7281 | NaN | 2.212 | 1.2837 | 1.6302 | 0.4301 | 1.2623 | 0.2413 | 2.5774 | 2.1445 | 2.0406 | 1 | FNN |
| 5 | 0.3249 | 2.5366 | 1.4082 | 2.212 | NaN | 2.5033 | 1.7141 | 1.8131 | 0.9703 | 2.3836 | 0.3709 | 2.5416 | 2.378 | 11 | |
| 6 | 2.3578 | 0.202 | 2.3642 | 1.2837 | 2.5033 | NaN | 2.3117 | 1.4613 | 1.8288 | 1.2107 | 2.7966 | 0.9768 | 2.3127 | 2 | |
| 7 | 1.8552 | 2.2764 | 0.331 | 1.6302 | 1.7141 | 2.3117 | NaN | 1.3277 | 1.3427 | 1.8676 | 1.9738 | 2.7391 | 0.9736 | 3 | |
| 8 | 1.7115 | 1.3501 | 1.3699 | 0.4301 | 1.8131 | 1.4613 | 1.3277 | NaN | 0.8638 | 0.6468 | 2.1805 | 2.1863 | 1.8941 | 9 | FNN |
| 9 | 0.854 | 1.7993 | 1.1718 | 1.2623 | 0.9703 | 1.8288 | 1.3427 | 0.8638 | NaN | 1.4332 | 1.3408 | 2.2137 | 2.0674 | 8 | FNN |
| 10 | 2.2397 | 1.0355 | 1.9656 | 0.2413 | 2.3836 | 1.2107 | 1.8676 | 0.6468 | 1.4332 | NaN | 2.7477 | 2.1193 | 2.2335 | 4 | |
| 11 | 0.5981 | 2.8459 | 1.6549 | 2.5774 | 0.3709 | 2.7966 | 1.9738 | 2.1805 | 1.3408 | 2.7477 | NaN | 2.7324 | 2.5864 | 5 | |
| 12 | 2.419 | 1.1773 | 2.717 | 2.1445 | 2.5416 | 0.9768 | 2.7391 | 2.1863 | 2.2137 | 2.1193 | 2.7324 | NaN | 2.6164 | 2 | |
| 13 | 2.5277 | 2.3165 | 1.1919 | 2.0406 | 2.378 | 2.3127 | 0.9736 | 1.8941 | 2.0674 | 2.2335 | 2.5864 | 2.6164 | NaN | 7 | nd |

*b*)

At $d$ =1, the number of FNN = 4

nd: not decidable

Table 5.7 The CDT method for $d$ = 1

$\mathbf{Q}_2$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | $\mathbf{i}_2^1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 1.9623 | 1.2991 | 0.9756 | 0.313 | 2.044 | 1.4552 | 0.7013 | 0.2947 | 1.1099 | 0.4108 | 2.4041 | 2.2773 | 1.2008 | 9 |
| 2 | 1.9623 | NaN | 2.2959 | 1.0165 | 2.0851 | 0.0871 | 2.267 | 1.3346 | 1.7113 | 0.8525 | 2.2106 | 0.4447 | 2.3018 | 1.9268 | 6 |
| 3 | 1.2991 | 2.2959 | NaN | 1.4363 | 1.0155 | 2.3467 | 0.202 | 1.1934 | 1.1685 | 1.6572 | 0.994 | 2.6451 | 1.1736 | 0.3735 | 7 |
| 4 | 0.9756 | 1.0165 | 1.4363 | NaN | 1.0693 | 1.0925 | 1.4728 | 0.3193 | 0.7038 | 0.2214 | 1.1942 | 1.4478 | 1.8968 | 1.1099 | 10 |
| 5 | 0.313 | 2.0851 | 1.0155 | 1.0693 | NaN | 2.1617 | 1.1858 | 0.7553 | 0.3896 | 1.2452 | 0.1295 | 2.5167 | 2.0623 | 0.9757 | 11 |
| 6 | 2.044 | 0.0871 | 2.3467 | 1.0925 | 2.1617 | NaN | 2.3116 | 1.4094 | 1.7902 | 0.9342 | 2.2867 | 0.3606 | 2.3114 | 1.9759 | 2 |
| 7 | 1.4552 | 2.267 | 0.202 | 1.4728 | 1.1858 | 2.3116 | NaN | 1.2626 | 1.2965 | 1.6901 | 1.1759 | 2.5928 | 0.9721 | 0.3631 | 3 |
| 8 | 0.7013 | 1.3346 | 1.1934 | 0.3193 | 0.7553 | 1.4094 | 1.2626 | NaN | 0.4113 | 0.5204 | 0.8782 | 1.7625 | 1.8363 | 0.9094 | 4 |
| 9 | 0.2947 | 1.7113 | 1.1685 | 0.7038 | 0.3896 | 1.7902 | 1.2965 | 0.4113 | NaN | 0.8635 | 0.5191 | 2.1483 | 2.0462 | 0.9978 | 1 |
| 10 | 1.1099 | 0.8525 | 1.6572 | 0.2214 | 1.2452 | 0.9342 | 1.6901 | 0.5204 | 0.8635 | NaN | 1.3731 | 1.2945 | 2.0661 | 1.327 | 4 |
| 11 | 0.4108 | 2.2106 | 0.994 | 1.1942 | 0.1295 | 2.2867 | 1.1759 | 0.8782 | 0.5191 | 1.3731 | NaN | 2.6407 | 2.0842 | 1.0026 | 5 |
| 12 | 2.4041 | 0.4447 | 2.6451 | 1.4478 | 2.5167 | 0.3606 | 2.5928 | 1.7625 | 2.1483 | 1.2945 | 2.6407 | NaN | 2.4815 | 2.2718 | 6 |
| 13 | 2.2773 | 2.3018 | 1.1736 | 1.8968 | 2.0623 | 2.3114 | 0.9721 | 1.8363 | 2.0462 | 2.0661 | 2.0842 | 2.4815 | NaN | 1.0867 | 7 |
| 14 | 1.2008 | 1.9268 | 0.3735 | 1.1099 | 0.9757 | 1.9759 | 0.3631 | 0.9094 | 0.9978 | 1.327 | 1.0026 | 2.2718 | 1.0867 | NaN | 7 |

*a*)

$\mathbf{Q}_3$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | $\mathbf{i}_2^1$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 2.3861 | 1.5738 | 2.0909 | 0.3249 | 2.3578 | 1.8552 | 1.7115 | 0.854 | 2.2397 | 0.5981 | 2.419 | 2.5277 | 9 | |
| 2 | 2.3861 | NaN | 2.3433 | 1.1293 | 2.5366 | 0.202 | 2.2764 | 1.3501 | 1.7993 | 1.0355 | 2.8459 | 1.1773 | 2.3165 | 6 | |
| 3 | 1.5738 | 2.3433 | NaN | 1.7281 | 1.4082 | 2.3642 | 0.331 | 1.3699 | 1.1718 | 1.9656 | 1.6549 | 2.717 | 1.1919 | 7 | |
| 4 | 2.0909 | 1.1293 | 1.7281 | NaN | 2.212 | 1.2837 | 1.6302 | 0.4301 | 1.2623 | 0.2413 | 2.5774 | 2.1445 | 2.0406 | 10 | |
| 5 | 0.3249 | 2.5366 | 1.4082 | 2.212 | NaN | 2.5033 | 1.7141 | 1.8131 | 0.9703 | 2.3836 | 0.3709 | 2.5416 | 2.378 | 11 | |
| 6 | 2.3578 | 0.202 | 2.3642 | 1.2837 | 2.5033 | NaN | 2.3117 | 1.4613 | 1.8288 | 1.2107 | 2.7966 | 0.9768 | 2.3127 | 2 | |
| 7 | 1.8552 | 2.2764 | 0.331 | 1.6302 | 1.7141 | 2.3117 | NaN | 1.3277 | 1.3427 | 1.8676 | 1.9738 | 2.7391 | 0.9736 | 3 | |
| 8 | 1.7115 | 1.3501 | 1.3699 | 0.4301 | 1.8131 | 1.4613 | 1.3277 | NaN | 0.8638 | 0.6468 | 2.1805 | 2.1863 | 1.8941 | 4 | |
| 9 | 0.854 | 1.7993 | 1.1718 | 1.2623 | 0.9703 | 1.8288 | 1.3427 | 0.8638 | NaN | 1.4332 | 1.3408 | 2.2137 | 2.0674 | 1 | |
| 10 | 2.2397 | 1.0355 | 1.9656 | 0.2413 | 2.3836 | 1.2107 | 1.8676 | 0.6468 | 1.4332 | NaN | 2.7477 | 2.1193 | 2.2335 | 4 | |
| 11 | 0.5981 | 2.8459 | 1.6549 | 2.5774 | 0.3709 | 2.7966 | 1.9738 | 2.1805 | 1.3408 | 2.7477 | NaN | 2.7324 | 2.5864 | 5 | |
| 12 | 2.419 | 1.1773 | 2.717 | 2.1445 | 2.5416 | 0.9768 | 2.7391 | 2.1863 | 2.2137 | 2.1193 | 2.7324 | NaN | 2.6164 | 6 | |
| 13 | 2.5277 | 2.3165 | 1.1919 | 2.0406 | 2.378 | 2.3127 | 0.9736 | 1.8941 | 2.0674 | 2.2335 | 2.5864 | 2.6164 | NaN | 7 | nd |

*b*)

At $d$ =2, the number of FNN is 0

Table 5.8 The CDT method for $d$ = 2

$\mathbf{Q}_3$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | $\mathbf{i}_3^1$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 2.3861 | 1.5738 | 2.0909 | 0.3249 | 2.3578 | 1.8552 | 1.7115 | 0.854 | 2.2397 | 0.5981 | 2.419 | 2.5277 | 5 | |
| 2 | 2.3861 | NaN | 2.3433 | 1.1293 | 2.5366 | 0.202 | 2.2764 | 1.3501 | 1.7993 | 1.0355 | 2.8459 | 1.1773 | 2.3165 | 6 | |
| 3 | 1.5738 | 2.3433 | NaN | 1.7281 | 1.4082 | 2.3642 | 0.331 | 1.3699 | 1.1718 | 1.9656 | 1.6549 | 2.717 | 1.1919 | 7 | |
| 4 | 2.0909 | 1.1293 | 1.7281 | NaN | 2.212 | 1.2837 | 1.6302 | 0.4301 | 1.2623 | 0.2413 | 2.5774 | 2.1445 | 2.0406 | 10 | |
| 5 | 0.3249 | 2.5366 | 1.4082 | 2.212 | NaN | 2.5033 | 1.7141 | 1.8131 | 0.9703 | 2.3836 | 0.3709 | 2.5416 | 2.378 | 1 | |
| 6 | 2.3578 | 0.202 | 2.3642 | 1.2837 | 2.5033 | NaN | 2.3117 | 1.4613 | 1.8288 | 1.2107 | 2.7966 | 0.9768 | 2.3127 | 2 | |
| 7 | 1.8552 | 2.2764 | 0.331 | 1.6302 | 1.7141 | 2.3117 | NaN | 1.3277 | 1.3427 | 1.8676 | 1.9738 | 2.7391 | 0.9736 | 3 | |
| 8 | 1.7115 | 1.3501 | 1.3699 | 0.4301 | 1.8131 | 1.4613 | 1.3277 | NaN | 0.8638 | 0.6468 | 2.1805 | 2.1863 | 1.8941 | 4 | |
| 9 | 0.854 | 1.7993 | 1.1718 | 1.2623 | 0.9703 | 1.8288 | 1.3427 | 0.8638 | NaN | 1.4332 | 1.3408 | 2.2137 | 2.0674 | 1 | |
| 10 | 2.2397 | 1.0355 | 1.9656 | 0.2413 | 2.3836 | 1.2107 | 1.8676 | 0.6468 | 1.4332 | NaN | 2.7477 | 2.1193 | 2.2335 | 4 | |
| 11 | 0.5981 | 2.8459 | 1.6549 | 2.5774 | 0.3709 | 2.7966 | 1.9738 | 2.1805 | 1.3408 | 2.7477 | NaN | 2.7324 | 2.5864 | 5 | |
| 12 | 2.419 | 1.1773 | 2.717 | 2.1445 | 2.5416 | 0.9768 | 2.7391 | 2.1863 | 2.2137 | 2.1193 | 2.7324 | NaN | 2.6164 | 6 | |
| 13 | 2.5277 | 2.3165 | 1.1919 | 2.0406 | 2.378 | 2.3127 | 0.9736 | 1.8941 | 2.0674 | 2.2335 | 2.5864 | 2.6164 | NaN | 7 | nd |

*a*)

At $d$ =3, the number of FNNs is 0 also.

| $d$ | FNNs |
|---|---|
| 1 | 4 |
| 2 | 0 |
| 3 | 0 |

*b*)

Table 5.9 The CDT method for $d$ = 3

The circled distances in the above tables are the distances between the reference points and their nearest neighbors at the first instance of time in $\Re^3$. The arrows represent the direction where the reference points and their nearest neighbors move after one time step.

## 5.3 The Predictive technique

Now that we have discussed the three geometric methods (CND, CDD, and CDT), we will now talk about the predictive technique. We said in Chapter 4 that the predictive technique can also be used to estimate $d_L$ by approximating the function $\mu$ in Equation (4.6). We also said that we will approximate $\mu$ by using a neural network. Since chaotic systems are nonlinear, we need to use a nonlinear network to make the approximation.

To estimate $d_L$ of the Henon map by using the predictive technique, we will use a nonlinear neural network which consists of two layers. The first layer has 2 neurons with sigmoid transfer functions and a Tapped Delay Line (TDL) connected to it. The second layer has one neuron with a linear transfer function. The TDL is fed with the measurements $y(m)$ to produce the delay-vectors $\mathbf{y}_d(m-1)$. The network structure is shown in Figure 5.1. While the number of taps in the TDL (*d*) is changed from 1 to 2 to 3, the network is trained to predict the current measurement $y(m)$. After the end of the training process, the sum of the squares of the prediction errors (SSE) is recorded as a function of *d*. We applied this technique to 100 points from the $X_1$-coordinate of the Henon map. As we can see from Table 5.10, the SSE changes significantly as *d* increases from 1 to 2 and remains almost the same at $d = 3$. We conclude that the network has accurately approximated $\mu$ and that $d_L = 2$. The log-plot of the SSE versus *d* is shown in Figure 5.2.

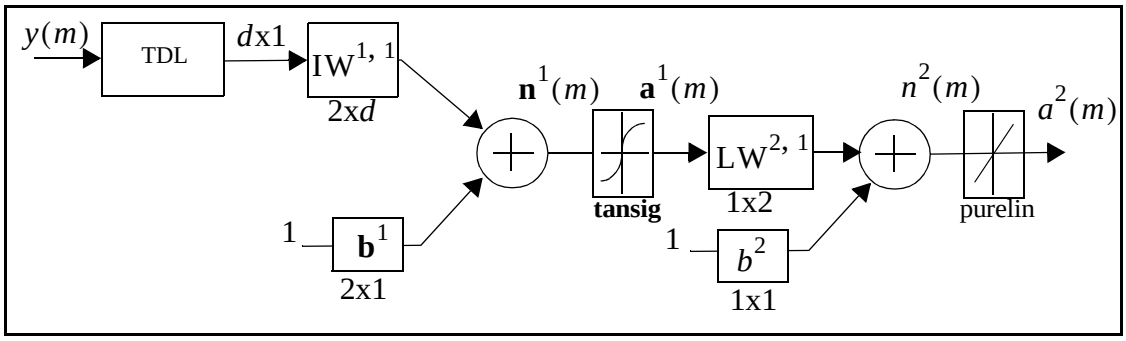



Figure 5.1 The nonlinear network used to estimate $d_L$ of the Henon map

| $d$ | SSE($d$) |
|---|---|
| 1 | 4.0321 |
| 2 | $4.7117 \times 10^{-7}$ |
| 3 | $4.0824 \times 10^{-7}$ |

Table 5.10 The neural network SSE as a function of $d$

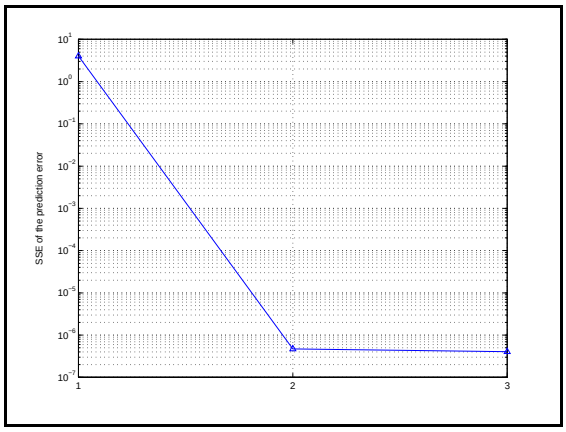



Figure 5.2 Log-Plot of the SSE versus $d$ for the Henon map.

## 5.4 Chapter summary

In this chapter, we demonstrated the estimation of the minimum embedding dimension of the Henon map by using the geometric and the predictive techniques. The purpose of this chapter was to provide some insight into the operation of the algorithms. In the next chapter, we will present the complete algorithms in full detail and discuss practical issues in using the algorithms on more complex systems than the Henon map.

# CHAPTER 6

# ADVANCED ALGORITHMS FOR ESTIMATING THE MINIMUM EMBEDDING DIMENSION

## 6.1 Introduction

In Chapter 3, we explained that two parameters are needed in order to apply the embedding theorem to model a chaotic system. These two parameters are the dimension (*d*) of the delay-vector ($\mathbf{y}_d$) and the delay-time (*T*) between the delay-vector coordinates. We presented in Chapter 4 three geometric methods (CND, CDD, and CDT) and one predictive method through which we estimate the minimum dimension ($d_L$) required to embed the system's attractor. More details were given in Chapter 5 regarding the application of the four methods to estimate $d_L$ of the Henon map. The purpose of this chapter is to give full detail of six algorithms which use the four methods mentioned above to estimate $d_L$. We will also show a method used to find the delay-time *T*. Before that, we first give a summary of the four methods.

The CND, CDD, and the CDT geometric methods depend on the idea that if the dimension of the space is not large enough to represent the attractor of the system, projection artifacts will appear in the projected attractor. These artifacts cause points on the attractor to be falsely projected close to each other and produce False Nearest Neighbors (FNN) (see Section 4.3 for more detail). The dimension $d_L$ can be estimated as the minimum dimen-

sion where the percentage of the FNNs does not change significantly with further increase in dimension.

The CND method detects the existence of FNNs by checking to see if the nearest neighbors in the space of dimension *d* remain neighbors in dimension $d + 1$. On the other hand, the CDD method detects the existence of FNNs by checking to see if the distance between the nearest neighbors in dimension *d* will increase significantly as the dimension increases to $d + 1$. For the case of the CDT method, detection of the existence of FNNs is done by checking to see if the distance between the nearest neighbors in dimension *d* will change significantly as time increases.

On the other hand, in the predictive method, the estimation of $d_L$ is done by approximating the function $\mu: \Re^d \rightarrow \Re^1$ that operates on the reconstructed attractor. $\mu$ is approximated by using a neural network with a Tapped Delay Line (TDL) connected to its input. As the number of taps in the TDL (*d*) increases, the prediction error decreases. At one point, further increase of *d* does not improve the prediction error. At this point, $d_L$ is found.

In the remaining parts of this chapter, we present in Section 6.2 a method used to find the delay-time *T*. In Section 6.3, we present two different algorithms based on the CDD and the CDT methods which were proposed by Abarbanel *et al* to estimated $d_L$. In Section 6.4, we discuss some limitations of the previous two algorithms and suggest four new algorithms to overcome these limitations. In Section 6.5, we present three of the four algorithms, which are based on the geometric technique. The fourth algorithm, which is based on the predictive techniques, is presented in Section 6.6. At the end, Section 6.7 concludes with a chapter summary.

## 6.2 The delay-time (*T*)

We explained in Section 4.2 that we wanted to find the values of the parameters *d* and *T* such that the system we developed by using the delay-vectors $\mathbf{y}_d(m)$ (see Equation (6.4)) is equivalent to the original unknown system of $\mathbf{x}(m)$ (see Equation (4.1)). In Chapters 4 and 5, we discussed the estimation of the minimum embedding dimension of $\mathbf{y}_d(m)$. In this section, we will discuss the delay-time *T*. *T* represents the time difference between the consecutive coordinates of $\mathbf{y}_d(m)$. If the value of *T* is too small, the different coordinates of $\mathbf{y}_d(m)$ will be highly correlated with each other. As a consequence, no new information is given to the equivalent system by the addition of new coordinates to $\mathbf{y}_d(m)$. An example of this case is measuring the room temperature every $1ms$. The resulting consecutive measurements will be almost the same. However, if the value of *T* is too large, the different coordinates of $\mathbf{y}_d(m)$ will be independent of each other and may look random. As a result, we will not be able to capture changes in the dynamics of the system.

One way to find a suitable *T* is by using the average mutual information [FS86, Fra89]. If *A* and *B* are two sets of measurements with elements $a_i$ and $b_j$ respectively, the mutual information between $a_i$ and $b_j$ is the amount learned by the measurement $a_i$ about the measurement $b_j$ which is

$$\log_2\left(\frac{P_{AB}(a_i, b_j)}{P_A(a_i)P_B(b_j)}\right), \tag{6.1}$$

where $P_A(a_i)$ and $P_B(b_j)$ are the individual probabilities of the measurements $a_i$ and $b_j$ respectively, while $P_{AB}(a_i, b_j)$ is the joint probability. The average mutual information between the sets of measurements *A* and *B* is:

$$I_{AB} = \sum_{j,\,k} P_{AB}(a_j, b_k)\log_2\left(\frac{P_{AB}(a_j, b_k)}{P_A(a_j)P_B(b_k)}\right). \tag{6.2}$$

Here $a_i$ represents the measurement $y(m)$ and $b_j$ represents the delayed version of $y(m)$ which is $y(m-T)$. The individual probabilities can be replaced by the histograms of $y(m)$ and $y(m-T)$. The joint probability, on the other hand, can be replaced by the histogram of the vector $\left[y(m)\ y(m-T)\right]^t$. So, the average mutual information between $y(m)$ and $y(m-T)$ is the amount learned by the measurement $y(m)$ about the measurement $y(m-T)$ which is

$$I(T) = \sum_m P\left(\left[x(m)\ y(m-T)\right]^t\right)\log_2\left(\frac{P\left(\left[y(m)\ y(m-T)\right]^t\right)}{P(y(m))P(y(m-T))}\right). \tag{6.3}$$

By evaluating this equation at $T = 1, 2, \ldots, T_{max}$, we choose the delay-time to be the location of the first minimum of $I(T)$, where $T_{max}$ is the maximum value of *T*. As an example, Figure 6.1 shows the average mutual information *I* versus *T* for the Lorenz model using 100,000 points from its X-coordinate (see Section 7.2.1). The value of *T* is changed from 1 through 50. The first minimum is found at $T = 10$ for this system. So, we set the delay-time for the Lorenz model to be 10 [Aba95].

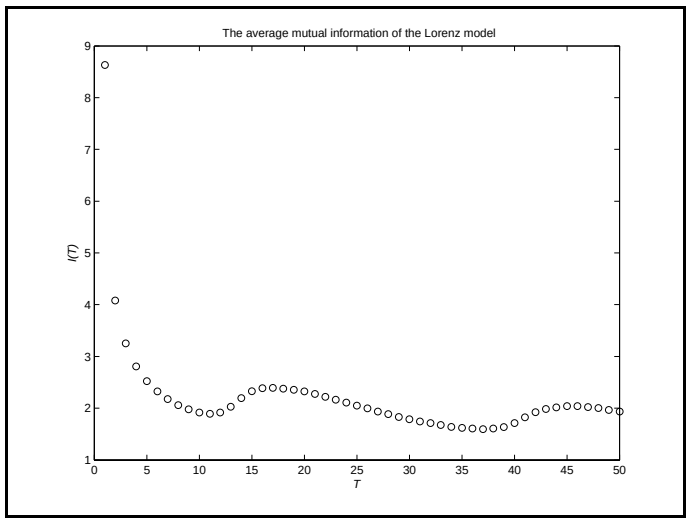



Figure 6.1 The Lorenz model average mutual information

In the next section, we present full detail of two algorithms proposed by Abarbanel *et al* in 1992 and 1993 to estimate $d_L$ which are based on the CDD and the CDT methods. After that, we present in Section 6.4 a discussion regarding some limitations of the previous two algorithms and suggest some approaches to overcome these limitations.

## 6.3 Two algorithms that use the local neighbor search

Abarbanel *et al* [KBA92, AbKe93] proposed two algorithms based on the CDD and the CDT geometric methods to estimate $d_L$ of a chaotic system. In the next section, we present the first algorithm which is based on the CDD method while in Section 6.3.2 we present the second algorithm which is based on the CDT method.

### 6.3.1 The $\mathrm{CDD}_L$ Algorithm

Abarbanel *et al* [KBA92, AbKe93] proposed an algorithm based on the CDD geometric method to estimate $d_L$ of a chaotic system. In a typical experiment, all we can see is a set of scalar measurements $y(m)$, where $m = 1, 2, \ldots, N$. The $\mathrm{CDD}_L$ algorithm starts by computing the theoretical minimum embedding dimension $d_E$ which is $d_E = 2d_c + 1$

where $d_c$ is the box-counting dimension of the original system. Notice that if $d_c$ is not known, the algorithm can start with an arbitrary large value for $d_E$. The delay-vectors $\mathbf{y}_{d_E}(m)$ are constructed from $y(m)$ as follows:

$$\mathbf{y}_{d_E}(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d_E-1)T)\right]^t, \tag{6.4}$$

where $T$ is the delay-time, $m = 1, 2, \ldots, M$ and $M = N-(d_E-1)T$. In the space $\Re^{d_E}$, the distance between $\mathbf{y}_{d_E}(m) \in \Re^{d_E}$ and every other point in this space is computed. From the $M-1$ computed distances, the $N_b$ neighbors with the shortest distances to $\mathbf{y}_{d_E}(m)$ are chosen. The value of $N_b$ is chosen to be between 10 and 100. The $N_b$ nearest neighbors of $\mathbf{y}_{d_E}(m)$ are saved in the matrix $\boldsymbol{Y}_{d_E}^{N_b}(m)$ as follows

$$\boldsymbol{Y}_{d_E}^{N_b}(m) = \left[\mathbf{y}_{d_E}(i_{d_E}^1(m))\ \mathbf{y}_{d_E}(i_{d_E}^2(m))\ \ldots\ \mathbf{y}_{d_E}(i_{d_E}^{N_b}(m))\right], \tag{6.5}$$

where $i_{d_E}^k(m)$ is the index of the $k^{th}$ neighbor of $\mathbf{y}_{d_E}(m)$ and $k = 1, 2, \ldots, N_b$. The $N_b$ indices of these neighbors are saved as a function of $m$ as follows

$${}^m\mathbf{i}_{d_E} = \left[i_{d_E}^1(m)\ i_{d_E}^2(m)\ \ldots\ i_{d_E}^{N_b}(m)\right], \tag{6.6}$$

where ${}^m\mathbf{i}_{d_E}$ is a row vector (see Section 5.2.1). The next example illustrates this idea.

6.3.1.1 Example: Neighbor indices

Let the reference time be $m = 2$, $N_b = 3$, and the space dimension be $d_E = 5$, then, if the 3 nearest neighbors of $\mathbf{y}_{d_E}(m)$ are $\boldsymbol{Y}_5^3 = \left[\mathbf{y}_5(63)\ \mathbf{y}_5(105)\ \mathbf{y}_5(10)\right]$, then ${}^m\mathbf{i}_{d_E} = {}^2\mathbf{i}_5 = \left[i_5^1(2)\ i_5^2(2)\ i_5^3(2)\right] = \left[63\ 105\ 10\right]$.

The next step for the $\mathrm{CDD}_L$ algorithm is to project the point $\mathbf{y}_{d_E}(m)$ and the matrix $\boldsymbol{Y}_{d_E}^{N_b}$ into the space $\Re^d$ to produce the point $\mathbf{y}_d(m)$ and the matrix $\boldsymbol{Y}_d^{N_b}$ respectively where

$d = 1, 2, \ldots, d_E - 1$. The projection from the space $\Re^{d_E}$ into the space $\Re^d$ is done by the vector-coordinates projection method which is explained next.

6.3.1.2 The vector-coordinates projection method

The delay-vector $\mathbf{y}_{d_E}(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d_E-1)T)\right]^t$ can be projected into the space $\Re^d$ by choosing the first $d$ coordinates of $\mathbf{y}_{d_E}(m)$ where $d \le d_E$. That means $\mathbf{y}_d(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d-1)T)\right]^t$.

The algorithm projects the point $\mathbf{y}_{d_E}(m)$ and the matrix $\boldsymbol{Y}_{d_E}^{N_b}$ into $\Re^d$ to find the point $\mathbf{y}_d(m)$ and the matrix $\boldsymbol{Y}_d^{N_b}$ respectively. The matrix $\boldsymbol{Y}_d^{N_b}$ contains the $N_b$ neighbors of $\mathbf{y}_d(m)$:

$$\boldsymbol{Y}_d^{N_b}(m) = \left[\mathbf{y}_d(i_{d_E}^1(m))\ \mathbf{y}_d(i_{d_E}^2(m))\ \ldots\ \mathbf{y}_d(i_{d_E}^{N_b}(m))\right]. \tag{6.7}$$

Notice that the indices $i_{d_E}^k(m)$ of the $N_b$ neighbors in $\Re^d$ are found in the space $\Re^{d_E}$. This gives a drawback to algorithms that use the local neighbor search method, as will be explained in Section 6.4. After finding the $N_b$ neighbors of $\mathbf{y}_d(m)$ (columns of the matrix $\boldsymbol{Y}_d^{N_b}(m)$), the $\mathrm{CDD}_L$ algorithm computes the distance between $\mathbf{y}_d(m)$ and its $N_b$ neighbors. The $N_b$ computed distances are saved in the vector

$$\mathbf{q}_d(m) = \left[\left\|\mathbf{y}_d(m) - \mathbf{y}_d(i_{d_E}^1(m))\right\|\ \left\|\mathbf{y}_d(m) - \mathbf{y}_d(i_{d_E}^2(m))\right\|\ \ldots\ \left\|\mathbf{y}_d(m) - \mathbf{y}_d(i_{d_E}^{N_b}(m))\right\|\right]. \tag{6.8}$$

The nearest neighbor of $\mathbf{y}_d(m)$ is the point with the smallest distance. The algorithm saves the minimum distance from $\mathbf{q}_d(m)$ in $r_d(m)$. The index of the nearest neighbor is labeled by *n*, as illustrated in the next example.

6.3.1.3 Example: Vector-coordinates projection

From Example 6.3.1.1, the distances between $\mathbf{y}_5(2)$ and its 3 neighbors are $\mathbf{q}_5(2) = \left[\left\|\mathbf{y}_5(2) - \mathbf{y}_5(63)\right\|\ \left\|\mathbf{y}_5(2) - \mathbf{y}_5(105)\right\|\ \left\|\mathbf{y}_5(2) - \mathbf{y}_5(10)\right\|\right]$. Assuming that

$\mathbf{q}_5(2) = \begin{bmatrix} 1.2 & 0.9 & 3.7 \end{bmatrix}$, the minimum distance is $r_5(2) = min(\mathbf{q}_5(2))$, which has the value $0.9$. So the nearest neighbor of $\mathbf{y}_5(2)$ is the point with the index $n = 105$. That means the nearest neighbor of $\mathbf{y}_5(2)$ is $\mathbf{y}_5(105)$.

It is noteworthy that the search for the nearest neighbor of $\mathbf{y}_d(m)$ is done locally among the $N_b$ neighbors only, and this is how the algorithm got its name $\text{CDD}_L$; the CDD method with a local neighbor search. After finding *n*, which is the index of the nearest neighbor of $\mathbf{y}_d(m)$, the $\text{CDD}_L$ algorithm runs a test to find if $\mathbf{y}_d(n)$ is a false nearest neighbor (FNN) of $\mathbf{y}_d(m)$, the reference point. To do that, the algorithm projects both points $\mathbf{y}_d(m)$ and $\mathbf{y}_d(n)$ into the space $\Re^{d+1}$ by the vector-coordinates projection method. If these two neighbors are true neighbors, their distance will not change significantly as the dimension of the space increases to $d + 1$. The distance between $\mathbf{y}_{d+1}(m)$ and $\mathbf{y}_{d+1}(n)$ is labeled by $e_{d+1}(m) = \left\| \mathbf{y}_{d+1}(m) - \mathbf{y}_{d+1}(n) \right\|$. Now, if $\mathbf{y}_d(n)$ is a FNN of $\mathbf{y}_d(m)$, the ratio

$$\frac{e_{d+1}(m) - r_d(m)}{r_d(m)} > \rho, \tag{6.9}$$

where the threshold is $1 < \rho < 10$. The above steps are repeated for all time points and the percentage of FNNs is recorded as a function of *d.* As *d* increases, the percentage of FNNs decreases. At one point, further increase of *d* does not change the percentage of FNNs. The $\text{CDD}_L$ algorithm repeats all the above steps for different sizes of neighborhood ($N_b$). The minimum embedding dimension $d_L$ is the dimension where the percentage of FNNs is independent of both the change in $N_b$ and in the increase of *d*. Figures 6.2 shows a pseudocode that summarizes the $\text{CDD}_L$ algorithm. In Chapter 7, we apply the $\text{CDD}_L$ algorithm to different examples of chaotic systems and show the estimated $d_L$ resulting from the algorithm.

{The $\text{CDD}_L$ algorithm
•*Choose* a value for the threshold $1 < \rho < 10$
•*Compute* the delay-time $T$ from the first minimum of the average mutual information, see Section 6.2
•*Compute* the theoretical minimum embedding dimension $d_E = 2d_c + 1$. If $d_c$ is not known choose an arbitrary large number for $d_E$
•*Initialize* the vector $\mathbf{k}_{fnn}$ of dimension $d_E - 1$ x 1
•*Create* the delay-vectors $\mathbf{y}_{d_E}(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d_E-1)T)\right]^t$, where $m = 1, 2, \ldots, M$
•*Compute* the distance between $\mathbf{y}_{d_E}(m)$ and every other point in $\Re^{d_E}$ and save them in the vector $\mathbf{q}_{d_E}(m)$.
•*Set* $10 \le N_b \le 100$ (the neighborhood size)
•*Find* the $N_b$ shortest distances in $\mathbf{q}_{d_E}(m)$ and save their indices in $i^k_{d_E}(m)$ where $k = 1, 2, \ldots, N_b$
•*Save* the $N_b$ neighbors of $\mathbf{y}_{d_E}(m)$ in the matrix $\boldsymbol{Y}^{N_b}_{d_E}(m) = \left[\mathbf{y}_{d_E}(i^1_{d_E}(m))\ \mathbf{y}_{d_E}(i^2_{d_E}(m))\ \ldots\ \mathbf{y}_{d_E}\left(i^{N_b}_{d_E}(m)\right)\right]$
•*for* $d = 1, 2, \ldots, d_E - 1$ (Vector dimension)
    •*for* $m = 1, 2, \ldots, M$ (Index)
        (Project the vectors in $\Re^{d_E}$ into $\Re^d$ by the vector-coordinates method, see Section 6.3.1.2.)
        •*Project* $\mathbf{y}_{d_E}(m)$ and $\boldsymbol{Y}^{N_b}_{d_E}(m)$ into $\Re^d$ to find $\mathbf{y}_d(m)$ and $\boldsymbol{Y}^{N_b}_{d}(m)$ respectively
        •*Compute* the distances between $\mathbf{y}_d(m)$ and the columns of $\boldsymbol{Y}^{N_b}_{d}(m)$ and save them in $\mathbf{q}_d$
        •*Find* the minimum distance in $\mathbf{q}_d$ and save it as $r_d$
        •*Set* the index of the point with the minimum distance in $\mathbf{q}_d$ as *n*, see Example 6.3.1.3
        •*Project* $\mathbf{y}_{d_E}(m)$ and $\mathbf{y}_{d_E}(n)$ into $\Re^{d+1}$ to find $\mathbf{y}_{d+1}(m)$ and $\mathbf{y}_{d+1}(n)$ respectively
        •*Compute* $e_{d+1} = \left\|\mathbf{y}_{d+1}(m) - \mathbf{y}_{d+1}(n)\right\|$.
        •*If* $(e_{d+1} - r_d)/r_d > \rho$
            $k_{fnn}(d) = k_{fnn}(d) + 1$
        *end if*
    *end m*
*end d* }

Figure 6.2 Pseudocode of the $\text{CDD}_L$ algorithm

### 6.3.2 The $\text{CDT}_L$ Algorithm

After presenting the first algorithm that uses the local neighbor search ($\text{CDD}_L$), now we present the second algorithm that also uses the local neighbor search which is the $\text{CDT}_L$. Abarbanel *et al* [AbKe93] proposed an algorithm based on the CDT geometric method to estimate $d_L$ of a chaotic system. As in the $\text{CDD}_L$ algorithm, the $\text{CDT}_L$ algorithm builds the delay-vectors $\mathbf{y}_{d_E}(m)$ from the measurements $y(m)$ as shown in Equation

(6.4). Then the $\text{CDT}_L$ computes the $N_b$ neighbors of $\mathbf{y}_{d_E}(m)$, where $10 \le N_b \le 100$. The algorithm now saves the $N_b$ computed neighbors of $\mathbf{y}_{d_E}(m)$ in the matrix $\boldsymbol{Y}_{d_E}^{N_b}(m)$. It also saves the indices of these neighbors as $i_{d_E}^{k}(m)$ where $k = 1, 2, \ldots, N_b$. In the next step, the algorithm projects the reference point $\mathbf{y}_{d_E}(m)$ and the matrix $\boldsymbol{Y}_{d_E}^{N_b}(m)$ into $\Re^d$ to find $\mathbf{y}_d(m)$ and $\boldsymbol{Y}_d^{N_b}(m)$ respectively where $d = 1, 2, \ldots, d_E$. The projection method used by the $\text{CDT}_L$ algorithm is different from that used by the $\text{CDD}_L$ algorithm. The $\text{CDT}_L$ uses the Principal Components Analysis (PCA) projection method while the $\text{CDD}_L$ uses the vector-coordinates projection method. The PCA projection method is explained below.

6.3.2.1 The PCA projection method

The PCA projection method starts by computing the covariance matrix at time *m* using the reference point $\mathbf{y}_{d_E}(m)$ and its $N_b$ neighbors as follows:

$$\mathbf{C}(m) = \frac{1}{N_b} \sum_{k=1}^{N_b} [\mathbf{y}_{d_E}(i_{d_E}^{k}(m)) - \mathbf{y}_{d_E}(m)][\mathbf{y}_{d_E}(i_{d_E}^{k}(m)) - \mathbf{y}_{d_E}(m)]^t. \qquad (6.10)$$

The next step in the PCA projection method is to compute the eigen values $(\lambda_1, \lambda_2, \ldots, \lambda_{d_E})$ and the eigen vectors $(\underline{\eta}_1, \underline{\eta}_2, \ldots, \underline{\eta}_{d_E})$ of the matrix $\mathbf{C}(m)$. The $d_E$ eigen values are arranged such that $\lambda_1 \ge \lambda_2 \ge \ldots \ge \lambda_{d_E}$. The corresponding eigen vectors are used to build the basis matrix $\mathbf{B}_{d_E} = \left[\underline{\eta}_1 \ \underline{\eta}_2 \ \ldots \ \underline{\eta}_{d_E}\right]$. The projection from $\Re^{d_E}$ into $\Re^d$ is done by choosing the first *d* columns of the matrix $\mathbf{B}_{d_E}$ where $d \le d_E$. The projection is found by the equation $\mathbf{y}_d^p(m) = \left[\underline{\eta}_1 \ \underline{\eta}_2 \ \ldots \ \underline{\eta}_d\right]^t \mathbf{y}_{d_E}(m)$. (The superscript *p* in $\mathbf{y}_d^p$ is used to emphasize that the projection method from $\Re^{d_E}$ into $\Re^d$ is the PCA.)

After projecting the points from $\Re^{d_E}$ into $\Re^d$, the distances between $\mathbf{y}_d^p(m)$ and its $N_b$ neighbors (columns of the matrix $\boldsymbol{Y}_d^{N_b, p}(m)$) are computed and saved in the vector $\mathbf{q}_d(m)$ as shown in Equation (6.8). It has been noted that the search for the nearest neighbor

of $\mathbf{y}_d^p(m)$ is done locally among the $N_b$ neighbors only, and this is how the algorithm got its name $\text{CDT}_L$, the CDT method with a local neighbor search. The index of the nearest neighbor of $\mathbf{y}_d^p(m)$ is labeled by *n*. After identifying the two neighbors in $\Re^d$, the $\text{CDT}_L$ algorithm runs a test to see whether these two neighbors are FNNs or not. In any chaotic system, distances between points change exponentially fast (as explained in detail in Chapter 8). However if two points are FNNs, the distance between them would increase faster than if they were true neighbors (see Section 4.3.1.3). The $\text{CDT}_L$ algorithm takes advantage of the increase in distances to check the existence of FNNs. The algorithm labels $\mathbf{y}_d^p(n) = \mathbf{y}_d^p(i_d^1(m))$ (the nearest neighbor of $\mathbf{y}_d^p(m)$) as a FNN if the distance between $\mathbf{y}_{d_E}(n)$ and $\mathbf{y}_{d_E}(m)$ in $\Re^{d_E}$ reaches a predefined threshold before a fixed number of time steps. The threshold used by this algorithm is a fraction of the attractor mean. The attractor mean is

$$\beta = \frac{1}{M}\sum_{k=1}^{M}\left\|\mathbf{y}_{d_E}(k) - \bar{\mathbf{y}}_{d_E}\right\|. \tag{6.11}$$

The threshold used by the $\text{CDT}_L$ algorithm is $\zeta\beta$ where $0 < \zeta < 1$. The $\text{CDT}_L$ algorithm computes the distance between $\mathbf{y}_{d_E}(m)$ and $\mathbf{y}_{d_E}(n) = \mathbf{y}_{d_E}(i_d^1(m))$ as time increases up to a maximum of *T* time-steps, (*T* is the delay-time where the first minimum of the average mutual information occurs, see Section 6.2). If the distance reaches the threshold $\zeta\beta$ before *T* time-steps, it labels $\mathbf{y}_d^p(i_d^1(m))$ as a FNN.

These steps are repeated for all data points and the percentage of the FNNs is recorded as a function of *d*. As *d* increases, the percentage of the FNNs decreases. At one point, further increase of *d* does not reduce the percentage of the FNNs. The above steps are repeated for different value of $N_b$, where $10 \le N_b \le 100$. The minimum embedding di-

mension $d_L$ of the system is found from the dimension where the percentage of the FNNs becomes independent of both the increase of $d$ and $N_b$. In Chapter 7, we demonstrate the $\mathrm{CDT}_L$ algorithm on different examples of chaotic systems.

In Figure 6.3, we show the pseudocode that summarizes the $\mathrm{CDT}_L$ algorithm. It shows that the algorithm uses a *while* loop to check for the FNNs. The first parameter of the loop is $\Delta$ which is the number of time steps used to check the existence of FNNs. The second parameter is $\sigma_\Delta$ which is the distance between the two neighbors after $\Delta$ time steps ahead. If the end of the data set was reached with $\Delta < T$, and the error criteria ($\sigma_\Delta \geq \zeta\beta$) was not satisfied, we can not have any conclusion about $\mathbf{y}_d^p(n)$ being either a FNN of $\mathbf{y}_d^p(m)$ or not. So, we have to discard it from our consideration. That means, we don't have enough data to check this nearest neighbor. For instance, let the index of the nearest neighbor be $n = 10$, the reference index be $m = 5$, the total number of points be $M = 50$, and the delay-time be $T = 41$. If $\Delta$ reached the end of the data set while the distance between the two points is still less than the predefined threshold, $\Delta = 50 - max(10, 5) = 40$, then $\Delta = 40 < T = 41$. We don't know if $\mathbf{y}_{d_E}(10)$ is a FNN of $\mathbf{y}_{d_E}(5)$ or not, since in the next time step (if we imagine that we can reach it) $\Delta = 41$, the point $\mathbf{y}_{d_E}(10)$ will be a FNN of $\mathbf{y}_{d_E}(5)$ if the distance $\sigma_\Delta \geq \zeta\beta$. On the other hand, $\mathbf{y}_{d_E}(10)$ will be a true nearest neighbor of $\mathbf{y}_{d_E}(5)$ if $\sigma_\Delta < \zeta\beta$.

{The $\text{CDT}_L$ algorithm

•*Compute* the delay-time $T$ from the first minimum of the average mutual information, see Section 6.2

•*Compute* $d_E = 2d_c + 1$. If $d_c$ is not known choose an arbitrary large number for $d_E$

•*Initialize* the vector $\mathbf{p}_K$ of dimension $d_E$ x 1 to hold the count of FNN

•*Create* the delay-vectors $\mathbf{y}_{d_E}(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d_E-1)T)\right]^t$, where $m = 1, 2, \ldots, M$

•*Compute* the distance between $\mathbf{y}_{d_E}(m)$ and every other point in $\Re^{d_E}$ and save them in the vector $\mathbf{q}_{d_E}(m)$.

•*Set* $10 \le N_b \le 100$ (the neighborhood size)

•*Find* the $N_b$ shortest distances in $\mathbf{q}_{d_E}(m)$ and save the vectors indices as $i^k_{d_E}(m)$ where $k = 1, 2, \ldots, N_b$

•*Save* the $N_b$ neighbors of $\mathbf{y}_{d_E}(m)$ in the matrix $\boldsymbol{Y}^{N_b}_{d_E}(m) = \left[\mathbf{y}_{d_E}(i^1_{d_E}(m))\ \mathbf{y}_{d_E}(i^2_{d_E}(m))\ \ldots\ \mathbf{y}_{d_E}\left(i^{N_b}_{d_E}(m)\right)\right]$

•*Choose* the ratio $0 < \zeta < 1$

•*Compute* $\bar{\mathbf{y}}_{d_E} = \frac{1}{M} \sum_{k = 1, 2, \ldots, M} \mathbf{y}_{d_E}(k)$ (average vector in the space $\Re^{d_E}$)

•*Compute* $\beta = \frac{1}{M} \sum_{k = 1, 2, \ldots, M} \left\|\mathbf{y}_{d_E}(k) - \bar{\mathbf{y}}_{d_E}\right\|$ ($\zeta\beta$ is the threshold for false neighbors, $\left\|\bar{\mathbf{y}}_{d_E}\right\|$: mean distance from the origin.)

•*for* $d = 1, 2, \ldots, d_E$ (Vector dimension)

  •*for* $m = 1, 2, \ldots, M$ (index)

    •*Compute* the covariance matrix $\mathbf{C} = \frac{1}{N_b} \sum_{k = 1, 2, \ldots, N_b} [\mathbf{y}_{d_E}(m) - \mathbf{y}_{d_E}(i^k_{d_E}(m))][\mathbf{y}_{d_E}(m) - \mathbf{y}_{d_E}(i^k_{d_E}(m))]^t$

    •*Compute* eigen values $\lambda_i$ and the eigen vectors $\underline{\eta}_i$ of $\mathbf{C}$.

    •*Arrange* the eigen values such that $\lambda_1 \ge \lambda_2 \ge \ldots \ge \lambda_{d_E}$

    •From the corresponding eigen vectors, build the basis matrix $\mathbf{B}_d = \left[\underline{\eta}_1\ \underline{\eta}_2\ \ldots\ \underline{\eta}_d\right]$

    •$\mathbf{y}^p_d(m) = \mathbf{B}^t_d\, \mathbf{y}_{d_E}(m)$ and $\boldsymbol{Y}^{N_b, p}_d(m) = \mathbf{B}^t_d\, \boldsymbol{Y}^{N_b}_{d_E}(m)$; $p$ for a vector found by the PCA

    •*Compute* the distance between $\mathbf{y}^p_d(m)$ and the columns of $\boldsymbol{Y}^{N_b, p}_d$ and save their distances in $\mathbf{q}_d$.

    •*Find* the minimum value of $\mathbf{q}_d$ and label the index of the vector that produced it by *n*.

    •Set $\sigma_\Delta = 0$, $\Delta = 1$ ($\sigma_\Delta$: distance, $\Delta$: time step.)

    •*while* $(\sigma_\Delta < \zeta\beta)$ AND $(\Delta \le M - max(m, n))$ AND $(\Delta \le T)$.

      • $\sigma_\Delta = \left\|\mathbf{y}_{d_E}(m + \Delta) - \mathbf{y}_{d_E}(n + \Delta)\right\|$

      • $\Delta = \Delta + 1$

    *end while*

    (If the end of the data set was reached without a conclusion, discard $\mathbf{y}_d(n)$.)

    •*if* $\Delta = (M - max(m, n))$ AND $(\Delta < T)$ AND $(\sigma_\Delta < \zeta\beta)$

      •*Label* $\mathbf{y}^p_d(n)$ as *Not decidable*

    •*elseif* $(\Delta \le T)$ AND $(\sigma_\Delta \ge \zeta\beta)$

      • $p_K(d) = p_K(d) + 1$

    *end if*

  *end m*

*end d*}

Figure 6.3 Pseudocode of the $\text{CDT}_L$ algorithm

After presenting the $\text{CDD}_L$ and $\text{CDT}_L$ algorithms for estimating the minimum embedding dimension of a chaotic system, we discuss in the next section some limitations of these two algorithms and suggest different approaches to overcome their limitations.

## 6.4 Limitations of the $\mathrm{CDD}_L$ and the $\mathrm{CDT}_L$ Algorithms

In Section 6.3, we found that both the $\mathrm{CDD}_L$ and the $\mathrm{CDT}_L$ algorithms use the local neighbor search method to find the nearest neighbor of the reference point. That means the search for the nearest neighbor of $\mathbf{y}_d(m)$ is done locally within the $N_b$ neighbors only, rather than within the whole data set. The local neighbor search was used to reduce the computational cost. However, in practice, it was observed that algorithms that use the local neighbor search often do not estimate the correct minimum embedding dimensions $d_L$ for systems with dimensions larger than three. In addition, poor estimates of $d_L$ could occur even for systems with dimension three as in the case of the Lorenz model using the $\mathrm{CDT}_L$ (see Table 7.1 in the next chapter). The $\mathrm{CDD}_L$ and $\mathrm{CDT}_L$ algorithms can also produce poor estimates of $d_L$ for noisy signals (see Table 7.1).

Limiting the search for the nearest neighbor of the reference point $\mathbf{y}_d(m) \in \Re^d$ to be within the $N_b$ projected neighbors, that were originally found in $\Re^{d_E}$, does not always find the actual nearest neighbor of $\mathbf{y}_d(m)$. This happens because the $N_b$ projected neighbors could actually be scattered in the attractor in the space $\Re^d$ and not close to $\mathbf{y}_d(m)$. When the dimension of the original system is small (3 or less), this problem may not be significant. However, as the dimension of the system increases, the effect will become pronounced. To improve the search for the nearest neighbor, we can find the nearest neighbor of $\mathbf{y}_d(m)$ within the whole attractor in the space $\Re^d$, then test this neighbor in the space $\Re^{d+1}$ to see if it is a FNN.

Further, when noise exists in the signal, the reconstructed attractor is blurred. Hence, the $N_b$ computed neighbors of $\mathbf{y}_{d_E}(m)$ may not actually be its closest neighbors.

When these $N_b$ neighbors are projected into $\Re^d$, they may look random in this space. As a result, the attractor information is lost, and the correct value of $d_L$ can't be found.

Another observation that can be obtained from the $\text{CDD}_L$ and $\text{CDT}_L$ algorithms is that they use two different projection methods: the vector-coordinate method and the PCA method, respectively. The PCA projection method has the advantage that the search for the nearest neighbor is done along the major variations of the signal. However, the vector-co-ordinate projection method is less expensive in term of its computational cost.

The above observations suggest that we should search for the nearest neighbor of the reference point among the whole data set rather than among the $N_b$ projected neighbors only. We call the search method that uses the whole data set the global neighbor search method. Practically, it was observed that when the global neighbor search method was used, the estimate of $d_L$ for a chaotic system with dimension greater than three is improved, as we will show in Chapter 7. In addition, the global neighbor search method was found to be more robust to noise than the local neighbor search method. (The local neighbor search method was mainly used to reduce the computational cost. Recently, a fast neighbor search algorithm has been introduced that provides significant reduction in computations [MPL00].)

In the next two sections, we present four new algorithms. In Section 6.5, we present the first three algorithms which apply the global neighbor search method to the CND, CDD, and CDT geometric methods. In Section 6.6, we present the fourth algorithm, which uses the predictive technique. The $\text{CDD}_G$ and $\text{CDT}_G$ algorithms are variations of algorithms presented by Abarbanel *et al*. These algorithms use global neighbor search, rather than a

local neighbor search. The CND and the predictive algorithms were completely developed as part of this research.

## 6.5 Three new algorithms that use the global neighbor search method

### 6.5.1 The first new algorithm: The CND

In Section 4.3.1.1, we presented the CND geometric method which is used to estimate $d_L$ of a chaotic system. We also presented the application of the CND method to estimate $d_L$ of the Henon map in Section 5.2.1. In this section, we present the first new algorithm which applies the global neighbor search to the CND method. As in the previous two algorithms, the CND algorithm computes the dimension $d_E$ and constructs the delay-vectors $\mathbf{y}_{d_E}(m)$ as explained in Section 6.3.1. The matrix $\boldsymbol{Y}_{d_E}^{M}$ that contains all the points in the space $\Re^{d_E}$ is constructed from the delay-vectors:

$$\boldsymbol{Y}_{d_E}^{M} = \left[\mathbf{y}_{d_E}(1)\ \mathbf{y}_{d_E}(2)\ \ldots\ \mathbf{y}_{d_E}(M)\right]. \tag{6.12}$$

The points in the space $\Re^{d_E}$ (columns of $\boldsymbol{Y}_{d_E}^{M}$) have to be projected into the space $\Re^{d}$ by the vector-coordinates projection method where $d = 1, 2, \ldots, d_E - 1$. (The use of the PCA projection method does not change the results significantly in this algorithm.) In the space $\Re^{d}$, the CND algorithm needs to find $\mathbf{y}_d(n)$ which is the nearest neighbor of the reference point $\mathbf{y}_d(m)$. To do that, it computes the distances between $\mathbf{y}_d(m)$ and every other point in the space $\Re^{d}$ and labels the index of the point with the shortest distance to $\mathbf{y}_d(m)$ by $n$. In the next step, the algorithm checks to see if $\mathbf{y}_d(n)$ is a FNN of $\mathbf{y}_d(m)$ by increasing the dimension of the space to $d + 1$ using the same projection method mentioned above. In the space $\Re^{d+1}$, the CND algorithm computes the first $w$ neighbors of $\mathbf{y}_{d+1}(m)$. (This is a new geometric algorithms. Abarbanel used only the nearest neighbor.) It saves their $w$ in-

dices as ${}^{m}\mathbf{i}_{d+1}$. If $\mathbf{y}_d(n)$ is a true neighbor of $\mathbf{y}_d(m)$, it should remain close as the dimension increases to $d + 1$. The CND algorithm now checks to see if *n* appears as an element of the vector ${}^{m}\mathbf{i}_{d+1}$. In other words, it checks to see if $\mathbf{y}_{d+1}(n)$ appears as one of the *w* neighbors of $\mathbf{y}_{d+1}(m)$. If *n* does not appear as an element of ${}^{m}\mathbf{i}_{d+1}$, the algorithm labels $\mathbf{y}_d(n)$ as a FNN of $\mathbf{y}_d(m)$. These steps are repeated until the last point of the data set, and the percentage of the FNNs is recorded as a function of *d*. As *d* increases, the percentage of the FNNs decreases. At one point, further increase of *d* does not change the percentage of the FNNs significantly.

To estimate $d_L$, the algorithm checks to see if the change in the percentage of the FNNs is less than some predefined threshold ($\alpha$) for five consecutive dimensions. That means if

$$\frac{\left|\mathrm{FNN}_{d+1} - \mathrm{FNN}_d\right|}{M} < \alpha \cdot 100, \tag{6.13}$$

for five consecutive dimensions, then it sets $d_L = d - 5$, where $\mathrm{FNN}_d$ is the number of FNN found at dimension *d*. That means, the change of the percentage of the FNNs is insignificant. The threshold value is chosen to be $0 < \alpha < 3$. If the signal is noise free, we set $\alpha = 0$, while if the signal is noisy, we increase $\alpha$ up to 3. This is since the existence of noise affects the percentage of FNNs, which causes the plot of FNNs with respect to *d* to be uneven. The above steps are repeated for different values of *w* and the estimated $d_L$ is plotted as a function of *w*. At one point, further increase of *w* does not change the estimated $d_L$. This is where the algorithm finds $d_L$. The pseudocode shown in the next figure summarizes the CND algorithm. In Chapter 7, we apply the CND algorithm to different exam-

ples of chaotic systems and show the estimated $d_L$ resulting from the algorithm.

{The CND algorithm
•*Compute* the delay-time *T* from the first minimum of the average mutual information, see Section 6.2
•*Compute* $d_E = 2d_c + 1$. If $d_c$ is not known choose an arbitrary large number for $d_E$
•*Set* $w_{max}$, and the threshold $0 < \alpha < 3$ and initialize *w* to 1
•*Initialize* a vector $\mathbf{k}_{fnn}$ of dimension $d_E - 1$ x 1 and a vector $\mathbf{d}_w$ of dimension $w_{max}$x1
•*Create* the delay-vectors $\mathbf{y}_{d_E}(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d_E-1)T)\right]^t$, where $m = 1, 2, \ldots, M$
•*Create* the matrix $\mathbf{Y}_{d_E}^M = \left[\mathbf{y}_{d_E}(1)\ \mathbf{y}_{d_E}(2)\ \ldots\ \mathbf{y}_{d_E}(M)\right]$
•*while* $w \le w_{max}$ (search window size)
  •*for* $d = 1, 2, \ldots, d_E - 1$
    •*Project* $\mathbf{Y}_{d_E}^M$ into $\Re^d$ to find $\mathbf{Y}_d^M$ (use the vector coordinates projection method)
    •*Project* $\mathbf{Y}_{d_E}^M$ into $\Re^{d+1}$ to find $\mathbf{Y}_{d+1}^M$ (use the vector coordinates projection method)
    •*for* $m = 1, 2, \ldots, M$
      •*Compute* the distances between $\mathbf{y}_d(m)$ and the columns of $\mathbf{Y}_d^M$ and save them in $\mathbf{q}_d$.
      •*Find* the point that produced the minimum value in $\mathbf{q}_d$ and label its index by *n*.
      •*Compute* the distances between $\mathbf{y}_{d+1}(m)$ and the columns of $\mathbf{Y}_{d+1}^M$ and save the distances in $\mathbf{q}_{d+1}$.
      •*Find* the $1^{st}$, $2^{nd}$, …, and $w^{th}$ minimum values of $\mathbf{q}_{d+1}$ and *label* their indices by ${}^m\mathbf{i}_{d+1}$. (Compare *n* to the elements of ${}^m\mathbf{i}_{d+1}$)
      •*if* $n \notin {}^m\mathbf{i}_{d+1}$
        $k_{fnn}(d) = k_{fnn}(d) + 1$ (if *n* is not an element of ${}^m\mathbf{i}_{d+1}$, label $\mathbf{y}_d(n)$ as a FNN)
      *end if*
    *end m*
    (look for a flat region in the curve of the percentage of FNNs with respect to *d*)
    •*if* $d > 5$ AND $\left|k_{fnn}(d-2) - k_{fnn}(d-1)\right| / M \text{x} 100 < \alpha$ (for 5 consecrative times)
      • $d_L = d - 5$
      •*Break* the *d* loop
    *end if*
  *end d*
  • $d_w(w) = d_L$
  • $w = w + 1$ (increase the search window size)
*end w*}

Figure 6.4 Pseudocode of the CND algorithm

### 6.5.2 The second new algorithm: The $\text{CDD}_G$

In Section 4.3.1.2, we presented the CDD geometric method which is used to estimate $d_L$ of a chaotic system. We also presented the application of the CDD method to estimate $d_L$ of the Henon map in Section 5.2.2. In Section 6.3.1, we presented an algorithm based on the CDD method with a local neighbor search ($\text{CDD}_L$). Some limitations of the $\text{CDD}_L$ algorithm were presented in the beginning of this section. To improve the performance of the $\text{CDD}_L$ algorithm, we present the new algorithm: the $\text{CDD}_G$ that can overcome the limitations of the $\text{CDD}_L$.

The $\text{CDD}_G$ applies the global neighbor search to the CDD geometric method. As in the previous algorithms, the $\text{CDD}_G$ algorithm computes the theoretical minimum embedding dimension $d_E$ and constructs the delay-vectors $\mathbf{y}_{d_E}(m)$, as described in Section 6.3.1. In addition, the matrix $\boldsymbol{Y}_{d_E}^M$ has to be built as shown in Equation (6.12). The algorithm projects the matrix $\boldsymbol{Y}_{d_E}^M$ into the space $\Re^d$ by the PCA projection method where $d = 1, 2, \ldots, d_E - 1$ (see Section 6.3.2.1 for more detail about the PCA projection method). In the space $\Re^d$, the $\text{CDD}_G$ algorithm computes the distance between the reference point $\mathbf{y}_d^p(m)$ and every other point in the space $\Re^d$. (Notice that the $\text{CDD}_L$ algorithm computes the $N_b$ neighbors of $\mathbf{y}_d(m)$ in the space $\Re^{d_E}$ then searches among the $N_b$ projected neighbors only for the nearest neighbor of $\mathbf{y}_d(m)$.) The computed distances are saved in the vector $\mathbf{q}_d(m)$. The algorithm now finds the minimum distance in $\mathbf{q}_d(m)$ and saves it as $r_d(m)$. It also labels the index of the point that produced the minimum distance as *n*. After identifying the nearest neighbor $\mathbf{y}_d^p(n) = \mathbf{y}_d^p(i_d^1(m))$ in the space $\Re^d$, the algorithm now needs to check if this neighbor is a FNN. To do that, it increases the dimension of the

space to $d+1$ by the same projection method mentioned above. In the space $\Re^{d+1}$, the distance between $\mathbf{y}_{d+1}^{p}(m)$ and $\mathbf{y}_{d+1}^{p}(n) = \mathbf{y}_{d+1}^{p}(i_d^1(m))$ is computed and saved as $e_{d+1}(m)$. As the dimension of the space increases, the distance between FNNs will increase as well. To apply this idea, the $\mathrm{CDD}_G$ algorithm labels $\mathbf{y}_d^p(n)$ as a FNN if

$$\frac{e_{d+1}(m) - r_d(m)}{r_d(m)} > \rho, \tag{6.14}$$

where the threshold is $1 < \rho < 10$. As the dimension of the space (*d*) increases, the percentage of the FNNs decreases. At one point, further increase of *d* does not improve the percentage of the FNNs significantly. The minimum dimension where this happens is the estimated $d_L$ of the system. In Chapter 7, we apply the $\mathrm{CDD}_G$ algorithm to different examples of chaotic systems and show the estimated $d_L$ resulting from the algorithm. Figure 6.5 shows the pseudocode that summarizes the $\mathrm{CDD}_G$ algorithm.

{The $\text{CDD}_G$ algorithm pseudocode

•*Compute* the delay-time $T$ from the first minimum of the average mutual information, see Section 6.2

•*Compute* $d_E = 2d_c + 1$. If $d_c$ is not known, choose an arbitrary large number for $d_E$

•*Set* the threshold $1 < \rho < 10$

•*Initialize* the vector $\mathbf{k}_{fnn}$ of dimension $d_E - 1$ x 1

•*Create* the delay-vectors $\mathbf{y}_{d_E}(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d_E-1)T)\right]^t$, where $m = 1, 2, \ldots, M$

•*Create* the matrix $\mathbf{Y}_{d_E}^M = \left[\mathbf{y}_{d_E}(1)\ \mathbf{y}_{d_E}(2)\ \ldots\ \mathbf{y}_{d_E}(M)\right]$

•*Compute* $\bar{\mathbf{y}}_{d_E} = \frac{1}{M}\sum_{k=1}^{M} \mathbf{y}_{d_E}(k)$ (average vector in the space $\Re^{d_E}$)

•*Compute* the covariance matrix $\mathbf{C} = \frac{1}{M}\sum_{k=1}^{M} [\mathbf{y}_{d_E}(k) - \bar{\mathbf{y}}_{d_E}][\mathbf{y}_{d_E}(k) - \bar{\mathbf{y}}_{d_E}]^t$

•*Compute* the eigen values $\lambda_i$ and eigen vectors $\underline{\eta}_i$ of $\mathbf{C}$, where $i = 1, 2, \ldots, d_E$

•*Arrange* the eigen values such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{d_E}$

•*Build* the basis matrix from the corresponding eigen vectors: $\mathbf{B}_{d_E} = \left[\underline{\eta}_1\ \underline{\eta}_2\ \ldots\ \underline{\eta}_{d_E}\right]$

•*for* $d = 1, 2, \ldots, d_E - 1$ (space dimension)

(Project the vectors in $\Re^{d_E}$ into $\Re^d$)

• $\mathbf{Y}_d^{M,p} = \left[\underline{\eta}_1\ \underline{\eta}_2\ \ldots\ \underline{\eta}_d\right]^t \mathbf{Y}_{d_E}^M$, the superscript $p$ indicates that the projection is done by the PCA

(Project the vectors in $\Re^{d_E}$ into $\Re^{d+1}$)

• $\mathbf{Y}_{d+1}^{M,p} = \left[\underline{\eta}_1\ \underline{\eta}_2\ \ldots\ \underline{\eta}_{d+1}\right]^t \mathbf{Y}_{d_E}^M$

•*for* $m = 1, 2, \ldots, M$ (Index)

•*Compute* the distance between $\mathbf{y}_d^p(m)$ and the columns of $\mathbf{Y}_d^{M,p}$ and save the distances as $\mathbf{q}_d$

•*Find* the minimum of $\mathbf{q}_d$ and save it as $r_d$

•*Label* the index of the point with the minimum distance by $n$

•*Compute* $e_{d+1} = \left\|\mathbf{y}_{d+1}^p(m) - \mathbf{y}_{d+1}^p(n)\right\|$ (from the columns of the matrix $\mathbf{Y}_{d+1}^{M,p}$)

•*If* $(e_{d+1}(m) - r_d(m))/r_d(m) > \rho$

• $k_{fnn}(d) = k_{fnn}(d) + 1$; increase the count of the FNN

•*end if*

*end m*

*end d*

}

Figure 6.5 Pseudocode of the $\text{CDD}_G$ algorithm

### 6.5.3 The third new algorithm: The $\mathrm{CDT}_G$

In Section 4.3.1.3, we presented the CDT geometric method which is used to estimate the minimum embedding dimension $d_L$ of a chaotic system. We also presented the application of the CDT method to estimate $d_L$ of the Henon map in Section 5.2.3. In Section 6.3.2, we showed the $\mathrm{CDT}_L$ algorithm which is based on the CDT method with a local neighbor search. Some limitations of the $\mathrm{CDT}_L$ algorithm were presented in the beginning of this section. To improve the performance of the $\mathrm{CDT}_L$ algorithm, we present a new algorithm known as the $\mathrm{CDT}_G$. This algorithm applies the global neighbor search to the CDT method

As in the previous algorithms, the $\mathrm{CDT}_G$ starts by computing the theoretical minimum embedding dimension $d_E$ as described in Section 6.3.1. After that, the delay-vectors $\mathbf{y}_{d_E}(m)$ have to be constructed according to Equation (6.4). Using these vectors, the matrix $\boldsymbol{Y}_{d_E}^{M}$ is built according to Equation (6.12). The algorithm now projects the matrix $\boldsymbol{Y}_{d_E}^{M}$ into $\Re^d$ to find $\boldsymbol{Y}_d^{M,\,p}$ where $d = 1, 2, \ldots, d_E$. The projection is done by using the PCA projection method. In the space $\Re^d$, the algorithm searches for $\mathbf{y}_d^p(n) = \mathbf{y}_d^p(i_d^1(m))$ which is the nearest neighbor of the reference point $\mathbf{y}_d^p(m)$. The search for the nearest neighbor of $\mathbf{y}_d^p(m)$ is done globally (among the $M-1$ points).

In the next step, the algorithm runs a test to find if $\mathbf{y}_d^p(n)$ is a FNN of $\mathbf{y}_d^p(m)$. To do that, it increases the time of both points ($\mathbf{y}_d^p(m)$ and $\mathbf{y}_d^p(n)$) and measures the distance between them. The test of the distance increase in the $\mathrm{CDT}_G$ algorithm is done in the space $\Re^d$ rather than in $\Re^{d_E}$ (as in the $\mathrm{CDT}_L$ algorithm). The search for the FNNs in $\Re^d$ has the advantage that if the two neighbors came from an attractor intersection (FNN), the distance between them as time increases will increase faster than if they were true neighbors. To ap-

ply this idea, the distance between the two neighbors $\mathbf{y}_d^p(m + \Delta)$ and $\mathbf{y}_d^p(n + \Delta)$ is measured according to the equation:

$$\sigma_\Delta(m) = \left\| \mathbf{y}_d^p(m + \Delta) - \mathbf{y}_d^p(n + \Delta) \right\|, \tag{6.15}$$

where $\Delta$ is the time step. If $\sigma_\Delta(m)$ reaches some predefined threshold before few steps ahead, the $\mathrm{CDT}_G$ algorithm labels $\mathbf{y}_d^p(n)$ as a FNN.

The threshold used by the algorithm is a fraction of the attractor mean ($\zeta\beta$), as shown in Equation (6.11). For the number of time steps, the algorithm uses the delay-time *T* as a measure of how fast the neighboring points diverge from each other. This test is repeated until the last data point, and then the percentage of the FNNs is recorded as a function of *d*. As *d* increases, the percentage of the FNNs will decrease. At one point, further increase of *d* does not change the percentage of the FNNs significantly. At this time, the minimum embedding dimension $d_L$ has been determined. In Chapter 7, we apply the $\mathrm{CDT}_G$ algorithm to different examples of chaotic systems and show the estimated $d_L$ resulting from the algorithm. The pseudocode in Figure 6.6 summarizes the $\mathrm{CDT}_G$ algorithm.

{The $\text{CDT}_G$ algorithm

•*Compute* the delay-time $T$ from the first minimum of the average mutual information, see Section 6.2

•*Compute* $d_E = 2d_c + 1$. If $d_c$ is not known choose an arbitrary large number for $d_E$

•*Initialize* the vector $\mathbf{p}_K$ of dimension $d_E$ x 1 to hold the count of FNN

•*Create* the delay-vectors $\mathbf{y}_{d_E}(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d_E-1)T)\right]^t$, where $m = 1, 2, \ldots, M$

•*Create* the matrix $\boldsymbol{Y}_{d_E}^M = \left[\mathbf{y}_{d_E}(1)\ \mathbf{y}_{d_E}(2)\ \ldots\ \mathbf{y}_{d_E}(M)\right]$

•*Choose* the ratio $0 < \zeta < 1$

•*Compute* $\bar{\mathbf{y}}_{d_E} = \frac{1}{M}\sum_{k=1}^{M} \mathbf{y}_{d_E}(k)$ (average vector in the space $\Re^{d_E}$)

•*Compute* $\beta = \frac{1}{M}\sum_{k=1,2,\ldots,M} \left\|\mathbf{y}_{d_E}(k) - \bar{\mathbf{y}}_{d_E}\right\|$ ($\zeta\beta$ is the threshold for false neighbors, $\left\|\bar{\mathbf{y}}_{d_E}\right\|$: mean distance from the origin.)

•*Compute* the covariance matrix $\mathbf{C} = \frac{1}{M}\sum_{k=1}^{M} [\mathbf{y}_{d_E}(k) - \bar{\mathbf{y}}_{d_E}][\mathbf{y}_{d_E}(k) - \bar{\mathbf{y}}_{d_E}]^t$

•*Compute* the eigen values $\lambda_i$ and eigen vectors $\underline{\eta}_i$ of $\mathbf{C}$, where $i = 1, 2, \ldots, d_E$

•*Arrange* the eigen values such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{d_E}$

•*Build* the basis matrix from the corresponding eigen vectors: $\mathbf{B}_{d_E} = \left[\underline{\eta}_1\ \underline{\eta}_2\ \ldots\ \underline{\eta}_{d_E}\right]$

•*for* $d = 1, 2, \ldots, d_E$ (Space dimension)

  •(Project $\boldsymbol{Y}_{d_E}^M$ into $\Re^d$)

  •$\boldsymbol{Y}_d^{M,p} = \left[\underline{\eta}_1\ \underline{\eta}_2\ \ldots\ \underline{\eta}_d\right]^t \boldsymbol{Y}_{d_E}^M$ with elements $\mathbf{y}_d^p(m)$

  •*for* $m = 1, 2, \ldots, M$ (Time index)

    •*Compute* the distance between $\mathbf{y}_d^p(m)$ and the columns of $\boldsymbol{Y}_d^{M,p}$ and save these distances in $\mathbf{q}_d$.

    •*Find* the minimum distance in $\mathbf{q}_d$ and label the index of the vector that produced it by $n$.

    •*Initialize* $\sigma_\Delta = 0$, $\Delta = 1$ ($\sigma_\Delta$: distance, $\Delta$: time step.)

    •*while* $(\sigma_\Delta < \zeta\beta)$ AND $(\Delta \leq M - max(m, n))$ AND $(\Delta \leq T)$.

      • $\sigma_\Delta = \left\|\mathbf{y}_d^p(m+\Delta) - \mathbf{y}_d^p(n+\Delta)\right\|$

      • $\Delta = \Delta + 1$

    *end while*

    (If the end of the data set was reached without a conclusion, discard $\mathbf{y}_d^p(n)$.)

    •*if* $\Delta = (M - max(m, n))$ AND $(\Delta < T)$ AND $(\sigma_\Delta < \zeta\beta)$

      •*Label* $\mathbf{y}_d^p(n)$ as *Not decidable*

    •*else if* $(\Delta \leq T)$ AND $(\sigma_\Delta \geq \zeta\beta)$

      • $p_K(d) = p_K(d) + 1$, increase the count of the FNNs

    *end if*

  *end m*

*end d*

}

Figure 6.6 Pseudocode of the $\text{CDT}_G$ algorithm

We have presented three new geometric algorithm. In the next section, we present a fourth algorithm, which is a predictive algorithm.

## 6.6 The fourth new algorithm: The Predictive

In Section 4.3.2, we presented the predictive method for estimating $d_L$ of a chaotic system. We also presented the application of the predictive method to estimate $d_L$ of the Henon map in Section 5.3. In this section, we present a new algorithm based on this technique to estimate $d_L$.

In a typical experiment, all we can observe is a set of scalar measurements $y(m)$ taken from the system. These measurements are produced from a map $h: \Re^k \rightarrow \Re^1$ where $\Re^k$ is the space of the original unknown attractor of the system. The evolution of the original states inside the attractor can be written as $\mathbf{x}(m+1) = \boldsymbol{f}(\mathbf{x}(m))$ (see Equation (4.1)). Using the delay-coordinate map, the original system dynamics can be reconstructed in a space of dimension *d*. The states in this space are the delay-vectors $\mathbf{y}_d(m) \in \Re^d$ as shown in Equation (6.4). If the reconstructed system is equivalent to the original one, the function $\mu: \Re^d \rightarrow \Re^1$ can be used to recreate the scalar measurements $y(m)$ (see Equation (4.6)). The predictive algorithm approximates $\mu$ by using a multilayer neural network with a Tapped Delay Line (TDL) connected to its input.

The main problem that may appear when using a multilayer network to approximate $\mu$ is that the error surface at the output of the network could be complicated [Hag95]. That means there could be more than one minimum and the network may converge to a local, rather than a global minimum. To insure that the network has converged to the optimal solution (global minimum), the algorithm repeats the training process a few times and then chooses the minimum Sum Squared prediction Error (SSE) from the different trials. The algorithm starts by sampling $y(m)$ at an interval *T* as follows:

$$y_s(m) = y(1 + (m-1)T), \tag{6.16}$$

where $m = 1, 2, \ldots, L$ and $L = \left\lfloor \frac{N-1}{T} \right\rfloor + 1$. For the case that the measurements are taken from a difference equation (like the Henon map), $T$ is set to 1. The mean of $y(m)$ has to be deducted from the measurements to insure that the signal is a zero mean. After that, the algorithm creates

$$s(m) = y_s(m) - \frac{1}{L} \sum_{k=1}^{L} y_s(k). \tag{6.17}$$

The delay-vectors $\mathbf{y}_d(m)$ at the input of the network are

$$\mathbf{y}_d^s(m) = \left[ s(m) \; s(m-1) \; \ldots \; s(m-d+1) \right]^t. \tag{6.18}$$

The network takes $\mathbf{y}_d^s(m-1)$ as an input to predict the current measurement $s(m)$ where $d$ is the number of taps in the TDL. As $d$ increases, the prediction error between the output of the network $\hat{s}(m)$ and $s(m)$ decreases. At one point, further increase of $d$ does not improve the prediction error significantly. At this point, the two systems are equivalent to each other and the number of taps in the TDL equals $d_L$ of the system.

To estimate $d_L$, the predictive algorithm checks to see if the change in the percentage of the change in the SSE at dimension $d$ ($\text{SSE}_d$) is less than some predefined threshold ($\gamma$) for five consecutive dimensions. That means, if

$$\frac{\left| \text{SSE}_{d+1} - \text{SSE}_d \right|}{\text{SSE}_d} < \gamma \cdot 100, \tag{6.19}$$

for five consecutive dimensions, then it sets $d_L = d - 5$. The threshold is $0 < \gamma < 3$. The pseudocode shown in Figure 6.7 summarizes the predictive algorithm.

In Chapter 7, we apply the predictive algorithm to different examples of chaotic systems and show the estimated $d_L$ resulting from the algorithm.

{The predictive algorithm

•*Compute* the delay-time $T$ from the first minimum of the average mutual information, see Section 6.2

•*Compute* $d_E = 2d_c + 1$. If $d_c$ is not known choose an arbitrary large number for $d_E$

•*Choose* $i_{max}$ (the maximum number of iterations, usually 5)

•*Choose* the threshold $0 < \gamma < 3$

•*Initialize* the matrix $\mathbf{P}_{sse}$ of dimension $i_{max}$x $d_E$ (to hold the SSE of the prediction errors)

•*Sample* $y(m) : y_s(m) = y(1 + (m-1)T)$

$$\text{•}s(m) = y_s(m) - \frac{1}{L}\sum_{k=1}^{L} y_s(k)$$ (deduct the mean of the signal, to insure the signal is a zero mean)

•for $i = 1, 2, \ldots, i_{max}$ (Iterations)

•*for* $d = 1, 2, \ldots, d_E$ (number of taps)

•*Input* = $\left[s(m-1)\ s(m-2)\ \ldots\ s(m-d)\right]$

•*Target* = $s(m)$

•*Create* a nonlinear multilayer neural network with 5 neurons in the hidden layer and 1 neuron in the output layer

•*Train* the neural network to predict the *Target* when it is presented with the *Input* over all given points. When the training stops, record the prediction errors $\mathbf{e}(m)$ and compute their Sum of Squares (SSE) and save them as a function of *d*:

$$p_{sse}(i, d) = \sum_{k=1}^{L-d} e(k)^2$$

*end d*

*end i*

•*for* $d = 1, 2, \ldots, d_E$

$$\text{•}m_{sse}(d) = min(\mathbf{p}_{sse}(d))$$

•*end d*

($d_L$ is the minimum dimension where $\mathbf{m}_{sse}$ does not change significantly with further increase in *d*)

•*for j=1,2,..., d*

•*if* $\frac{\left|\mathbf{m}_{sse}(d+1) - \mathbf{m}_{sse}(d)\right|}{\mathbf{m}_{sse}(d)} < \gamma \cdot 100$ (for five consecutive times)

$$\text{•}d_L = d - 5$$

*end if*

•*end j*

*plot* $\mathbf{m}_{sse}$ versus *d*

}

Figure 6.7 Pseudocode of the predictive algorithm

## 6.7 Chapter summary

In this chapter, we presented two algorithms by Abarbanel, which are based on the CDD and the CDT geometric methods to estimate the minimum embedding dimension of a chaotic system. We discussed some limitations of these two algorithms and suggested new approaches to overcome them. We used the global neighbor search rather than the local neighbor search method to compute the nearest neighbor of the reference point. Four new algorithms were presented that can overcome the limitations of the local neighbor search algorithms. The four algorithms are based on the three geometric methods CND, CDD, CDT, and the predictive technique. Full details for each algorithm were given and the pseudocode that summarizes each algorithm was provided. Besides giving full detail of the six algorithms that are used to estimate the minimum embedding dimension, we showed a method to find the delay-time that is used to build the delay-vectors. In the next chapter, we apply the six algorithms to different examples of chaotic systems and compare the results.

# CHAPTER 7

# MINIMUM EMBEDDING DIMENSION RESULTS

## 7.1 Introduction:

In Chapter 6, we presented six different algorithms showing two algorithms by Abarbanel and four new algorithms. Those algorithms are used to estimate the minimum embedding dimension of chaotic systems. In this chapter, we will apply the six algorithms to nine different chaotic systems. These systems are different from each other and cover artificial, industrial, and biological systems. Further, we will test the ability of the different algorithms to distinguish between chaotic signals and signals generated from a random source (noise). Chaotic systems are deterministic. They exist in different dimensions based on the dynamics that generate them. Beside being different in dimension, chaotic systems can be different with respect to noise content in the measurements taken from them.

In the next section, we will give a brief introduction to the different systems that will be used to test the algorithms. In Section 7.3 we will investigate the results of estimating $d_L$ for six noise free systems while in Section 7.4 we will estimate $d_L$ for a noisy chaotic circuit and for three practical systems from the Santa Fe competition. In Section 7.5 we will investigate the ability of the algorithms to distinguish between chaotic signals and random ones. Tabulations of the results and a comparison between them will be presented in Section 7.6. And finally we will present the chapter summary in Section 7.7.

## 7.2 Testing systems

We have shown in Chapter 5 the estimation of $d_L$ for a chaotic Henon map by applying the geometric and the predictive techniques. In this chapter, we will use more complicated systems for further investigation of the performance of the six algorithms mentioned in Chapter 6. We will divide the testing systems into two categories: 1) noise free systems, 2) noise contaminated systems. In the next two subsections, we will shed some light on nine systems belonging to these categories.

### 7.2.1 Noise free chaotic systems

1) Lorenz model: In 1963, Ed. Lorenz [Lo63] was the first scientist to discover chaos when he was modeling the fluid convection phenomena. His model is a small representation of the earth's atmosphere. It can be written as a set of three differential equations:

$$\dot{x} = s(y - x) \tag{7.1}$$

$$\dot{y} = -xz + rx - y \tag{7.2}$$

$$\dot{z} = xy - bz \tag{7.3}$$

where $s = 16$, $b = 4$, and $r = 40$. The system's equations were solved numerically using the fourth order Runge-Kutta algorithm with a fixed step size of 0.01 *sec*. The next figure shows a 3-D plot of the Lorenz attractor.

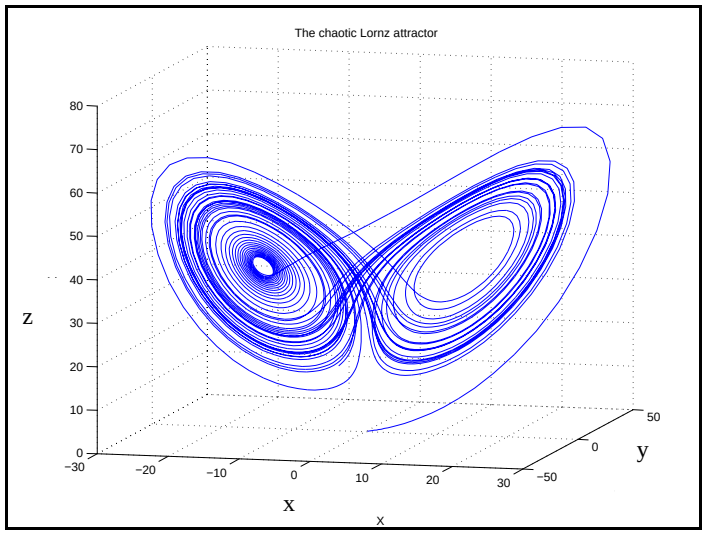



Figure 7.1 Lorenz chaotic attractor

2) The second system that we used to test the algorithms is the chaotic circuit. Chaos can exist in electrical circuits as well, as found by N. F. Rulkov *et al* [RVRDV92]. The circuit diagram is shown below.

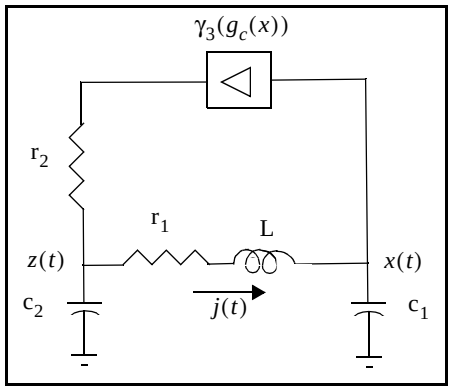



Figure 7.2 The chaotic circuit diagram

The circuit response can be written as a set of three differential equations:

$$\dot{x} = y \tag{7.4}$$

$$\dot{y} = -x - \gamma_1 y + z \tag{7.5}$$

$$\dot{z} = \gamma_2(\gamma_3 g_c(x(t)) - z) - \gamma_4 y \tag{7.6}$$

where $x$ and $z$ are the voltages across the two capacitors $c_1$ and $c_2$ respectively. $\gamma_3$ is the

gain of the nonlinear amplifier. $y(t) = j(t)\sqrt{\mathrm{L}/\mathrm{c}_1}$ where $j(t)$ is the current flowing through the inductor L. $\gamma_2 = \frac{\sqrt{\mathrm{L}\mathrm{c}_1}}{\mathrm{r}_1\mathrm{c}_2}$, $\gamma_1 = \mathrm{r}_2\sqrt{\mathrm{c}_1/\mathrm{L}}$, and $\gamma_4 = \mathrm{c}_1/\mathrm{c}_2$, $\mathrm{r}_1 = 3.98\ k\Omega$, $\mathrm{r}_2 = 361\ \Omega$, $\mathrm{c}_1 = 260\ nF$, $\mathrm{c}_2 = 170.8\ nF$, $\mathrm{L} = 152.6\ mH$ and $\gamma_3 = 24.24$. The nonlinear amplifier function is

$$g_c(x) = \begin{cases} 0.528 & \text{if } x \le -1.2 \\ x(1-x^2) & \text{if } -1.2 < x \le 1.2 \\ -0.528 & \text{if } x > 1.2 \end{cases}. \tag{7.7}$$

The circuit response is chaotic at $\gamma_3 > 22.5$ as shown in the 3-D plot below.

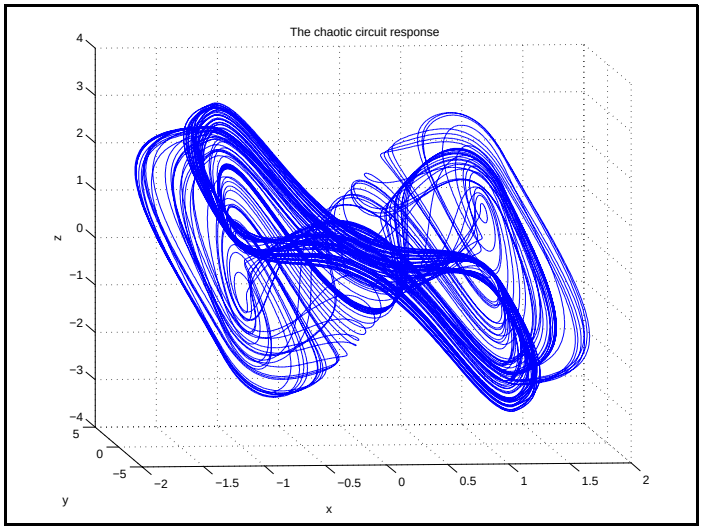



Figure 7.3 The chaotic circuit response

3) The third system that we will use to test the six algorithms is the Rossler model. In 1976, O. E. Rossler [Ros76] proposed an artificial chaotic system consisting of three differential equations:

$$\dot{x} = -y - z \tag{7.8}$$

$$\dot{y} = x + ay \tag{7.9}$$

$$\dot{z} = a + z(x - c) \tag{7.10}$$

where $a = 0.2$, and $c = 5.7$. The set of the three differential equations were solved numerically using the fourth order Runge-Kutta algorithm at a fixed step size of 0.01 *sec*. The 3-D plot below shows the Rossler attractor.

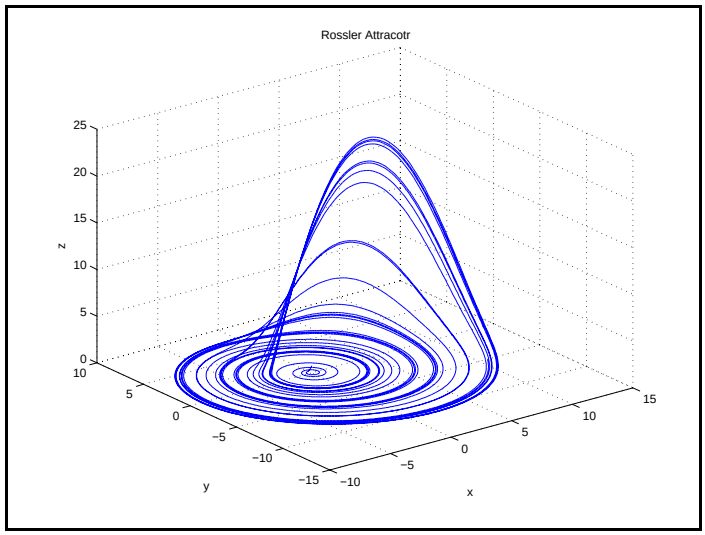



Figure 7.4 The Rossler model attractor

4) All the previous systems are three dimensional systems. To test the algorithms with higher dimensional systems, we will use Mackey-Glass (MG) chaotic system. The MG system is represented by the following differential equation

$$\dot{x}(t) = \frac{0.2x(t - t_f)}{1 + x(t - t_f)^{10}} - 0.1x(t), \tag{7.11}$$

where $t_f$ is a delay-time [MG77]. Equation (7.11) is used to model blood production. $x(t)$ represents the concentration of the blood at time $t$ (when the blood is produced) and $x(t - t_f)$ is the concentration of the blood when the request for more blood is made. For patients with Leukemia, the delay-time $t_f$ could be large which causes the concentration of

the blood to oscillate. When $t_f$ is excessively large ($t_f > 16.8$), the concentration of the blood becomes chaotic (see also [Far81]).

Equation (7.11) can be approximated by the following difference equation:

$$x(n+1) = \frac{1}{2m + bt_f}\left((2m - bt_f)x(n) + at_f\left(\frac{x(n-m)}{1 + x(n-m)^{10}} + \frac{x(n-m+1)}{1 + x(n-m+1)^{10}}\right)\right) \quad (7.12)$$

(see [KaSc00]). Equation (7.12) can be used to produce chaotic systems of dimension $m + 1$. We fixed $t_f$ to be 23 and tested the algorithms with three examples from the MG approximation of dimensions 4, 7, and 13. Up do this point, we have discussed six noise free systems. In the next subsection, we will discuss three practical systems that will also be used to test the six algorithms.

### 7.2.2 Practical systems

1) We will begin with the chaotic data set *A* from the Santa Fe competition data sets [WeGe95]. This sequence represents a special challenge since it is short (1000 points). Beside being short it is also contaminated with noise (the Signal to Noise Ratio (SNR) is up to 70 dB). It was measured from a laser machine and its attractor is a Lorenz like attractor [HAW89].

2) The second practical system is the $B_1$ data set from the same competition. The first column of this set was collected from an Electrocardiogram (ECG) signal. According to a group of researchers, an ECG signal can be modeled by a low dimensional chaotic system. This set is contaminated with noise and it is also non-stationary (caused by patient motions).

3) The third practical system is the $D_1$ data set from the same competition. This set is large ($10^5$ points), it has 9 degrees of freedom [WeGe95], and has a small non-station-

arity. It will be used to test the ability of the algorithms to estimate $d_L$ using measurements taken from a high dimensional chaotic system.

## 7.3 Estimating $d_L$ for the noise free chaotic systems

In the next two subsections, we will present the results found from Abarbanel's two algorithms: the $\text{CDD}_L$ and $\text{CDT}_L$ on the six noise free systems (Section 7.2.1). At the end of the two subsections, we will present the results found from our four algorithms (CND, $\text{CDD}_G$, $\text{CDT}_G$, and the predictive) for the same six systems. Notice that in this section we will present the plots with a brief analysis of the results. In Section 7.6, we will tabulate all the results from the different algorithms and discuss them.

### 7.3.1 Using the $\text{CDD}_L$ algorithm

We will begin by showing the results found by Abarbanel's first algorithm: the $\text{CDD}_L$ (see Section 6.3.1). The plots in Figure 7.5 show the estimated $d_L$ for the six noise free systems using this algorithm. The four curves in each graph are found by changing the number of neighbors ($N_b$ = 20, 50, 70, and 100). We can see from the figure that for systems with dimensions greater than 4, the estimation of the minimum embedding dimensions ($d_L$) are poor. The plots also show that the estimated $d_L$ for the case of the Rossler model is poor as well. It estimated $d_L = 2$ where the correct dimension is 3.

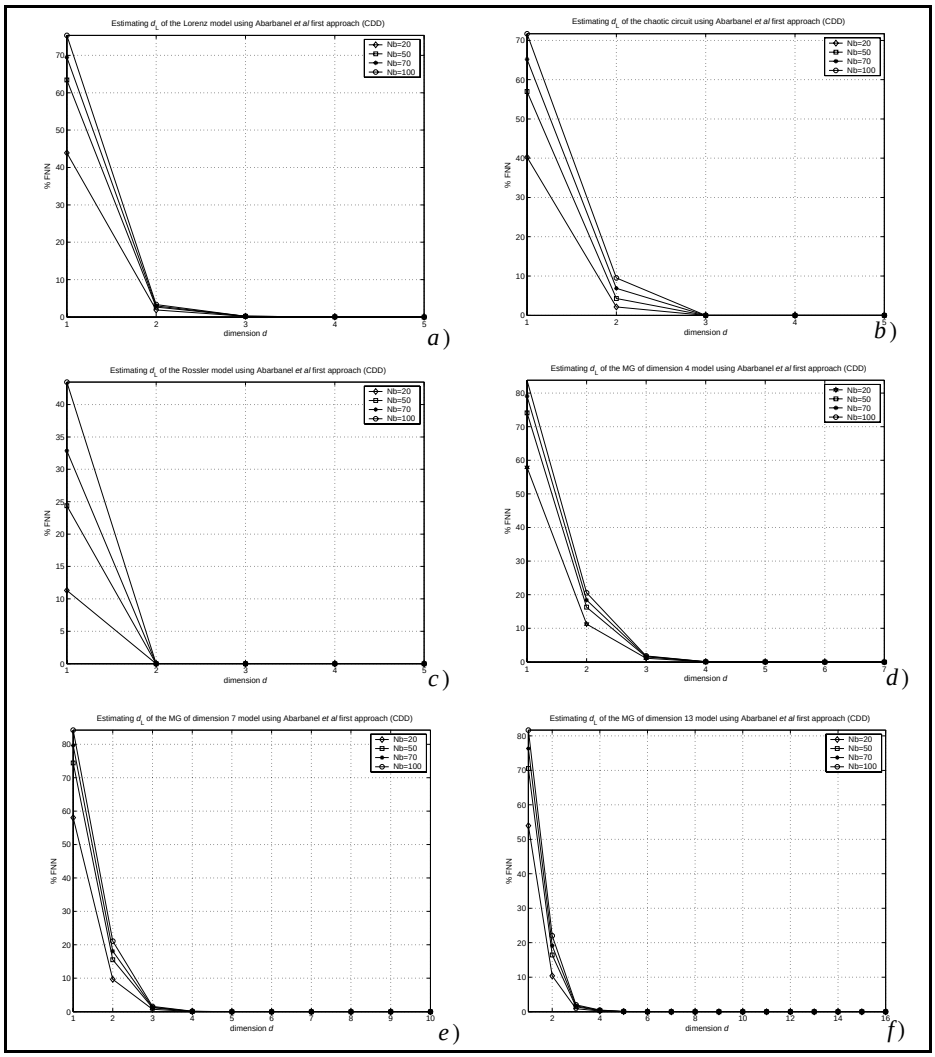



Figure 7.5 The estimated $d_L$ for the six noise free systems using the $\text{CDD}_L$ algorithm. The vertical axis is the percentage of the FNNs found from each dimension and the horizontal axis is the dimension $d$, *a*) for the Lorenz model, $d_L = 3$, *b*) for the chaotic circuit, $d_L = 3$, *c*) for the Rossler model, $d_L = 2$, *d*) for MG of dimension 4, $d_L = 4$, *e*) for MG of dimension 7, $d_L = 4$, *f*) for MG of dimension 13, $d_L = 4$

### 7.3.2 Using the $\text{CDT}_L$ algorithm

The estimated $d_L$ using the $\text{CDT}_L$ algorithm for the six noise free systems is shown in Figure 7.6. By comparing the results found from these plots with those of the $\text{CDD}_L$ algorithm, we can see that the $\text{CDT}_L$ did not improve the estimation of $d_L$ for the six noise free systems.

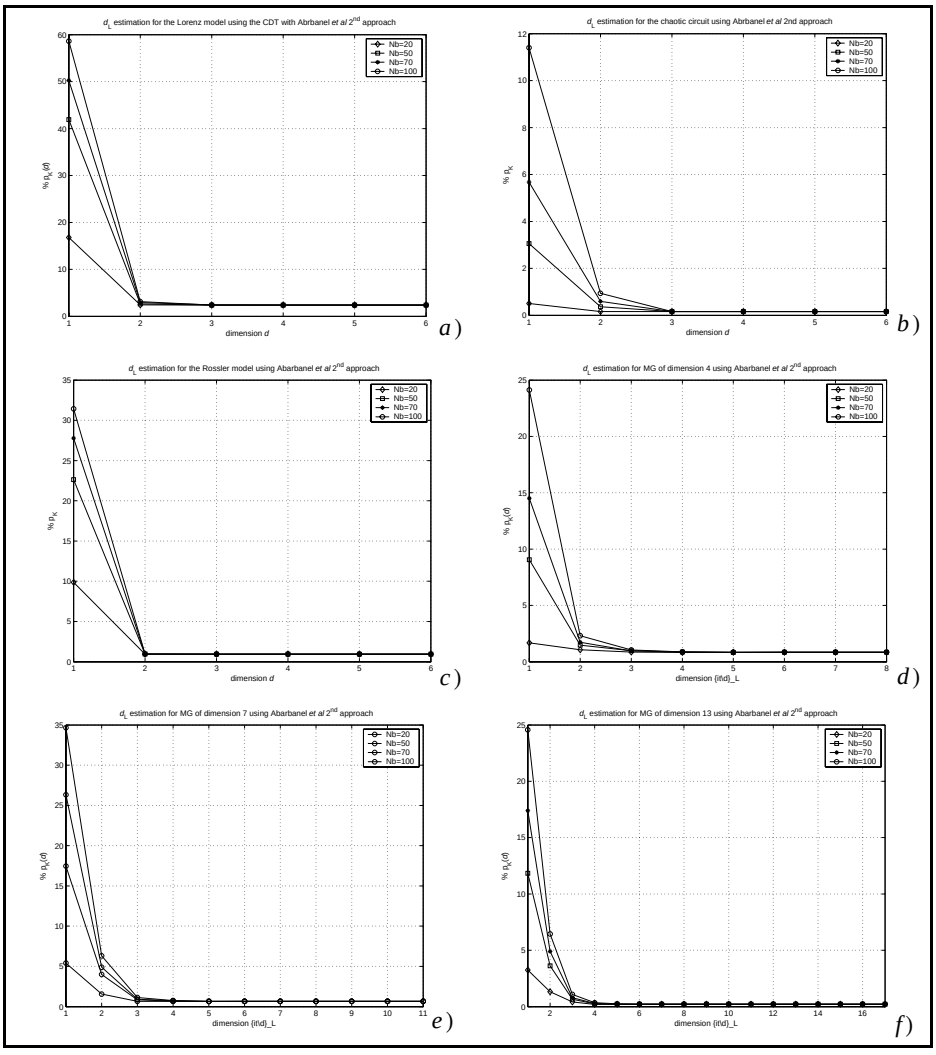



Figure 7.6 The estimated $d_L$ for the six noise free systems using the $\mathrm{CDT}_L$ algorithm. The vertical axis is the percentage of the FNNs and the horizontal axis is the dimension *d*, *a*) for Lorenz model, $d_L = 2$, *b*) for chaotic circuit, $d_L = 3$, *c*) for Rossler model, $d_L = 2$, *d*) for MG of dimension 4, $d_L = 3$, *e*) for MG of dimension 7, $d_L = 3$, *f*) for MG of dimension 13, $d_L = 4$

After presenting the results found from the $\mathrm{CDD}_L$ and the $\mathrm{CDT}_L$ algorithms, we present next the results found from our four algorithms: the CND, the $\mathrm{CDD}_G$, the $\mathrm{CDT}_G$, and the predictive.

### 7.3.3 Using the CND algorithm

The plots in Figure 7.7 show the results of the estimated $d_L$ for the six noise free systems using our first algorithm (CND) (see Section 6.5.1). As we can see from the figure,

the CND algorithm gave good estimates of $d_L$ for the systems of dimensions up to 4. On the other hand, it gave poor estimates of $d_L$ for the systems of dimensions 7 and 13.

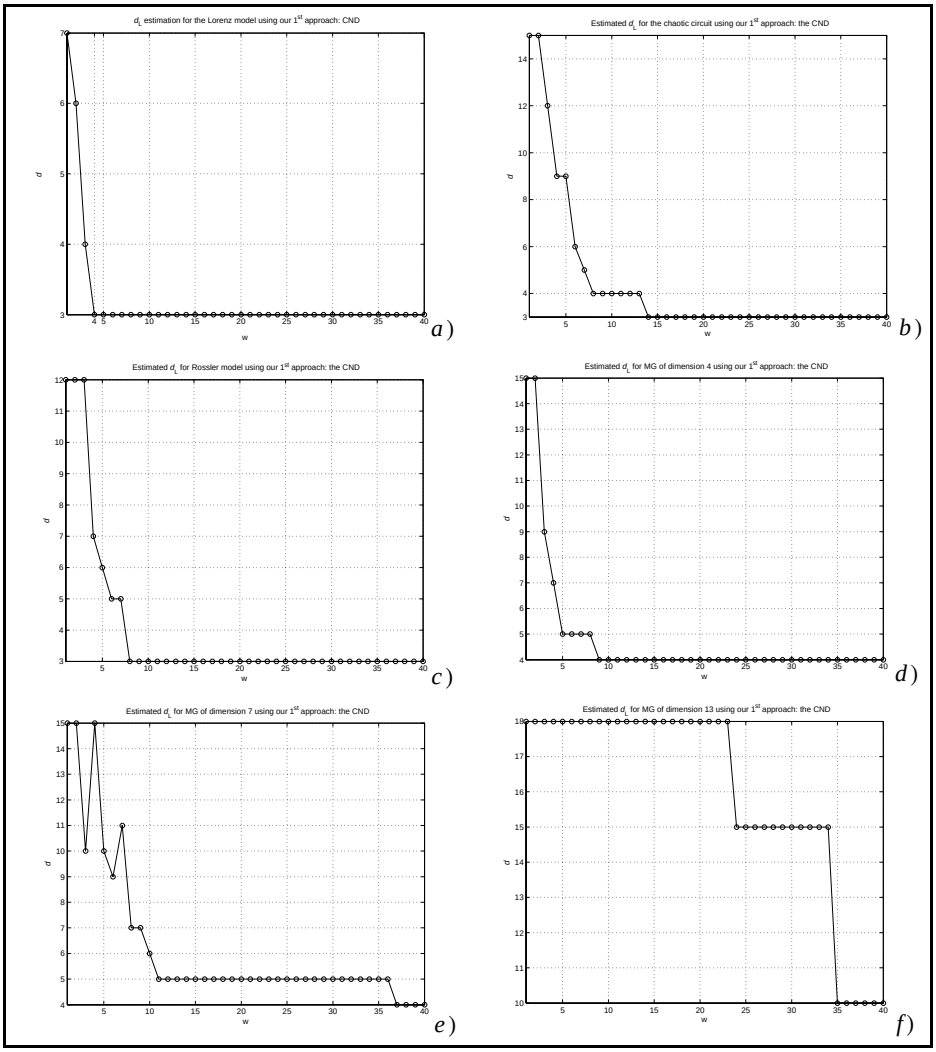



Figure 7.7 Estimating $d_L$ for the noise free systems using our first algorithm (CND). The vertical axis is the estimated $d_L$ and the horizontal axis is the number of neighbors used ($w$). *a*) for Lorenz model, $d_L = 3$, *b*) for the chaotic circuit, $d_L = 3$, *c*) for Rossler model, $d_L = 3$, *d*) for MG of dimension 4, $d_L = 4$, *e*) for MG of dimension 7, $d_L = 5$, *f*) for MG of dimension 13, this algorithm does not give a stable answer

### 7.3.4 Using the $\mathrm{CDD}_G$ algorithm

The plots in Figure 7.7 show the results of the estimated $d_L$ for the six noise free systems using our second algorithm ($\mathrm{CDD}_G$) (see Section 6.5.2). By comparing the esti-

mated $d_L$ from the plots in the figure, we can see that the $\text{CDD}_G$ algorithm gave similar results to those of the CND algorithm.

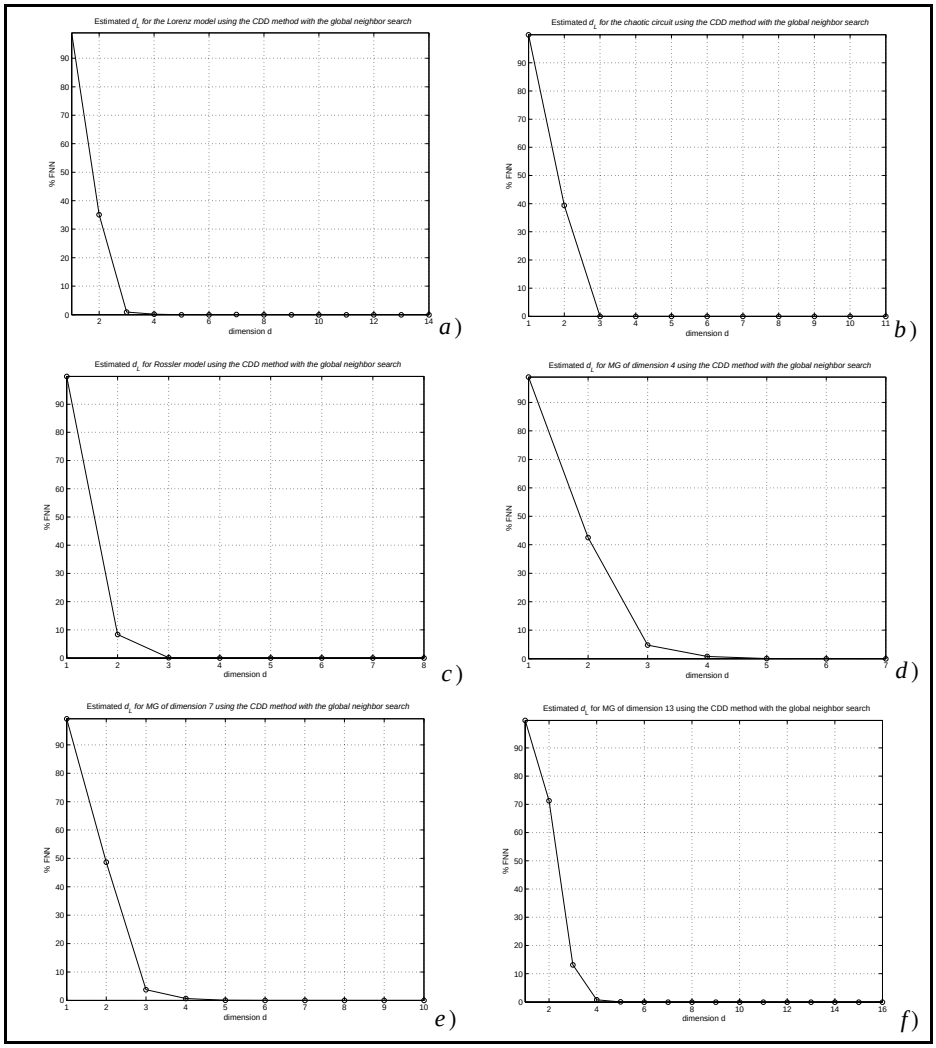



Figure 7.8 Estimating $d_L$ for the noise free systems using our second algorithm ($\text{CDD}_G$). The vertical axis is the percentage of the FNNs found and the horizontal axis is the dimension *d*. *a*) for Lorenz model, $d_L = 3$, *b*) for the chaotic circuit, $d_L = 3$, *c*) for Rossler model, $d_L = 3$, *d*) for MG of dimension 4, $d_L = 4$, *e*) for MG of dimension 7, $d_L = 4$, *f*) for MG of dimension 13, $d_L = 4$

### 7.3.5 Using the $\text{CDT}_G$ algorithm

By comparing our third algorithm ($\text{CDT}_G$) (see Section 6.5.3) with the CND and $\text{CDD}_G$, we can see from Figure 7.9 that the $\text{CDT}_G$ algorithm has improved the estimation of $d_L$ for the system of dimension 7.

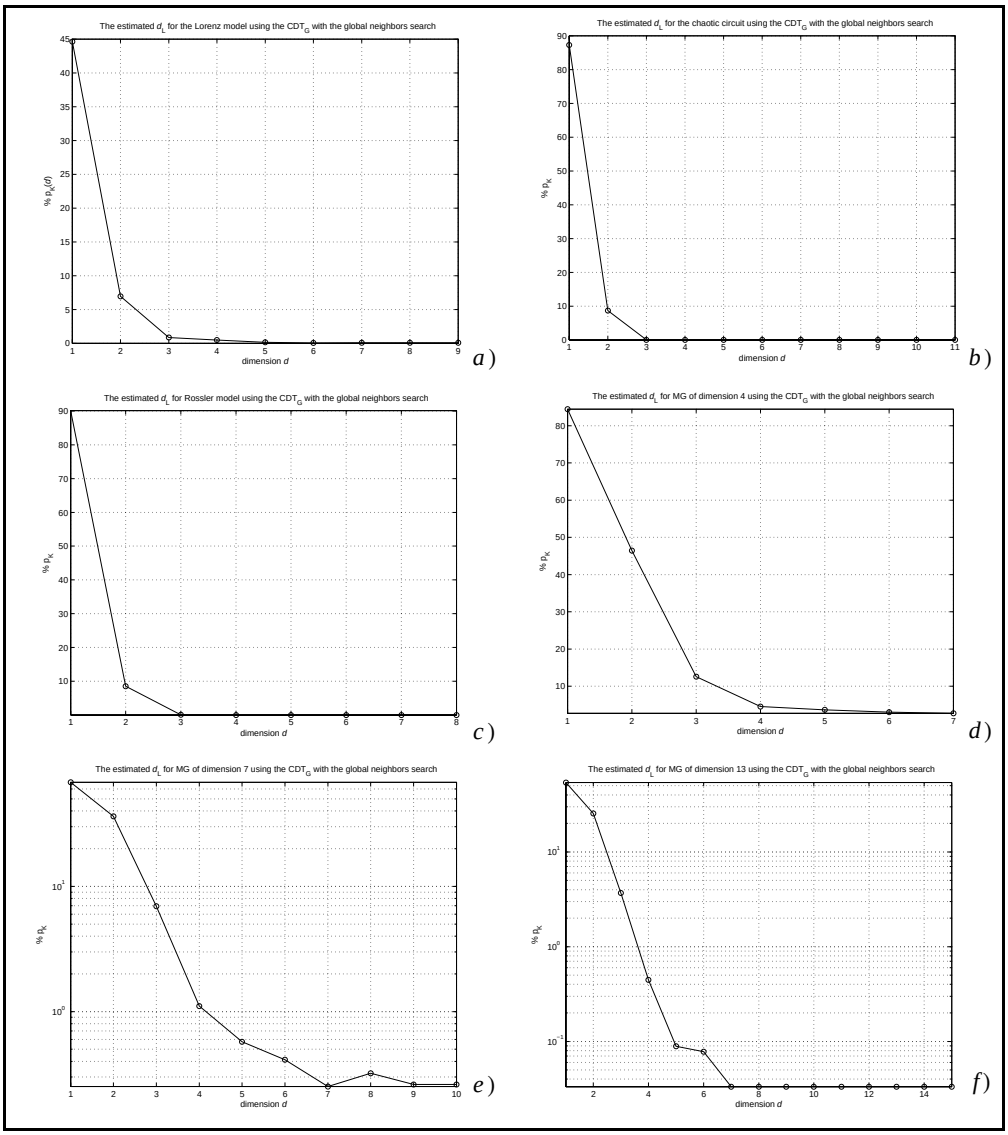



Figure 7.9 Estimating $d_L$ for the noise free systems using our third algorithm (CDT$_G$). The vertical axis is the percentage of FNNs and the horizontal axis is the dimension *d*. *a*) for Lorenz model, $d_L = 3$, *b*) for the chaotic circuit, $d_L = 3$, *c*) for Rossler model, $d_L = 3$, *d*) for MG of dimension 4, $d_L = 4$, *e*) for MG of dimension 7, $d_L = 7$, *f*) for MG of dimension 13, $d_L = 7$.

### 7.3.6 Using the predictive algorithm

For our fourth algorithm (the predictive), the estimated $d_L$ is found where the SSE of the prediction error is not changing significantly. The plots in Figure 7.10 summarize the estimation of $d_L$ for the six noise free systems.

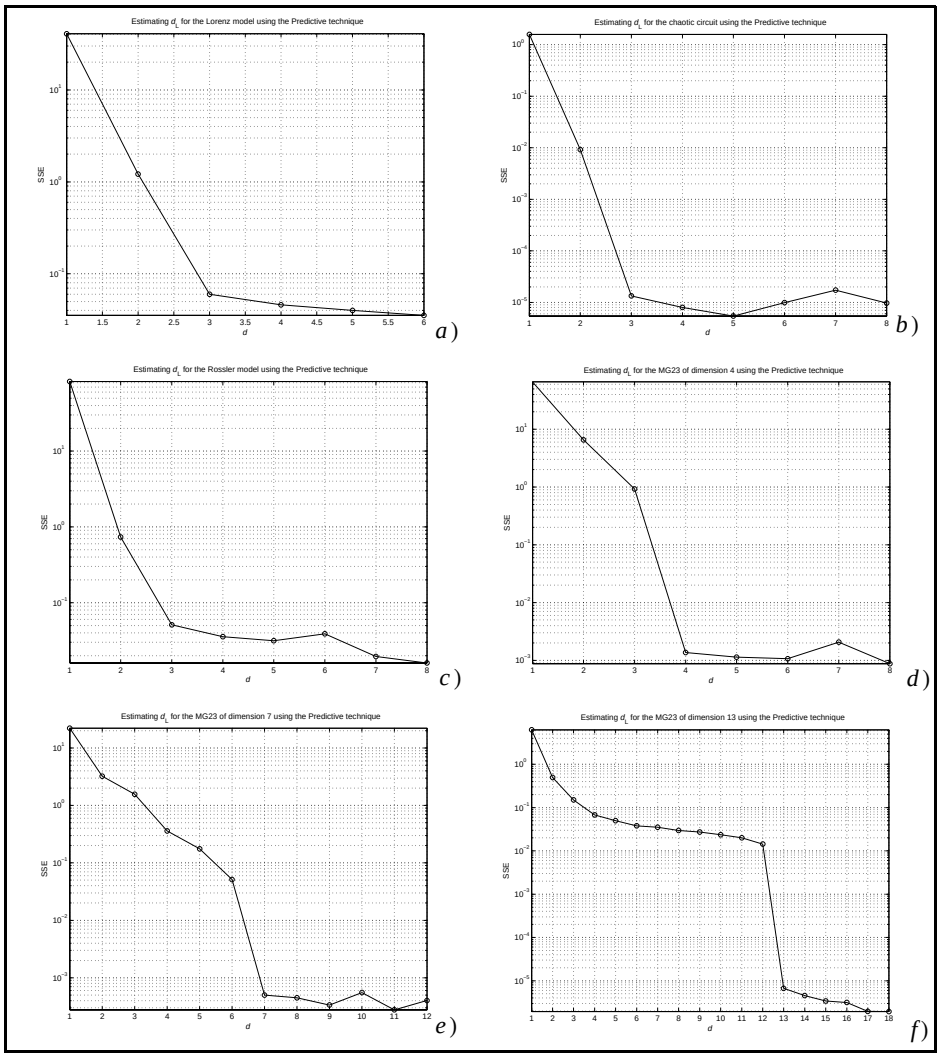



Figure 7.10 The predictive algorithm results for the noise free systems, the vertical axis is the SSE of the prediction errors and the horizontal axis is the dimension *d*, *a*) for Lorenz model, $d_L = 3$, *b*) for the chaotic circuit, $d_L = 3$, *c*) for Rossler model, $d_L = 3$, *d*) for MG of dimension 4, $d_L = 4$, *e*) for MG of dimension 7, $d_L = 7$, *f*) for MG of dimension 13, $d_L = 13$.

As we can see from Figure 7.10 above, the SSEs (of the prediction errors) of the neural network used to estimate $d_L$ have dropped significantly when the dimension (*d*) has reached the minimum embedding dimension of the signal. Then it did not improve significantly after that. As a result, the predictive algorithm gave good estimates of $d_L$ for all the six noise free systems.

In the next section, we will apply the same algorithms to estimate $d_L$ for the noisy systems.

## 7.4 Estimating $d_L$ for the noisy systems

In the previous section, we have investigated the estimation of $d_L$ for six different noise free systems using the six algorithms. In this section, we will test the same algorithms to estimate $d_L$ for three practical systems that we introduced in Section 7.2.2. But before we do that, let us investigate the estimation of $d_L$ for a noisy chaotic circuit. After that, we will show the results of estimating $d_L$ for the three practical systems.

### 7.4.1 Estimating $d_L$ for the noisy chaotic circuit (cc)

We will use 5000 points from the X-coordinate of the chaotic circuit (cc) (after discarding the transients) to represent the measurements from this system. We will also add different levels of noise to these measurements. The signal to noise ratio (SNR) in decibels (dB) is defined as:

$$\text{SNR} = 10 \log \frac{(\sigma_s)^2}{(\sigma_n)^2}, \tag{7.13}$$

where $(\sigma_s)^2$ is the variance of the signal and $(\sigma_n)^2$ is the variance of the noise. We will use the following SNR values: 200, 100, 50, and 20 dB for our purpose.

7.4.1.1 Using the $\text{CDD}_L$ algorithm

The resulting plots when using the $\text{CDD}_L$ algorithm are shown in Figure 7.11. As we can see from the figure, the $\text{CDD}_L$ algorithm was able to give the correct estimate of $d_L$ for the cases where the SNR are 200, and 100 dB. On the other hand, it was not able to estimate the correct $d_L$ for the cases where the SNR are 50, and 20 dB.

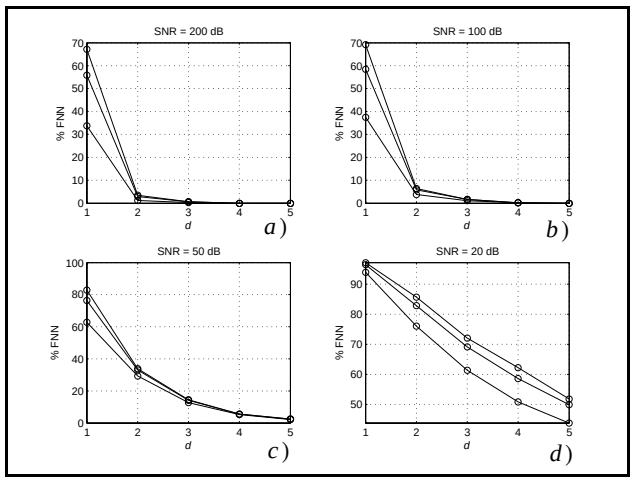



Figure 7.11 Estimating $d_L$ for the noisy cc using Abarbanel *et al* first algorithm ($\mathrm{CDD}_L$). *a*) at 200 dB, $d_L = 3$, *b*) at 100 dB, $d_L = 3$, *c*) at 50 dB, $d_L > 5$, *d*) at 20 dB, the algorithm assumes the signal is noise

### 7.4.1.2 Using the $\mathrm{CDT}_L$ algorithm.

The resulting plots when using the $\mathrm{CDT}_L$ algorithm to estimate $d_L$ for the noisy circuit are shown in Figure 7.12. We can see from the figure that the $\mathrm{CDT}_L$ was susceptible to noise. It fails to find the correct minimum embedding dimension for any of the four cases.

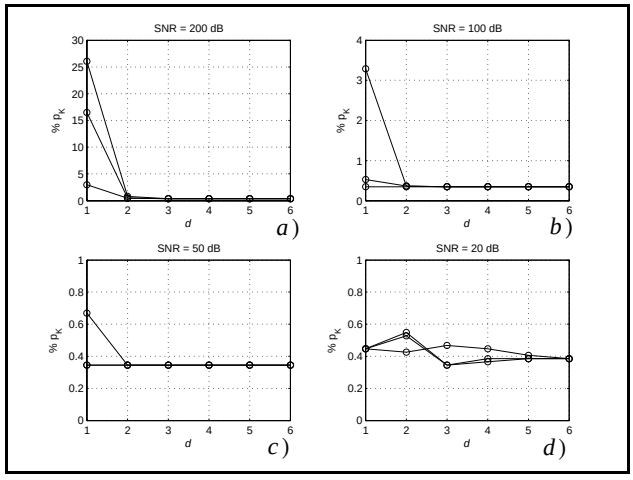



Figure 7.12 Estimating $d_L$ for the noisy cc using $\mathrm{CDT}_L$ algorithm. *a*) at 200 dB, $d_L = 2$, *b*) 100 dB, $d_L = 2$, *c*) at 50 dB, and *d*) at 20 dB the algorithm assumes the signal is noise

7.4.1.3 Using the CND algorithm

The resulting plots when using our first algorithm (the CND) are shown in Figure 7.13. We can see from the figure that the CND algorithm gave the correct $d_L$ for the cases where the SNR are 200, 100, and 50 db. It gave a wrong estimate; $d_L = 4$ when the SNR 20 dB.

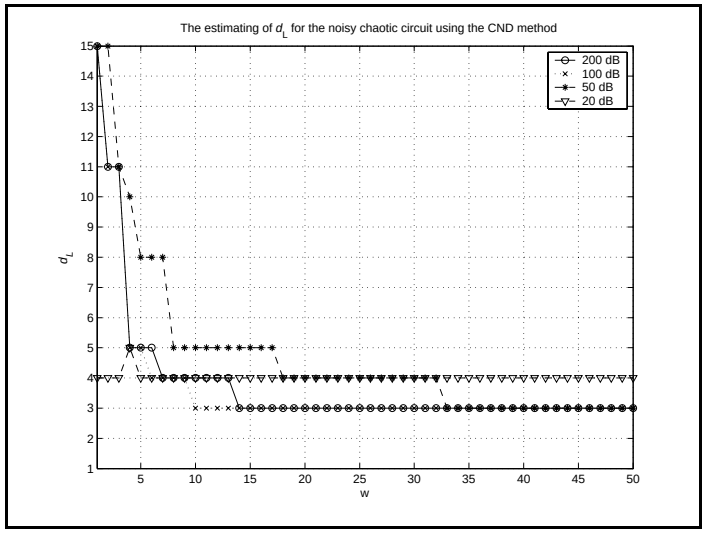



Figure 7.13 Estimating $d_L$ for the noisy cc using our first algorithm (the CND), *a*) at 200 dB, $d_L = 3$, *b*) at 100 dB, $d_L = 3$, *c*) at 50 dB, $d_L = 3$, *d*) at 20 dB, $d_L = 4$

7.4.1.4 Using the $\text{CDD}_G$ algorithm

The resulting plots when using our second algorithm ($\text{CDD}_G$) to estimate $d_L$ for the noisy cc are shown in Figure 7.14. We can see that this algorithm gave good estimates of $d_L$ for the four cases of noisy cc.

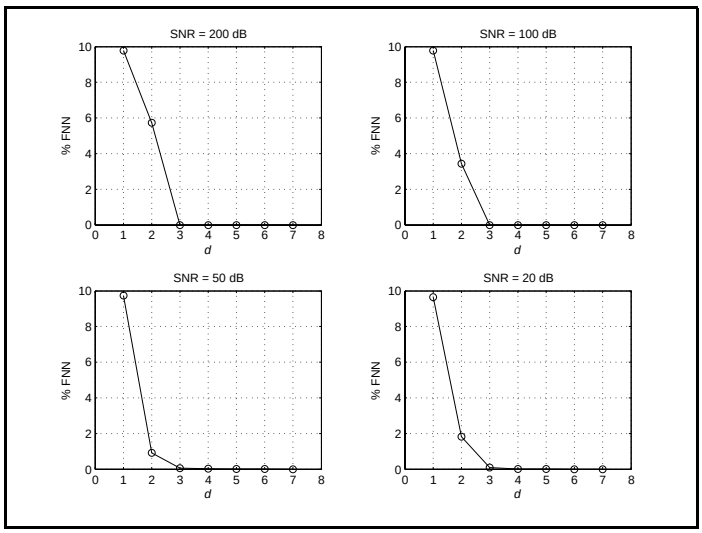



Figure 7.14 Estimating $d_L$ for the noisy cc using our second algorithm ($\text{CDD}_G$). It shows $d_L = 3$ for SNR=200, 100, 50, and 20 dB

### 7.4.1.5 Using the $\text{CDT}_G$ algorithm

The resulting plots of estimating $d_L$ for the noisy cc when using our third algorithm ($\text{CDT}_G$) are shown in Figure 7.15. As seen from the figure, this algorithm gave good estimates of $d_L$ for the cases where the SNR are 200, 100, and 50 dB. For the case where the SNR is 20 dB, it gave a wrong estimate of $d_L$, as seen from the right bottom curve. Notice here that the semilog plot (at the 20 dB case) can't show the percentage of FNNs when it equals zero (at $d = 3$).

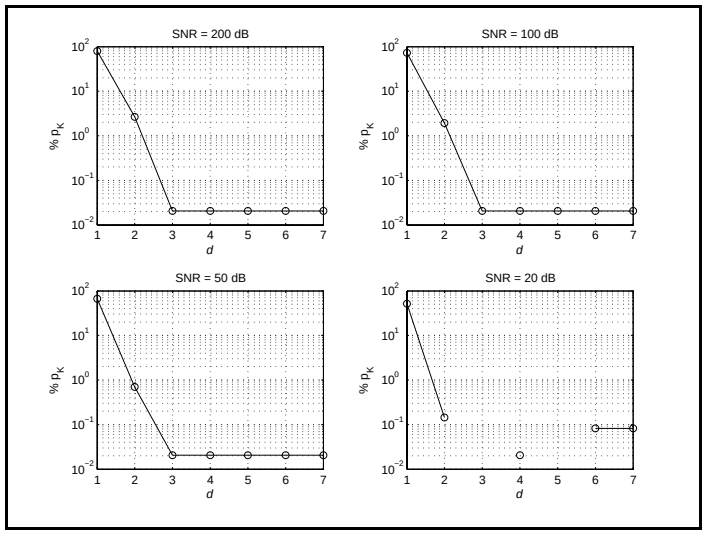



Figure 7.15 Estimated $d_L$ for the noisy cc using the $\mathrm{CDT}_G$ algorithm. For SNR=200, 100, and 50, $d_L = 3$. At SNR=20db, not stable

7.4.1.6 Using the predictive algorithm

The resulting plots when using the predictive algorithm are shown in Figure 7.16. From the figure, we can see that this algorithm gave good estimates of $d_L$ for the cases where the SNR are 200, 100, and 50 dB. It fails when the SNR is 20 dB.

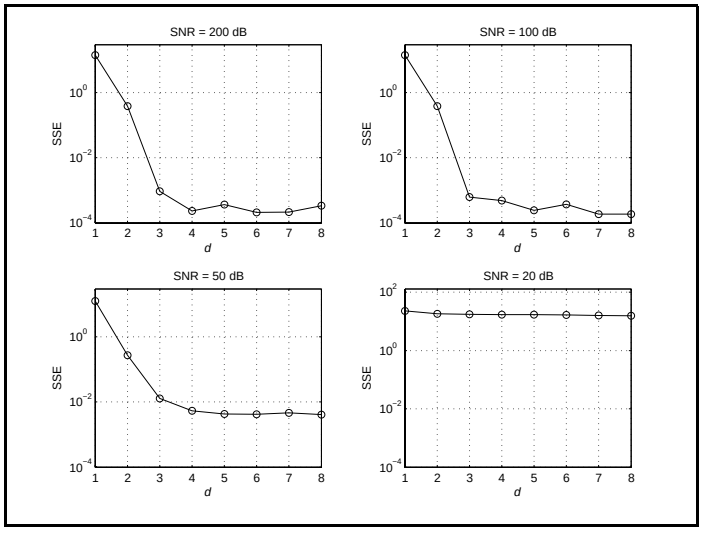



Figure 7.16 Estimating $d_L$ using our fourth algorithm (Predictive), $d_L = 3$ for 200, 100, and 50 dB, at 20 dB it assumes a random signal

### 7.4.2 Estimating $d_L$ for the Santa Fe data sets

7.4.2.1 Using the $\mathrm{CDD}_L$ algorithm

The plots in Figure 7.17 show the estimated $d_L$ for the data sets $A$, $B_1$, and $D_1$ using the $\mathrm{CDD}_L$ algorithm. We can see from the figure that this algorithm failed to find the correct $d_L$ for the cases of the $A$ and $D_1$ data sets. For the $B_1$ data set it gave $d_L = 4$.

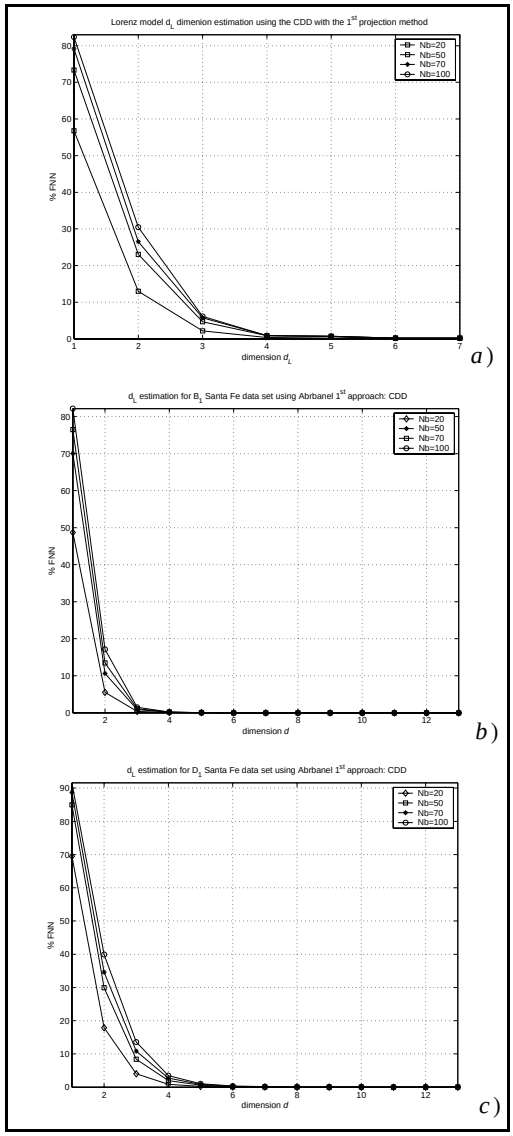



Figure 7.17 Using the $\mathrm{CDD}_L$ algorithm, *a*) for data set $A$, $d_L = 4$, *b*) for $B_1$, $d_L = 4$, *c*) for $D_1$, $d_L = 6$

7.4.2.2 Using the $\mathrm{CDT}_L$ algorithm

The plots in Figure 7.18 show the estimated $d_L$ for *A*, $B_1$, and $D_1$ Santa Fe data sets using the $\mathrm{CDT}_L$ algorithm. We can see from the figure that this algorithm gave good estimate of $d_L$ for *A* data set. It gave $d_L = 10$ for $D_1$ data set and failed to find a reasonable estimate of $d_L$ for $B_1$ data set.

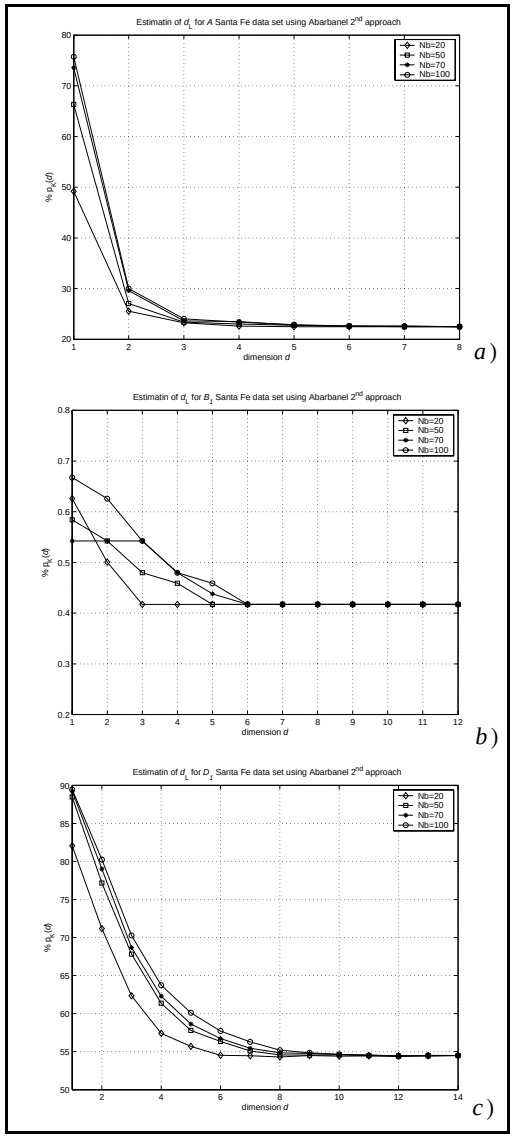



Figure 7.18 Using the $\mathrm{CDT}_L$ algorithm, *a*) for data set *A*, $d_L = 3$, *b*) for $B_1$, $d_L = 6$, *c*) for $D_1$, $d_L = 10$.

7.4.2.3 Using the CND algorithm

The plots in Figure 7.19 show the estimated $d_L$ for *A*, $B_1$, and $D_1$ Santa Fe data sets using our first algorithm (the CND). We can see that this algorithm gave a good estimate of $d_L$ for data set *A*. It gave $d_L = 5$ for data set $B_1$, while it failed to find a good estimate of $d_L$ for the data set $D_1$.

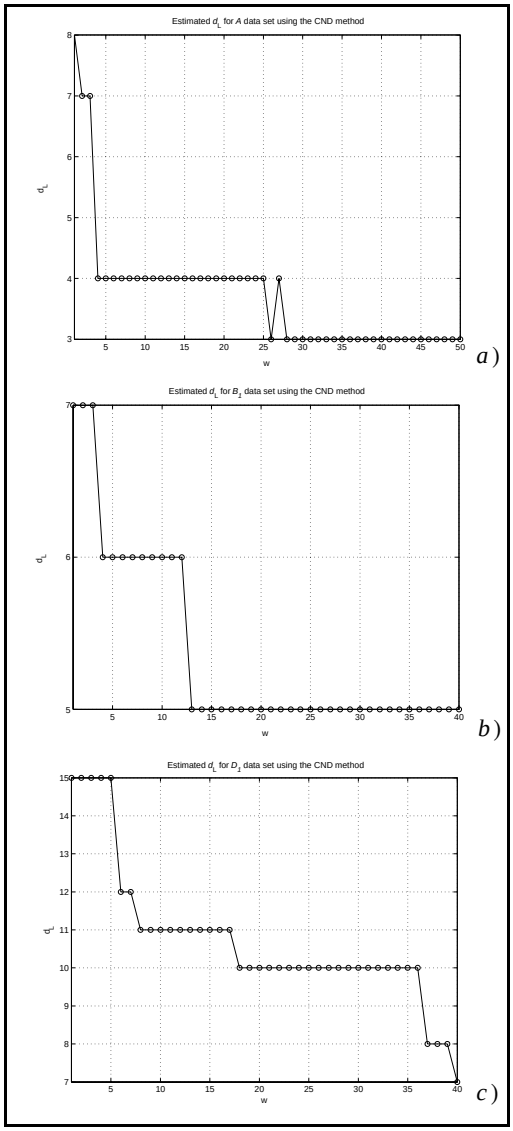



Figure 7.19 Using the CND algorithm, *a*) for data set *A*, $d_L = 3$, *b*) for $B_1$, $d_L = 5$, *c*) for $D_1$, the result is not stable

### 7.4.2.4 Using the $CDD_G$ algorithm

The plots in Figure 7.20 show the estimated $d_L$ for *A*, $B_1$, and $D_1$ Santa Fe data sets using the $CDD_G$ algorithm. As seen from the figure, this algorithm gave a good estimate of $d_L$ for the *A* data set. It gave $d_L = 5$ for the $B_1$ data set. It gave $d_L = 11$ for the $D_1$ data set, which is higher than the actual value ($d_L = 9$).

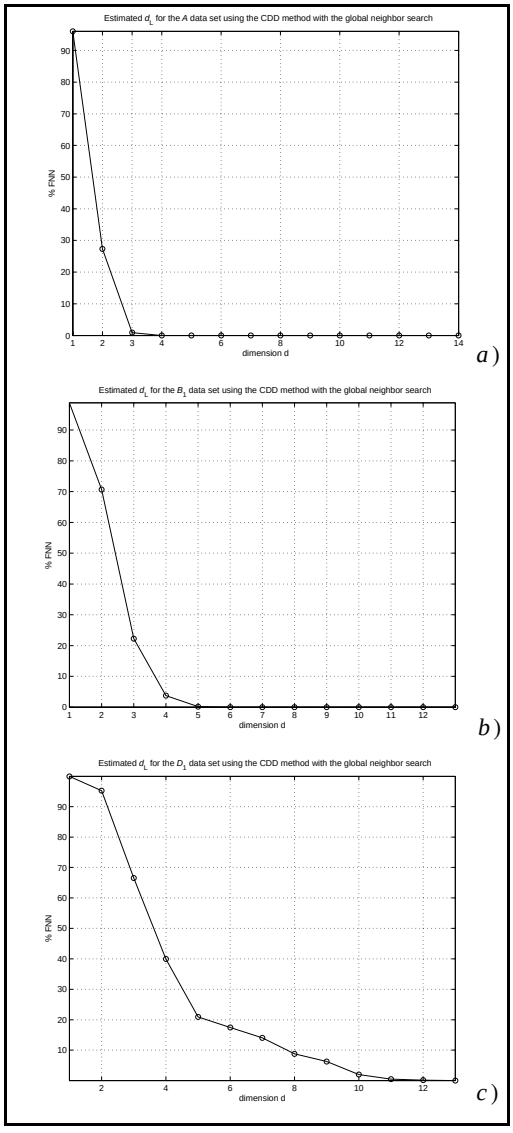



Figure 7.20 Using the $CDD_G$ algorithm, *a*) for data set *A*, $d_L = 3$, *b*) for $B_1$, $d_L = 5$, *c*) for $D_1$, $d_L = 11$

### 7.4.2.5 Using the $CDT_G$ algorithm

The plots in Figure 7.21 show the estimated $d_L$ for *A*, $B_1$, and $D_1$ Santa Fe data sets using the $CDT_G$ algorithm. Figures 7.21*a* and 7.21*b* show that the estimated $d_L$ for the *A* data set is 3, and for $B_1$ data set $d_L = 5$. This algorithm gave a bad estimate of $d_L$ for the $D_1$ data set ($d_L = 6$).

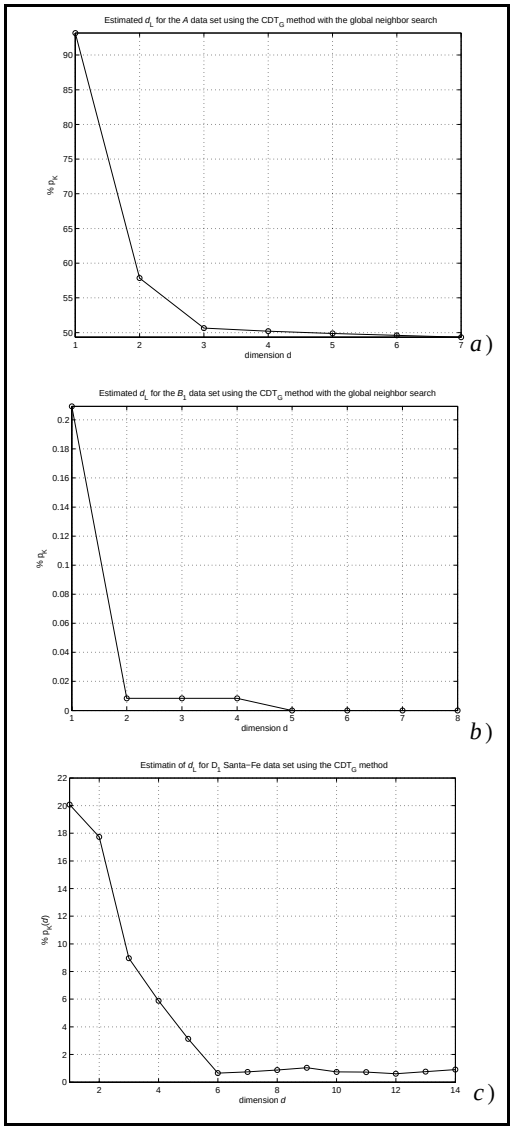



Figure 7.21 Using the $CDT_G$ algorithm, *a*) for data set *A*, $d_L = 3$, *b*) for $B_1$, $d_L = 5$, *c*) for $D_1$, $d_L = 6$

7.4.2.6 Using the predictive algorithm

The plots in Figure 7.22 show the estimated $d_L$ for *A*, $B_1$, and $D_1$ Santa Fe data sets using the predictive algorithm. As seen from the figure, this algorithm gave a good estimate of $d_L$ for the *A* data set. It estimated $d_L = 4$ for the $B_1$ data set, and $d_L = 10$ for the $D_1$ data set.

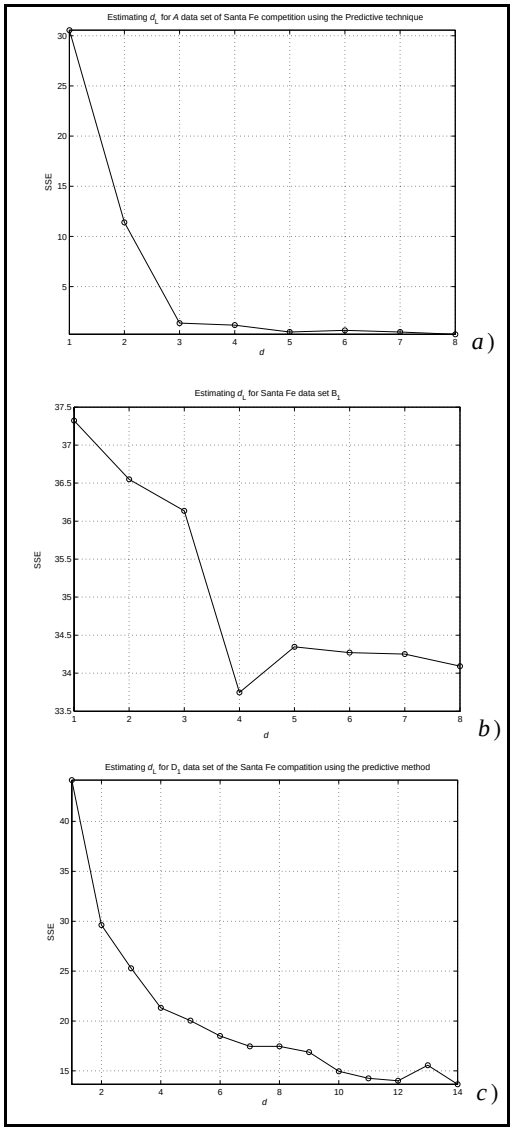



Figure 7.22 Using the predictive algorithm, *a*) for data set *A*, $d_L = 3$, *b*) for $B_1$, $d_L = 4$, *c*) for $D_1$, $d_L = 10$

Up to this point, we have shown the results of the estimated $d_L$ for the noise free and the practical systems using all the six algorithms. In the next section, we will test the ability of these algorithms to distinguish between signals generated from a chaotic system (deterministic), and signals generated from a random process (noise).

## 7.5 Testing the algorithms with random signals

We generated 3000 points from a normally distributed random process with zero mean and unit variance. We will test the ability of each algorithm to recognize that the origin of the signal is a random process rather than a deterministic source. According to Abarbanel *et al* [AbKe93] the embedding dimension will approach $\infty$ for random signals.

Figure 7.23 shows the results found from Abarbanel *et al* two algorithms. We can see that the first algorithm ($\mathrm{CDD}_L$) did not recognize that the signal is a random one. It gave 11 as the minimum embedding dimension of the signal. Their second algorithm ($\mathrm{CDT}_L$) gave $\infty$ as the minimum embedding dimension for the signal, which is what we expect from a random signal.

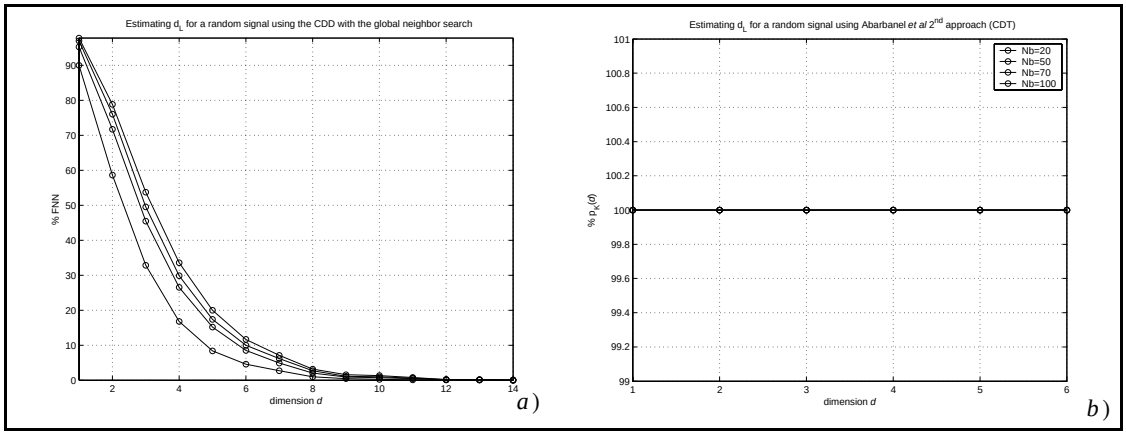



Figure 7.23 Testing Abarbanel *et al* two algorithms with a random signal, *a*) the $\mathrm{CDD}_L$ algorithm fails to recognize the random signal, *b*) the $\mathrm{CDT}_L$ algorithm succeeded in recognizing the random signal

Figure 7.24 shows the plots resulting from testing our four algorithms: the CND, $\mathrm{CDD}_G$,

$\text{CDT}_G$, and the predictive to recognize the random signal. As we can see from the plots, the CND, $\text{CDT}_G$, and the predictive algorithm were able to recognize the random signal. They estimated $d_L = \infty$. On the other hand, the $\text{CDD}_G$ algorithm fails to recognize the random signal, it estimated $d_L = 11$.

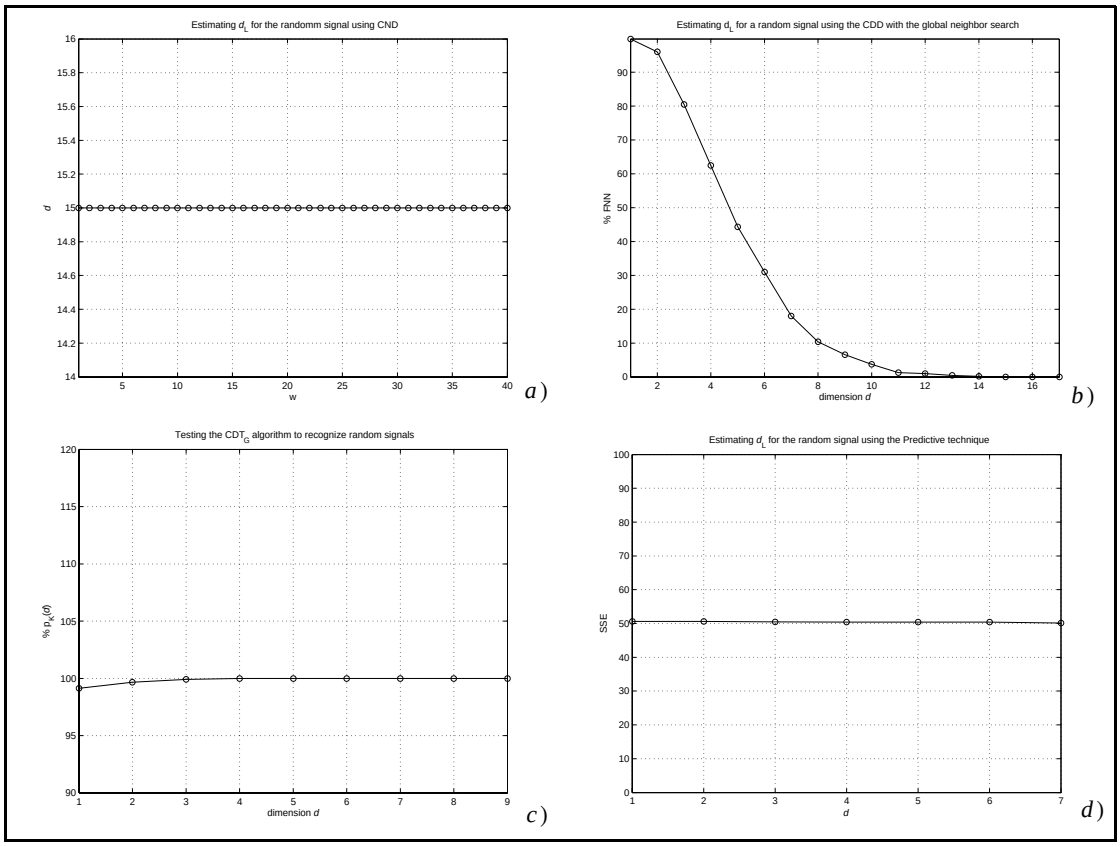



Figure 7.24 Testing our four algorithms for estimating the dimension of the random signal, *a*) the CND algorithm recognizes the signal is random by not changing the estimated $d_L$ as *w* increases, *b*) for the $\text{CDD}_G$ algorithm, it did not recognize the random signal, *c*) for the $\text{CDT}_G$ algorithm, it was able to recognize the random signal, the same for the predictive algorithm in *d*).

## 7.6 Tables and discussions

In the previous three sections we showed the resulting plots for the estimated $d_L$ for the nine systems using the six algorithms. In this section, we will summarize these results, then discuss them.

### 7.6.1 Tables

In Table 7.1, we summarize the results found from the previous section. More discussions of these results will be presented later. (Wrong estimates are circled.)

| | | Abarbanel *et al* Algorithms | | Our Algorithms | | | |
|---|---|---|---|---|---|---|---|
| Systems | Original dimension | $CDD_L$ | $CDT_L$ | CND | $CDD_G$ | $CDT_G$ | Predictive |
| Lorenz | 3 | 3 | (2) | 3 | 3 | 3 | 3 |
| Chaotic circuit | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Rossler | 3 | (2) | (2) | 3 | 3 | 3 | 3 |
| MG of dim 4 | 4 | 4 | (3) | 4 | 4 | 4 | 4 |
| MG of dim 7 | 7 | (4) | (3) | (5) | (4) | 7 | 7 |
| MG of dim 13 | 13 | (4) | (4) | (N. able) | (4) | (7) | 13 |
| cc with 200 dB | 3 | 3 | (2) | 3 | 3 | 3 | 3 |
| cc with 100 dB | 3 | 3 | (2) | 3 | 3 | 3 | 3 |
| cc with 50 dB | 3 | (5) | (noise) | 3 | 3 | 3 | 3 |
| cc with 20 dB | 3 | (noise) | (noise) | (4) | 3 | (N. Able) | (noise) |
| Santa Fe *A* | 3 | (4) | 3 | 3 | 3 | 3 | 3 |
| Santa Fe $B_1$ | 4 | 4 | (6) | (5) | (5) | (5) | 4 |
| Santa Fe $D_1$ | 9 | (6) | (10) | (N. able) | (11) | (6) | (10) |
| Noise recognition | | (N. Able) | Able | Able | (N. Able) | Able | Able |

Table 7.1 Tabulation of the estimated $d_L$ for all the testing systems shown in Sections 7.3 through 7.5. The wrong estimates are circled. The abbreviation cc: chaotic circuit, and N. Able: not able

Next we measured the time (in *seconds*) required by each algorithm to estimate $d_L$ for data set *A*. Table 7.2 shows the time for each algorithm.

| Abarbanel *et al* Two algorithms | | Our algorithms | | | |
|---|---|---|---|---|---|
| $CDD_L$ | $CDT_L$ | CND | $CDD_G$ | $CDT_G$ | Predictive |
| 2.26 | 11.0 | 12.7 | 0.24 | 0.85 | 140 |

Table 7.2 Six algorithms estimation time (in *seconds*) for $d_L$ of data set *A* of the Santa Fe competition, it shows the $CDD_G$ is the fastest and the predictive is the slowest.

We can see from the table above that the $\mathrm{CDD}_G$ algorithm required the minimum time; 0.24 *sec*. While the predictive algorithm required the maximum time; 140 *sec*.

### 7.6.2 Discussion of the Results

Next we will discuss how the following factors affect the results: 1) the original dimension of the system, 2) the use of the local versus the global neighbor search method, 3) the estimation time, 4) sensitivity of each algorithm to the threshold value, 5) the ability of each algorithm to recognize random signals, and 6) dependence of the algorithms on the number of points needed to estimate $d_L$. At the end of this section we will make general conclusions about the best way to estimate $d_L$.

#### 7.6.2.1 The effect of the original dimension of the system

From the results shown in Table 7.1, we can generally see that when the dimension of the original system is greater than four, the geometric algorithms ($\mathrm{CDD}_L$, $\mathrm{CDT}_L$, CND, $\mathrm{CDD}_G$, and $\mathrm{CDT}_G$) give incorrect estimates of $d_L$. On the other hand, the predictive algorithm gives the correct estimates as seen in the last column of the table. For the case of the data set $D_1$ the dimension is 9 (see [WeGe95 page 6]). We can see that the best estimate provided by our algorithms for this system is 10.

#### 7.6.2.2 Local versus global neighbor search methods

To reduce the computational cost in the local neighbors search algorithms ($\mathrm{CDD}_L$ and $\mathrm{CDT}_L$), the search for the nearest neighbor (in $\Re^d$) is done among the $N_b$ neighbors rather than the whole data set. On the other hand, the search for the $N_b$ neighbors for each point in $\Re^{d_E}$ is done by using the specialized neighbors search algorithm mentioned in Chapter 6. The $\mathrm{CDT}_L$ algorithm gave incorrect estimates of $d_L$ for the noisy chaotic circuit

(cc) as seen in the middle row of the table. While the $\mathrm{CDD}_L$ gave incorrect estimates of $d_L$ for 50 and 20 dB cases.

On the other hand, the algorithms $\mathrm{CND}$, $\mathrm{CDD}_G$, and $\mathrm{CDT}_G$ are global neighbor search algorithms. That means the nearest neighbor is computed among the whole data set. To reduce the computational cost, we only use the specialized neighbor search algorithm mentioned in Chapter 6. The performance of these algorithms has improved significantly for the noisy chaotic circuit (cc), as seen from the table. Further, we can see that the CND and the predictive algorithms were not able to give the correct estimate of $d_L$ when the $\mathrm{SNR} = 20$ dB.

7.6.2.3 Estimation time

Table 7.2 shows the time required to estimate $d_L$ for the *A* data set using the six algorithms. The sequence of algorithms arranged in a descending order of speed is as follows: $\mathrm{CDD}_G$, $\mathrm{CDT}_G$, $\mathrm{CDD}_L$, $\mathrm{CDT}_L$, $\mathrm{CND}$, then the predictive algorithm.

7.6.2.4 Sensitivity to the threshold value

We found that our algorithms were not sensitive to the threshold value used in the estimation process. In the CND algorithm, we used a threshold value between 0 and 3. For the $\mathrm{CDD}_G$ algorithm, we used a threshold value between 0.1 and 0.5. For the case of the $\mathrm{CDT}_G$ algorithm, the threshold used was between 2 and 10. Choosing the value of the threshold depends on the amount of noise contained in the signal.

7.6.2.5 Noise detection

From the last row of in Table 7.1, we can see that the $\mathrm{CDD}_L$ and the $\mathrm{CDD}_G$ algorithms were not able to distinguish random signals from chaotic ones. Both algorithms gave 11 as the minimum embedding dimension, which is not correct since the signal is not deterministic.

7.6.2.6 Dependence of the algorithms on the number of data points

From the results found in Table 7.1, we conclude that the algorithms are not sensitive to the number of points needed to give the correct estimate of $d_L$. For the predictive technique, it is important to insure that the number of points is greater than the number of parameters used in the neural network to prevent over fitting [Hag95].

### 7.6.3 General conclusions

From what we have seen above, we conclude the following: 1) The predictive algorithm gives the best results as along as the SNR is not too low. However, this algorithm has its own drawback. Its computational time is much larger than those of the geometric algorithms. 2) Using the global neighbor search reduces the computational time and improves the estimation. 3) To give confidence to the estimation process, one can run more than one algorithm and compare their results. We suggest the use of the predictive and the $\mathrm{CDD}_G$ algorithms to do the estimation.

## 7.7 Chapter Summary

We have seen in this chapter the results of the estimated minimum embedding dimension for many chaotic dynamical systems using the different algorithms that we presented in Chapter 6. We also gave full discussion of the results that we found and drew

conclusions on the best way to estimate the minimum embedding dimension of chaotic systems. In the Chapter 8, we will talk about the theory of Lyapunov exponents of chaotic systems.

# CHAPTER 8

# THEORY OF LYAPUNOV EXPONENTS

## 8.1 Introduction

In Chapter 1, we said that the objective of this research is to model chaotic systems. We also said that in addition to estimating the model order (the minimum embedding dimension $d_L$), we need to estimate the model parameters (the set of Lyapunov exponents). In Chapters 3 through 7, we discussed estimating $d_L$. In Chapter 2, we found that one of the main characteristics of chaotic systems is their sensitivity to initial conditions. This sensitivity is quantified by the value of the Lyapunov exponents of the chaotic system, as will be seen later. In this chapter, we will explore the theory of Lyapunov exponents and use it to define equivalent chaotic systems. We will also prove a new theorem that relates the poles of a linear system to the set of Lyapunov exponents (LEs).

In the next section, we discuss the sensitivity of some linear systems to initial conditions. In Section 8.3, we explore the theory of first order chaotic systems. The set of LEs for a multidimensional chaotic system is shown in Section 8.4. Two sets of invariants that determine equivalent chaotic systems are discussed in Section 8.5. LEs can be approximated by Jacobian matrices as shown in Section 8.6. Some linear algebra definitions and theorems are presented in Section 8.7. Some definitions and theorems from multilinear algebra are presented in Section 8.8. Using these concepts from linear and multilinear al-

gebra, a new theorem that relates the poles of a linear system to the set of LEs is proven in Section 8.9. Finally in Section 8.10 we present the chapter summary.

## 8.2 Sensitivity of some linear systems to initial conditions

We have seen in Section 2.3.1 that one of the characteristics of chaotic systems is their sensitivity to initial conditions. But it is important to notice that not every sensitivity to initial conditions originates from a chaotic behavior. As an example, we can see in Figure 8.1 the response of the linear system:

$$x(k+1) = f(x(k)) = cx(k), \tag{8.1}$$

for two different initial conditions $x_1(0) = 0.5$ and $x_2(0) = 0.55$ where $c > 1$. As we can see from the figure, the difference between the two responses (dotted curve) grows exponentially as time increases, but the system responses (solid and dashed curves) are not chaotic.

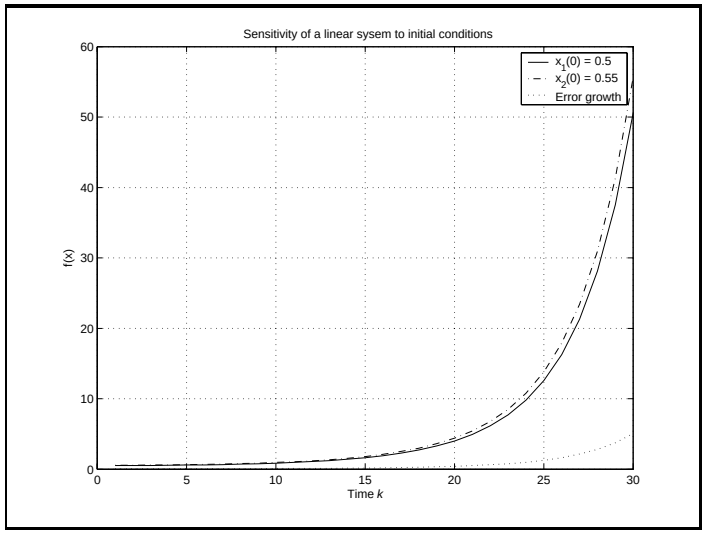



Figure 8.1 Linear systems can be sensitive to initial conditions

The solution of Equation (8.1) is

$$x(k) = c^k x(0). \tag{8.2}$$

Perturbing the initial condition $x(0)$ by $\varepsilon$ produces $x_\varepsilon(0)$ that is

$$x_\varepsilon(0) = x(0) + \varepsilon. \tag{8.3}$$

We label the initial perturbation from $x(0)$ to $x_\varepsilon(0)$ by

$$\delta(0) = x_\varepsilon(0) - x(0) = \varepsilon. \tag{8.4}$$

After $k$ iterations, the system's response for the new initial condition is $x_\varepsilon(k) = c^k x_\varepsilon(0)$, and the perturbation grows to

$$\delta(k) = x_\varepsilon(k) - x(k) = c^k \varepsilon. \tag{8.5}$$

From the above equation, we can see that

$$\left|\frac{\delta(k)}{\delta(0)}\right| = \left|\frac{c^k \varepsilon}{\varepsilon}\right| = c^k. \tag{8.6}$$

This ratio measures how fast the perturbation grows with time.

Using Equation (8.6), we will investigate the perturbation growth rate for the chaotic system of the tent map: $x(k+1) = f(x(k)) = 3/4(1 - |1 - 2x(k)|)$, see Equation (2.3). We begin by choosing three different initial conditions $\{0.202,\ 0.347, 0.869\}$ and four different initial perturbations $\{10^{-3},\ 10^{-4},\ 10^{-5},\ 10^{-6}\}$. Next we iterate the map and record the number of iterations required for the evolved perturbations to exceed some predefined threshold. Using the resulting perturbation $\delta(k)$ and the number of iterations found ($k$), we can compute the mean of the logarithm of the perturbation growth rate in Equation (8.6):

$$\log(c) = \frac{1}{k}\log\left|\frac{\delta(k)}{\delta(0)}\right|. \tag{8.7}$$

(The log in Equation (8.7) is the natural log.) Table 8.1 shows the results found from the different initial conditions $x(0)$ and initial perturbations $\delta(0)$ when the threshold used is 0.001 [PJS92 page 709].

| $x(0)$ | $\delta(0)$ | $k$ | $\delta(k)$ | $\log(c)$ |
|---|---|---|---|---|
| 0.202 | 0.001 | 1 | 0.0015 | 0.40547 |
| 0.202 | 0.0001 | 6 | 0.0011391 | 0.40547 |
| 0.202 | 1.00E-05 | 12 | 0.0012975 | 0.40547 |
| 0.202 | 1.00E-06 | 18 | -0.0014779 | 0.40547 |
| 0.347 | 0.001 | 1 | 0.0015 | 0.40547 |
| 0.347 | 0.0001 | 6 | 0.0011391 | 0.40547 |
| 0.347 | 1.00E-05 | 12 | 0.0012975 | 0.40547 |
| 0.347 | 1.00E-06 | 18 | -0.0014779 | 0.40547 |
| 0.869 | 0.001 | 1 | -0.0015 | 0.40547 |
| 0.869 | 0.0001 | 6 | -0.0011391 | 0.40547 |
| 0.869 | 1.00E-05 | 12 | 0.0012975 | 0.40547 |
| 0.869 | 1.00E-06 | 18 | 0.0014779 | 0.40547 |

Table 8.1 Estimating $\log(c)$ for $f(x(k)) = (3/4)(1 - |1 - 2x(k)|)$.

From the above table, we can see that $\log(c)$ has the same value regardless of changes in the initial condition or the initial perturbation. This result quantifies the Lyapunov exponent (LE) which is defined below.

## 8.3 Lyapunov Exponent (LE) of a first order chaotic system

Using the chain rule, we can write the ratio in Equation (8.6) as follows:

$$\left|\frac{\delta(k)}{\delta(0)}\right| = \left|\frac{\delta(k)}{\delta(k-1)}\right|\left|\frac{\delta(k-1)}{\delta(k-2)}\right|\ldots\left|\frac{\delta(1)}{\delta(0)}\right| = c^k. \tag{8.8}$$

Substituting this equation for Equation (8.7) produces:

$$\log(c) = \frac{1}{k}\sum_{i=1}^{k}\log\left|\frac{\delta(i)}{\delta(i-1)}\right|. \tag{8.9}$$

Notice that we can use the function $f$ to write Equation (8.5) as

$$\delta(k) = f(x_\varepsilon(k-1)) - f(x(k-1)), \tag{8.10}$$

which is

$$\delta(k) = f(x(k-1) + \delta(k-1)) - f(x(k-1)). \tag{8.11}$$

Now we can write the ratio in Equation (8.9) as

$$\frac{\delta(k)}{\delta(k-1)} = \frac{f(x(k-1) + \delta(k-1)) - f(x(k-1))}{\delta(k-1)}. \tag{8.12}$$

By taking the limit as $\delta \to 0$ of Equation (8.12), we can write

$$\lim_{\delta \to 0} \frac{f(x(k-1) + \delta(k-1)) - f(x(k-1))}{\delta(k-1)} = \dot{f}(x(k-1)), \tag{8.13}$$

where $\dot{f} = \frac{df(x)}{dx}$. Using Equation (8.13), the right hand side of Equation (8.9) can be written as follows (in the limit as the perturbation goes to zero)

$$\frac{1}{k}\sum_{i=1}^{k} \log\left|\lim_{\delta \to 0} \frac{\delta(i)}{\delta(i-1)}\right| = \frac{1}{k}\sum_{i=1}^{k} \log\left|\dot{f}(x(i-1))\right|. \tag{8.14}$$

The LE for a first order chaotic system is denoted by $\lambda$ and defined to be the limit as $k \to \infty$ of Equation (8.14):

$$\lambda = \lim_{k \to \infty}\left(\frac{1}{k}\sum_{i=1}^{k} \log\left|\dot{f}(x(i-1))\right|\right) = \log(c). \tag{8.15}$$

The LE quantifies the mean growth of infinitesimally small errors in the initial condition of a chaotic system. By applying Equation (8.15) to the tent map in Equation (2.3), we can compute its LE:

$$\lambda = \lim_{k \to \infty} \left( \frac{1}{k} \sum_{i=1}^{k} \log \left| \frac{d}{dx}(3/4)(1 - |1 - 2x(i)|) \right| \right), \tag{8.16}$$

which is

$$\lambda = \lim_{k \to \infty} \left( \frac{1}{k} \sum_{i=1}^{k} \log(2(3/4)|sign(1 - 2x(i))|) \right), \tag{8.17}$$

or

$$\lambda = \lim_{k \to \infty} \left( \frac{1}{k} \sum_{i=1}^{k} \log(3/2) \right) = 0.40547, \tag{8.18}$$

which is the same value that was found experimentally.

What we have seen in this section is that the LE of a first order system is a measure of the local mean growth rate of infinitesimally small errors. A chaotic system will be locally unstable, although globally bounded. Therefore, we expect the LE to be positive. A negative LE would indicate local stability, and therefore the system would not exhibit chaotic behavior.

Notice also that for a first order linear system, as in Equation (8.1), the LE is the log of the system pole. Later in this chapter, we will introduce a new theorem that demonstrates the relationship between the LEs and the poles of a multidimensional linear system.

## 8.4 Lyapunov exponents for a multidimensional system

In the previous section, we defined the LE for a first order system. In this section, we define the LEs for a multidimensional system. A multidimensional system in $\Re^d$ has $d$ LEs that characterize it. They can have positive, negative, or zero values. At least one of

these exponents has to be positive for an attractor to be chaotic [Aba95]. The set of $d$ LEs is sometimes called the spectrum of LEs.

The state evolution of a chaotic system in $\Re^d$ is governed by the map $\boldsymbol{f}$ ($\mathbf{x}(k+1) = \boldsymbol{f}(\mathbf{x}(k))$, see Equation (2.2)). Let an infinitesimal perturbation from the initial state $\mathbf{x}(0)$ to $\mathbf{x}_\varepsilon(0)$ be

$$\underline{\delta}(0) = \mathbf{x}_\varepsilon(0) - \mathbf{x}(0), \tag{8.19}$$

where the distance $\|\mathbf{x}_\varepsilon(0) - \mathbf{x}(0)\| = \varepsilon$, which is infinitesimal. After $k$ time steps, $\mathbf{x}(0)$ evolves to $\mathbf{x}(k)$, and $\mathbf{x}_\varepsilon(0)$ evolves to $\mathbf{x}_\varepsilon(k)$. The new perturbation vector is

$$\underline{\delta}(k) = \mathbf{x}_\varepsilon(k) - \mathbf{x}(k). \tag{8.20}$$

The finite time LE (sometimes called the local LE) is defined as

$$\lambda^k(\mathbf{x}(0), \underline{\delta}(0), k) = \frac{1}{k}\log\frac{\|\underline{\delta}(k)\|}{\|\underline{\delta}(0)\|}. \tag{8.21}$$

The LE (sometimes called the global LE) is defined as

$$\lambda(\mathbf{x}(0), \underline{\delta}(0)) = \lim_{k\to\infty} \lambda^k(\mathbf{x}(0), \underline{\delta}(0), k). \tag{8.22}$$

There are $d$ LEs depending on the orientation of the initial perturbation vector $\underline{\delta}(0)$ (see [EcRu85] page 630).

As we have seen above, to find the LEs for a multidimensional system, we need to find the evolution of an initial perturbation vector $\underline{\delta}(0)$. From this evolution, the ratio between the evolved vector and the initial one is used to find the LEs. More details will be given later of methods used to evaluate the LEs.

## 8.5 Invariant sets in modeling by embedding

From the set of Lyapunov exponents, Kaplan and Yorke [KaYo79] conjectured that the Lyapunov dimension can be computed, which is

$$d_{Lyp} = k_L + \frac{\sum_{j = 1, 2, \ldots, k_L} \lambda_j}{\left|\lambda_{k_L + 1}\right|}, \tag{8.23}$$

where $k_L$ is chosen such that $\sum_{j = 1, 2, \ldots, k_L} \lambda_j > 0$. The importance of the Lyapunov dimension comes from the fact that its value is closely related to the box-counting dimension of the original system (see also [Aba95])

$$d_{Lyp} \approx d_c . \tag{8.24}$$

The Lyapunov exponents and the Lyapunov dimension are the same in both the original system and its chaotic model obtained through delay embedding. This is an important result. It means that although the attractor of the delay embedding model may not look the same as the attractor of the original system, they share the same global properties. This means that the two systems are equivalent in the sense of these global properties. The other important aspect is that $d_{Lyp}$ can be found easily if one can accurately estimate the set of LEs.

## 8.6 LEs from the Jacobian matrix

As we said in the beginning of this chapter, LEs characterize chaotic systems. To compute the set of LEs of the system, we can use Equation (8.22). That means we need to follow the evolution of the initial perturbation vector $\underline{\delta}(0)$ with time. The perturbed vector $\mathbf{x}_\varepsilon(k)$ is mapped by $\boldsymbol{f}$ according to Equation (2.2):

$$\mathbf{x}_\varepsilon(k+1) = \boldsymbol{f}(\mathbf{x}_\varepsilon(k)). \tag{8.25}$$

Which can be written as

$$\mathbf{x}(k+1) + \underline{\delta}(k+1) = \boldsymbol{f}(\mathbf{x}(k) + \underline{\delta}(k)) = \boldsymbol{f}(\mathbf{x}_\varepsilon(k)). \tag{8.26}$$

Notice that $\underline{\delta}(k+1)$ does not necessarily equal $\boldsymbol{f}(\underline{\delta}(k))$ since $\boldsymbol{f}$ is not linear. The map $\boldsymbol{f}(\mathbf{x}_\varepsilon(k))$ can be approximated around $\mathbf{x}(k)$ by using the Taylor series expansion as follows:

$$\boldsymbol{f}(\mathbf{x}(k) + \underline{\delta}(k)) \approx \boldsymbol{f}(\mathbf{x}(k)) + \left.\frac{\partial \boldsymbol{f}}{\partial \mathbf{x}}\right|_{\mathbf{x} = \mathbf{x}(k)} (\mathbf{x} - \mathbf{x}(k)). \tag{8.27}$$

By using Equations (8.26) and (8.27), we can write

$$\mathbf{x}(k+1) + \underline{\delta}(k+1) \approx \mathbf{x}(k+1) + \left.\frac{\partial \boldsymbol{f}}{\partial \mathbf{x}}\right|_{\mathbf{x} = \mathbf{x}(k)} \underline{\delta}(k), \tag{8.28}$$

or

$$\underline{\delta}(k+1) \approx \left.\frac{\partial \boldsymbol{f}}{\partial \mathbf{x}}\right|_{\mathbf{x} = \mathbf{x}(k)} \underline{\delta}(k). \tag{8.29}$$

We denote the Jacobian matrix in Equation (8.29) by $\mathbf{J}_{\mathbf{x}(k)}$ whose elements are

$$(\mathbf{J}_{\mathbf{x}(k)})_{(u,\,v)} = \left.\frac{\partial f_u}{\partial x_v}\right|_{\mathbf{x} = \mathbf{x}(k)}, \tag{8.30}$$

where $u, v = 1, 2, \ldots, d$. Now we can write Equation (8.29) as

$$\underline{\delta}(k+1) \approx \mathbf{J}_{\mathbf{x}(k)}\underline{\delta}(k), \tag{8.31}$$

which can be written as

$$\underline{\delta}(k+1) \approx \mathbf{J}_{\mathbf{x}(k)}\mathbf{J}_{\mathbf{x}(k-1)}\underline{\delta}(k-1). \tag{8.32}$$

In general, we can write the evolution of an initial perturbation $\underline{\delta}(0)$ after $k$ time steps as:

$$\underline{\delta}(k) \approx \mathbf{J}_{\mathbf{x}(k-1)}\mathbf{J}_{\mathbf{x}(k-2)}\ldots\mathbf{J}_{\mathbf{x}(0)}\underline{\delta}(0). \tag{8.33}$$

For simplicity of notation, let $\mathbf{J}_{\mathbf{x}}^{k}$ represent the multiplication of the $k$ Jacobian matrices:

$$\mathbf{J}_{\mathbf{x}}^{k} = \mathbf{J}_{\mathbf{x}(k-1)}\mathbf{J}_{\mathbf{x}(k-2)}\cdots\mathbf{J}_{\mathbf{x}(0)}. \tag{8.34}$$

Now we can write the evolution of an initial perturbation vector as

$$\underline{\delta}(k) \approx \mathbf{J}_{\mathbf{x}}^{k}\underline{\delta}(0). \tag{8.35}$$

The LEs of the system are

$$\lambda = \lim_{k\to\infty}\frac{1}{k}\log\left\|\underline{\delta}(k)\right\| = \lim_{k\to\infty}\frac{1}{k}\log\left\|\mathbf{J}_{\mathbf{x}}^{k}\underline{\delta}(0)\right\|, \tag{8.36}$$

there are $d$ LEs of the system depending on the orientation of the perturbation $\underline{\delta}(0)$ (see [EcRu85] page 630).

### 8.6.1 Oseledec theorem

In 1968, Oseledec, V. [Ose68] proved that the limit in Equation (8.37) below exists and that it is independent of the initial condition,

$$\Lambda_{\mathbf{x}} = \lim_{k\to\infty}((\mathbf{J}_{\mathbf{x}}^{k})^{T}\mathbf{J}_{\mathbf{x}}^{k})^{1/(2k)}. \tag{8.37}$$

Further and most important, he proved that the logarithm of the eigen values of the matrix $\Lambda_{\mathbf{x}}$ equal the LEs of the system (see also [EcRu85]).

In the next two sections, we define some terms from linear and multilinear algebra. These definitions will be used in Section 8.9 to prove a new theorem that relates the poles of a linear system to the LEs, as computed from the eigen values of the Oseledec matrix $\Lambda_{\mathbf{x}}$. This theorem will provide insights into the meaning of the LEs and will suggest new algorithms for estimating the LEs.

## 8.7 Linear Algebra definitions

### 8.7.1 Definition: Inner product

A scalar function which operates on two vectors $\mathbf{x}, \mathbf{y} \in \Re^d$ and denoted by $(\ ,\ )$ is defined as an inner product if it satisfies the next three conditions:

$$(\mathbf{x}, \mathbf{y}) = (\mathbf{y}, \mathbf{x}). \tag{8.38}$$

$$(\mathbf{x}, a\mathbf{y}_1 + b\mathbf{y}_2) = a(\mathbf{x}, \mathbf{y}_1) + b(\mathbf{x}, \mathbf{y}_2). \tag{8.39}$$

$$(\mathbf{x}, \mathbf{x}) \geq 0. \tag{8.40}$$

The inner product is evaluated as follows

$$(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}, \tag{8.41}$$

(see [Hag95]).

### 8.7.2 Definition: Vector Norm

Given a vector $\mathbf{x} \in \Re^d$, a real valued function $g: \Re^d \rightarrow \Re^1$ is a vector norm if it satisfies the next three conditions:

$$g(\mathbf{x}) \geq 0 \qquad \text{where } g(\mathbf{x}) = 0 \text{ iff } \mathbf{x} = \mathbf{0} \tag{8.42}$$

$$g(\alpha \mathbf{x}) = |\alpha| g(\mathbf{x}) \text{ where } \alpha \in \Re^1 \tag{8.43}$$

$$g(\mathbf{x}^1 + \mathbf{x}^2) \leq g(\mathbf{x}^1) + g(\mathbf{x}^2) \text{ where } \mathbf{x}^1, \mathbf{x}^2 \in \Re^d. \tag{8.44}$$

The vector norm is denoted by $\| \ \|$. One choice of the norm is the 2-norm:

$$\|\mathbf{x}\| = (|x_1|^2 + |x_2|^2 + \ldots + |x_d|^2)^{1/2} = (\mathbf{x}^T \mathbf{x})^{1/2}. \tag{8.45}$$

After defining the vector norm, we will use it next to define the matrix norm.

### 8.7.3 Definition: Matrix Norm

Given a real $dxd$ matrix **A**, a real valued function $g: \Re^{dxd} \rightarrow \Re^1$ is a matrix norm if it satisfies the next three conditions:

$$g(\mathbf{A}) \geq 0 \quad \text{where } g(\mathbf{A}) = 0 \text{ iff } \mathbf{A} = \mathbf{0} \tag{8.46}$$

$$g(\alpha \mathbf{A}) = |\alpha| g(\mathbf{A}) \text{ where } \alpha \in \Re^1 \tag{8.47}$$

$$g(\mathbf{A}^1 + \mathbf{A}^2) \leq g(\mathbf{A}^1) + g(\mathbf{A}^2) \text{ where } \mathbf{A}^1, \mathbf{A}^2 \text{ are } dxd \text{ matrices}. \tag{8.48}$$

As with the vector norm, the matrix norm is denoted by $\| \ \|$. One choice of the matrix norm can be defined by using the vector norm as follows:

$$\|\mathbf{A}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|. \tag{8.49}$$

(Notice that we may use *T* as a super script to mean real matrix transpose. If the matrix is complex, *T* means the complex conjugate.)

8.7.3.1 Some important properties of the matrix norm

Two important properties of the matrix norm which are used in the coming sections are:

$$\|\mathbf{Ax}\| \le \|\mathbf{A}\|\|\mathbf{x}\| \text{ where } \mathbf{x} \in \Re^d. \tag{8.50}$$

$$\left\|\mathbf{A}^1\mathbf{A}^2\right\| \le \left\|\mathbf{A}^1\right\|\left\|\mathbf{A}^2\right\| \text{ where } \mathbf{A}^1, \mathbf{A}^2 \text{ are } d\text{x}d \text{ matrices}. \tag{8.51}$$

These properties can be found, for example, from [GoVa96].

8.7.3.2 Lemma: Norm of a diagonal matrix

For a diagonal matrix $\mathbf{D}_d$ with elements $\lambda_i$ where $|\lambda_d| \ge |\lambda_{d-1}| \ge \ldots \ge |\lambda_1|$,

$$\|\mathbf{D}_d\| = |\lambda_d|. \tag{8.52}$$

*proof*

By using the definition of the matrix norm in Equation (8.49), we can write

$$\|\mathbf{D}_d\|^2 \ge \|\mathbf{D}_d\mathbf{e}_i\|^2 = |\lambda_i|^2, \tag{8.53}$$

where $\mathbf{e}_i = \begin{bmatrix}0 & \ldots & 1 & \ldots & 0\end{bmatrix}^T$ such that the 1 appears in the $i^{th}$ row, and $i = 1, 2, \ldots, d$.

This means Equation (8.53) can be written as follows:

$$\|\mathbf{D}_d\|^2 \ge |\lambda_d|^2. \tag{8.54}$$

Now let's assume that $\mathbf{x}'$ is a unit vector that satisfies the norm condition:

$$\|\mathbf{D}_d\| = \|\mathbf{D}_d\mathbf{x}'\|. \tag{8.55}$$

Since $\mathbf{D}_d$ is a diagonal matrix, we can write the square of the norm in Equation (8.55) as follows:

$$\|\mathbf{D}_d\|^2 = |\lambda_d|^2|x'_1|^2 + |\lambda_{d-1}|^2|x'_2|^2 + \ldots + |\lambda_1|^2|x'_d|^2. \tag{8.56}$$

Notice that $\|\mathbf{D}_d\|^2$ has the following upper bound:

$$\|\mathbf{D}_d\|^2 \le |\lambda_d|^2(|x'_1|^2 + |x'_2|^2 + \ldots + |x'_d|^2) = |\lambda_d|^2. \tag{8.57}$$

By combining Equations (8.57) and (8.54), we can see that

$$\|\mathbf{D}_d\| = |\lambda_d|. \tag{8.58}$$

### 8.7.4 Singular value decomposition

For each $dxd$ matrix $\mathbf{A}$, there exist orthogonal matrices $\mathbf{Q}, \mathbf{V}$ such that

$$\mathbf{Q}^T\mathbf{AV} = \mathbf{S}, \tag{8.59}$$

where $\mathbf{S}$ is a diagonal matrix with elements $\sigma_d \geq \sigma_{d-1} \geq \ldots \geq \sigma_1$, which are the singular values of $\mathbf{A}$ (see [GoVa96 page 70]).

8.7.4.1 Important SVD properties

Two important properties can be found from the SVD of a matrix. The first one is that the norm of the matrix is equal to the largest singular value of the matrix:

$$\|\mathbf{A}\| = \sigma_d. \tag{8.60}$$

The second property is that the $i^{th}$ singular value of the matrix $\mathbf{A}$ is equal to the square root of the $i^{th}$ eigen value of the matrix $\mathbf{A}^T\mathbf{A}$:

$$\sigma_i(\mathbf{A}) = (\lambda_i(\mathbf{A}^T\mathbf{A}))^{1/2}, \tag{8.61}$$

where $i = 1, 2, \ldots, d$ (see [TrBa97 page 34]).

### 8.7.5 Definition: Diagonalizable matrix

A matrix $\mathbf{A}$ is said to be diagonalizable if there exists an invertible matrix $\mathbf{W}$ such that $(\mathbf{W})^{-1}\mathbf{AW} = \mathbf{D}$; a diagonal matrix.

## 8.8 Multilinear algebra definitions

### 8.8.1 Vector exterior products

Let $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d\}$ be the basis set of a vector space V. The exterior product of two vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ is denoted by $\mathbf{v}_i \wedge \mathbf{v}_j$ ($\mathbf{v}_i$ wedge $\mathbf{v}_j$). It has the following properties:

$$\mathbf{v}_i \wedge \mathbf{v}_j = 0 \text{ if } i = j \tag{8.62}$$

$$\mathbf{v}_i \wedge \mathbf{v}_j = -\mathbf{v}_j \wedge \mathbf{v}_i \text{ if } i \neq j\,. \tag{8.63}$$

This relation can be extended linearly. So if $\mathbf{a} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2$ and $\mathbf{b} = b_1\mathbf{v}_1 + b_2\mathbf{v}_2$, then

$$\mathbf{a} \wedge \mathbf{b} = (a_1\mathbf{v}_1 + a_2\mathbf{v}_2) \wedge (b_1\mathbf{v}_1 + b_2\mathbf{v}_2) = (a_1b_2 - a_2b_1)\mathbf{v}_1 \wedge \mathbf{v}_2\,. \tag{8.64}$$

Geometrically, we know that a vector **a** in the space $\Re^2$ represents a specific direction in this space. The magnitude of the vector represents its length. The exterior product of two vectors **a** and **b** represents the oriented plane segment of the parallelogram with sides **a** and **b**. Its magnitude represents the area of the resulting parallelogram. The same is true for higher order products. The next figure shows two vector exterior products. For more details see [Bay96].

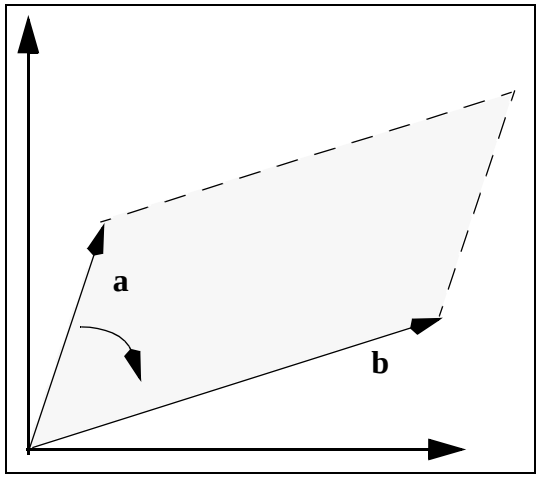



Figure 8.2 Vectors **a** and **b** exterior product (**a**^**b**)

### 8.8.2 Linear operator exterior power

Let V be a vector space of dimension $d$ with the basis set $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d\}$. Define $\mathrm{V}^{\wedge 2}$ to be the vector space of dimension $\binom{d}{2}$ with the basis set

$$\{\mathbf{v}_i \wedge \mathbf{v}_j \,|\, 1 \le i < j \le d\}. \tag{8.65}$$

By generalizing the above definition, we can see that $\mathrm{V}^{\wedge q}$ is a vector space of dimension $\binom{d}{q}$ where $0 \le q \le d$. The basis set for $\mathrm{V}^{\wedge q}$ is

$$\{\mathbf{v}_{i_1} \wedge \mathbf{v}_{i_2} \ldots \wedge \mathbf{v}_{i_q} \,|\, 1 \le i_1 < i_2 < \ldots < i_q \le d\}. \tag{8.66}$$

Let the linear operator $L$ be a map from V to itself. We define the linear operator $L$ exterior power, $L^{\wedge q}$, such that

$$L^{\wedge q}(\mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \ldots \wedge \mathbf{x}_q) = L\mathbf{x}_1 \wedge L\mathbf{x}_2 \wedge \ldots \wedge L\mathbf{x}_q, \tag{8.67}$$

where $\mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \ldots \wedge \mathbf{x}_q \in \mathrm{V}^{\wedge q}$.

Let's equip the space V with an inner product (see Section 8.7.1), and let $\mathbf{v}_i, \mathbf{w}_i \in \mathrm{V}$. From this inner product of V, we can define the inner product of $\mathrm{V}^{\wedge q}$ as the bilinear extension of

$$(\mathbf{v}_1 \wedge \mathbf{v}_2 \wedge \ldots \wedge \mathbf{v}_q, \mathbf{w}_1 \wedge \mathbf{w}_2 \wedge \ldots \wedge \mathbf{w}_q) = det\left( \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_q^T \end{bmatrix} \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \ldots & \mathbf{w}_q \end{bmatrix} \right). \tag{8.68}$$

(It can be shown that this is a valid inner product.)

8.8.2.1 Definition: Adjoint of a linear operator

Given a linear operator $L:\mathrm{V}\to\mathrm{V}$, define its adjoint to be the linear operator $L^{*}:\mathrm{V}\to\mathrm{V}$ which is determined by

$$(L\mathbf{v},\,\mathbf{w}) \;=\; (\mathbf{v},\,L^{*}\mathbf{w}), \tag{8.69}$$

where $\mathbf{v},\,\mathbf{w}\in\mathrm{V}$ (see [Arn98] page 118 and the references therein).

8.8.2.2 Lemma: Adjoint of the wedge

If $L^{*}$ is the adjoint of $L$,

$$(L^{*})^{\wedge q} \;=\; (L^{\wedge q})^{*}. \tag{8.70}$$

*Proof*

By applying the adjoint condition to $L^{\wedge q}$, we have

$$\begin{aligned}&(L^{\wedge q}(\mathbf{v}_1\wedge\mathbf{v}_2\wedge\ldots\wedge\mathbf{v}_q),\,\mathbf{w}_1\wedge\mathbf{w}_2\wedge\ldots\wedge\mathbf{w}_q) \;=\\ &(\mathbf{v}_1\wedge\mathbf{v}_2\wedge\ldots\wedge\mathbf{v}_q,\,(L^{\wedge q})^{*}(\mathbf{w}_1\wedge\mathbf{w}_2\wedge\ldots\wedge\mathbf{w}_q))\end{aligned} \tag{8.71}$$

Also

$$\begin{aligned}&(L^{\wedge q}(\mathbf{v}_1\wedge\mathbf{v}_2\wedge\ldots\wedge\mathbf{v}_q),\,\mathbf{w}_1\wedge\mathbf{w}_2\wedge\ldots\wedge\mathbf{w}_q) \;=\\ &(L\mathbf{v}_1\wedge L\mathbf{v}_2\wedge\ldots\wedge L\mathbf{v}_q,\,\mathbf{w}_1\wedge\mathbf{w}_2\wedge\ldots\wedge\mathbf{w}_q) \;=\\ &(\mathbf{v}_1\wedge\mathbf{v}_2\wedge\ldots\wedge\mathbf{v}_q,\,L^{*}\mathbf{w}_1\wedge L^{*}\mathbf{w}_2\wedge\ldots\wedge L^{*}\mathbf{w}_q) \;=\\ &(\mathbf{v}_1\wedge\mathbf{v}_2\wedge\ldots\wedge\mathbf{v}_q,\,(L^{*})^{\wedge q}(\mathbf{w}_1\wedge\mathbf{w}_2\wedge\ldots\wedge\mathbf{w}_q))\end{aligned} \tag{8.72}$$

We conclude from Equations (8.71) and (8.72) that

$$\begin{aligned}&(\mathbf{v}_1\wedge\mathbf{v}_2\wedge\ldots\wedge\mathbf{v}_q,\,(L^{\wedge q})^{*}(\mathbf{w}_1\wedge\mathbf{w}_2\wedge\ldots\wedge\mathbf{w}_q)) \;=\\ &(\mathbf{v}_1\wedge\mathbf{v}_2\wedge\ldots\wedge\mathbf{v}_q,\,(L^{*})^{\wedge q}(\mathbf{w}_1\wedge\mathbf{w}_2\wedge\ldots\wedge\mathbf{w}_q))\end{aligned} \tag{8.73}$$

for all $\mathbf{v}_i,\,\mathbf{w}_j\in\mathrm{V}$. As a result, we can see that

$$(L^*)^{\wedge q} = (L^{\wedge q})^* . \tag{8.74}$$

8.8.2.3 Linear operator properties

We list next some properties of linear operators. For reference to the proof of these properties see [Arn98] page 118 and the references therein.

(1) If **A** is a matrix representation for $L$ with respect to some basis set,

$$\lambda(\mathbf{A}) = \lambda(L) . \tag{8.75}$$

(2) If **A** is a matrix representation for $L_1$ with respect to some basis set, and **B** is a matrix representation for $L_2$ with respect to the same basis set,

$$\mathbf{AB} \text{ is a matrix representation for } L_1 L_2 . \tag{8.76}$$

(3) If **A** is a matrix representation for $L$ with respect to an orthonormal basis,

$$\mathbf{A}^T \text{ is a matrix representation for } L^* . \tag{8.77}$$

(4) From (2) and (3), we can see that

$$\mathbf{A}^T\mathbf{A} \text{ is a matrix representation for } L^* L . \tag{8.78}$$

(5) If $L_1$ and $L_2$ are linear operators,

$$(L_1 L_2)^{\wedge q} = L_1^{\wedge q} L_2^{\wedge q} . \tag{8.79}$$

(6) If **A** is a matrix representation for $L$ with respect to an orthonormal basis, and its eigen values are $\lambda_i(\mathbf{A})$, then the eigen values of $L^{\wedge q}$ are

$$\lambda(L^{\wedge q}) = \{\lambda_{i_1}(\mathbf{A})\lambda_{i_2}(\mathbf{A})\ldots\lambda_{i_q}(\mathbf{A}) \,|\, 1 \le i_1 < i_2 < \ldots < i_q \le d\} . \tag{8.80}$$

8.8.2.4 Lemma: Eigen values of $(L^* L)^{\wedge q}$

If **A** is a matrix representation for $L$ with respect to an orthonormal basis, the set of eigen values of $(L^* L)^{\wedge q}$ is

$$\lambda((L^* L)^{\wedge q}) = \left\{ (\sigma_{i_1}(\mathbf{A})\sigma_{i_2}(\mathbf{A})\ldots\sigma_{i_q}(\mathbf{A}))^2 \middle| 1 \le i_1 < i_2 < \ldots < i_q \le d \right\}. \tag{8.81}$$

*Proof*

Since $\mathbf{A}^T\mathbf{A}$ is a matrix representation for $L^* L$,

$$\lambda(\mathbf{A}^T\mathbf{A}) = \lambda(L^* L). \tag{8.82}$$

But we know from Equation (8.61) that

$$(\sigma(\mathbf{A}))^2 = \lambda(\mathbf{A}^T\mathbf{A}). \tag{8.83}$$

So from Equation (8.80), we have

$$\lambda((L^* L)^{\wedge q}) = \left\{ (\sigma_{i_1}(\mathbf{A})\sigma_{i_2}(\mathbf{A})\ldots\sigma_{i_q}(\mathbf{A}))^2 \middle| 1 \le i_1 < i_2 < \ldots < i_q \le d \right\}. \tag{8.84}$$

After defining the matrix norm and the linear operator exterior power, we will use them to prove a new theorem for computing the limit as $k \to \infty$ of the eigen values of the linear Oseledec matrix: $((\mathbf{A}^T)^k(\mathbf{A})^k)^{1/(2k)}$.

## 8.9 The linear Oseledec matrix

Consider again the Oseledec matrix $\Lambda_{\mathbf{x}}$ defined in Equation (8.37). Oseledec [Ose68] proved that this limit exists and that the LEs of the system are equal to the logarithms of the eigen values of this Oseledec matrix. In order to gain insight into the meaning of the LEs, let's consider a linear system. The linear counter part to Equation (2.2) (or Equation (8.25)) is

$$\mathbf{x}(k+1) = \boldsymbol{f}(\mathbf{x}(k)) = \mathbf{A}\mathbf{x}. \tag{8.85}$$

For this system, the Jacobian matrix, (defined in Equation (8.30)) becomes

$$\mathbf{J}_{\mathbf{x}(k)} = \mathbf{A}. \tag{8.86}$$

We then find that the product of the $k$ Jacobian matrices in Equation (8.34) becomes

$$\mathbf{J}_{\mathbf{x}}^{k} = \mathbf{A}\mathbf{A}\ldots\mathbf{A} = (\mathbf{A})^{k}. \tag{8.87}$$

The Oseledec matrix of Equation (8.37) can then be written as

$$\Lambda_{\mathbf{x}} = \lim_{k\to\infty}((\mathbf{J}_{\mathbf{x}}^{k})^{T}\mathbf{J}_{\mathbf{x}}^{k})^{1/(2k)} = \lim_{k\to\infty}((\mathbf{A}^{T})^{k}(\mathbf{A})^{k})^{1/(2k)}. \tag{8.88}$$

We know that the eigen values of the matrix **A** are the poles of a linear system. Our next step is to find the relationship between the eigen values of $\Lambda_{\mathbf{x}}$ and the eigen values of **A**.

### 8.9.1 Theorem: Eigen values of a linear Oseledec matrix

Let a matrix $\mathbf{A}:\Re^{d}\to\Re^{d}$ be diagonalizable and full rank with eigen values $\lambda_i(\mathbf{A})$, where $i = 1, 2, \ldots, d$. The limit as $k\to\infty$ of the $i^{th}$ eigen value of the matrix

$$\mathbf{O}_k = ((\mathbf{A}^{T})^{k}(\mathbf{A})^{k})^{1/(2k)} \tag{8.89}$$

converges to $|\lambda_i(\mathbf{A})|$.

*Proof*

Before we prove the theorem, we summarize the proof in the following steps:

(1) For a general matrix (not necessarily symmetric), prove that the limit of the $k^{th}$ root of the largest singular value of $(\mathbf{A})^k$ converges to the magnitude of the largest eigen value of **A**. This is done by first finding upper and lower bounds on the norm of $(\mathbf{A})^k$. Next, we prove that the limits of the $k^{th}$ root of the two bounds are equal (this part is proven in Proposition 8.9.3.2).

(2) Divide the limit of the $k^{th}$ root of the norm of $(L^{*k}L^k)^{\wedge q+1}$ by that of $(L^{*k}L^k)^{\wedge q}$. Apply the result found in (1) to this ratio to find the limit of the $k^{th}$ root of the $(d-q)^{th}$ singular value of $(\mathbf{A})^k$. ($L^k$ is the composition of *k* linear operators *L*.)

(3) Repeat step (2), but take the ratios of the eigen values instead of the singular values. This step finds the magnitude of the $(d-q)^{th}$ eigen value of **A**.

(4) Equate the two quantities found in (2) and (3) to find that the limit of the $k^{th}$ root of the $i^{th}$ singular value of $(\mathbf{A})^k$ converges to the magnitude of the $i^{th}$ eigen value of **A**.

(5) Singular values of $(\mathbf{A})^k$ are the eigen values of $(\mathbf{O}_k)^{2k}$ (Lemma 8.9.3.1). Taking the limit of the $k^{th}$ root of both values complete the proof of the theorem.

### 8.9.2 For a symmetric matrix

We will begin by proving the result for a symmetric matrix. We know from linear algebra that if **A** is symmetric,

$$\mathbf{A}^T = \mathbf{A}, \tag{8.90}$$

and its eigen vectors are orthonormal. So we can write

$$\mathbf{AW} = \mathbf{WD}, \tag{8.91}$$

where **W** is an orthonormal matrix whose columns are the eigen vectors of **A**, and **D** is a diagonal matrix containing the eigen values of **A**. Now we can write **A** as

$$\mathbf{A} = \mathbf{WDW}^T. \tag{8.92}$$

We can write the matrix $(\mathbf{O}_k)^{2k}$ in Equation (8.89) as

$$(\mathbf{O}_k)^{2k} = (\mathbf{A}^T)^k(\mathbf{A})^k, \tag{8.93}$$

which is

$$(\mathbf{O}_k)^{2k} = \mathbf{WD}^T\mathbf{W}^T\mathbf{WD}^T\mathbf{W}^T\ldots\mathbf{WD}^T\mathbf{W}^T\mathbf{WDW}^T\ldots\mathbf{WDW}^T. \tag{8.94}$$

Since $\mathbf{W}^T\mathbf{W} = \mathbf{I}$; which is the identity matrix, Equation (8.94) can be simplified to

$$(\mathbf{O}_k)^{2k} = \mathbf{W}(\mathbf{D}_a)^{2k}\mathbf{W}^T, \tag{8.95}$$

where $\mathbf{D}_a$ is a diagonal matrix with elements $\left|\lambda_i(\mathbf{A})\right|$. Eigen values of the matrices in Equation (8.95) are

$$\lambda_i((\mathbf{O}_k)^{2k}) = \lambda_i((\mathbf{D}_a)^{2k}). \tag{8.96}$$

Because $\mathbf{D}_a$ is diagonal,

$$\lambda_i(\mathbf{O}_k) = \lambda_i(\mathbf{D}_a) = \left|\lambda_i(\mathbf{A})\right|, \tag{8.97}$$

where $i = 1, 2, \ldots, d$.

### 8.9.3 For a general matrix

Now that we have proven the theorem for a symmetric matrix, next we prove it for the general case (where **A** is not necessarily symmetric). In this part of the proof, it will be convenient to define eigen values of the matrix $\mathbf{O}_k$ using singular values of the matrix $(\mathbf{A})^k$, as shown below.

8.9.3.1 Lemma: Eigen values of $\mathbf{O}_k$ are equal to the singular values of $(\mathbf{A})^k$

Given the matrix $\mathbf{O}_k$, which is defined as shown in Equation (8.89), its eigen values are equal to the $k^{th}$ root of the singular values of $(\mathbf{A})^k$:

$$\lambda_i(\mathbf{O}_k) = (\sigma_i((\mathbf{A})^k))^{1/k}, \tag{8.98}$$

where $i = 1, 2, \ldots, d$.

*proof*

Taking the $(2k)^{th}$ power of both sides of Equation (8.89) gives

$$(\mathbf{O}_k)^{2k} = (\mathbf{A}^T)^k(\mathbf{A})^k. \tag{8.99}$$

From Equation (8.61), we have

$$\sigma_i((\mathbf{A})^k) = (\lambda_i((\mathbf{A}^T)^k(\mathbf{A})^k))^{1/2}. \tag{8.100}$$

But we can see from Equation (8.99) that

$$\lambda_i((\mathbf{O}_k)^{2k}) = \lambda_i(\mathbf{A}^T)^k(\mathbf{A})^k. \tag{8.101}$$

From Equations (8.100) and (8.101), we have

$$\lambda_i((\mathbf{O}_k)^{2k}) = (\sigma_i((\mathbf{A})^k))^2. \tag{8.102}$$

Taking the $(2k)^{th}$ root of both sides of Equation (8.102) gives

$$\lambda_i(\mathbf{O}_k) = (\sigma_i((\mathbf{A})^k))^{1/k}. \tag{8.103}$$

As a result from this lemma, we can see that in order to find $\lambda_i(\mathbf{O}_k)$, we can use $(\sigma_i((\mathbf{A})^k))^{1/k}$.

8.9.3.2 Proposition: Limit of the $k^{th}$ root of $\sigma_d((\mathbf{A})^k)$

Let a real $d$x$d$ matrix $\mathbf{A}$ be diagonalizable, and let its largest eigen value be $\lambda_d(\mathbf{A})$, and its largest singular value be $\sigma_d(\mathbf{A})$, then

$$\lim_{k \to \infty} (\sigma_d((\mathbf{A})^k))^{1/k} = \left|\lambda_d(\mathbf{A})\right|. \tag{8.104}$$

*Proof*

From Equation (8.60), we can see that the norm of the matrix is equal to the largest singular value of the matrix:

$$\left\|(\mathbf{A})^k\right\| = \sigma_d((\mathbf{A})^k). \tag{8.105}$$

By using the definition of the matrix norm in Equation (8.49), we can write

$$\left\|(\mathbf{A})^k\right\| \geq \left\|(\mathbf{A})^k\mathbf{x}\right\|, \tag{8.106}$$

where $\mathbf{x}$ is a unit vector in $\Re^d$. If we let

$$\mathbf{x} = \mathbf{u}_d, \tag{8.107}$$

where $\mathbf{u}_d$ is a unit eigen vector of **A** corresponding to the largest eigen value, we can write (using Equation (8.43))

$$\left\|(\mathbf{A})^k\right\| = \sigma_d((\mathbf{A})^k) \geq \left|\lambda_d(\mathbf{A})\right|^k. \tag{8.108}$$

After finding the lower bound for $\sigma_d((\mathbf{A})^k)$, the next step is to find an upper bound for it. Since we assumed the matrix **A** is diagonalizable, we can write it as follows:

$$\mathbf{A} = \mathbf{W}\mathbf{D}(\mathbf{W})^{-1}, \tag{8.109}$$

where **D** is a diagonal matrix with elements $\lambda_i(\mathbf{A})$, the columns of **W** are the eigen vectors of **A**, and $i = 1, 2, \ldots, d$. Since $(\mathbf{W})^{-1}\mathbf{W} = \mathbf{I}$; which is the identity matrix, the matrix $(\mathbf{A})^k$ can be written as follows

$$(\mathbf{A})^k = \mathbf{W}\mathbf{D}(\mathbf{W})^{-1}\mathbf{W}\mathbf{D}(\mathbf{W})^{-1}\ldots\mathbf{W}\mathbf{D}(\mathbf{W})^{-1} = \mathbf{W}(\mathbf{D})^k(\mathbf{W})^{-1}. \tag{8.110}$$

By applying the norm property in Equation (8.51) to Equation (8.110), we have:

$$\sigma_d((\mathbf{A})^k) = \left\|(\mathbf{A})^k\right\| \leq \|\mathbf{W}\|\left\|(\mathbf{D})^k\right\|\left\|(\mathbf{W})^{-1}\right\|. \tag{8.111}$$

To find the upper bound of $\sigma_d((\mathbf{A})^k)$, we can use the norm property in Equation (8.52) to find the norm of the diagonal matrix $(\mathbf{D})^k$:

$$\left\|(\mathbf{D})^k\right\| = \left|\lambda_d(\mathbf{A})\right|^k. \tag{8.112}$$

Now we can write the upper bound of $\sigma_d((\mathbf{A})^k)$ in Equation (8.111) as follows

$$\sigma_d((\mathbf{A})^k) \leq \left|\lambda_d(\mathbf{A})\right|^k\|\mathbf{W}\|\left\|(\mathbf{W})^{-1}\right\|. \tag{8.113}$$

By combining the lower and upper bounds of $\sigma_d((\mathbf{A})^k)$, we can see that

$$\left|\lambda_d(\mathbf{A})\right|^k \le \sigma_d((\mathbf{A})^k) \le \left|\lambda_d(\mathbf{A})\right|^k (\|\mathbf{W}\|\|(\mathbf{W})^{-1}\|). \tag{8.114}$$

The $k^{th}$ root of Equation (8.114) is:

$$\left|\lambda_d(\mathbf{A})\right| \le (\sigma_d((\mathbf{A})^k))^{1/k} \le \left|\lambda_d(\mathbf{A})\right| (\|\mathbf{W}\|\|(\mathbf{W})^{-1}\|)^{1/k}. \tag{8.115}$$

Notice that for any scalar $x > 0$,

$$\lim_{k \to \infty} (x)^{1/k} = 1. \tag{8.116}$$

Therefore,

$$\lim_{k \to \infty} (\|\mathbf{W}\|\|(\mathbf{W})^{-1}\|)^{1/k} = 1. \tag{8.117}$$

Now we can see that by the limit as $k \to \infty$, the upper and lower bounds for $(\sigma_d((\mathbf{A})^k))^{1/k}$ become equal. Taking the limit as $k \to \infty$ of Equation (8.115) and using Equation (8.60) gives

$$\lim_{k \to \infty} \|(\mathbf{A})^k\|^{1/k} = \lim_{k \to \infty} (\sigma_d((\mathbf{A})^k))^{1/k} = \left|\lambda_d(\mathbf{A})\right|. \tag{8.118}$$

Recall that we are trying to prove that the eigen values of $\mathbf{O}_k$ are equal to the magnitude of the eigen values of **A**. So far, we have shown that the largest eigen value of $\mathbf{O}_k$ (which is the same as $(\sigma_d((\mathbf{A})^k))^{1/k}$) is equal to the magnitude of the largest eigen value of **A**. The next step is to show that the remaining eigen values are also equal. Let **A** be a matrix representation for $L$ with respect to an orthonormal basis. By using the linear operator property in Equation (8.76), we can see that $(\mathbf{A})^k$ is a matrix representation for the composition of the $k$ operators: $LL\ldots L$, which is denoted by $L^k$:

$$L^k = LL\ldots L \quad (k-\text{times}). \tag{8.119}$$

Similarly, we denote the composition of the $k$ adjoints of $L$ by

$$L^{*k} = L^* L^* \ldots L^* \quad (k-\text{times}). \tag{8.120}$$

The next step in the proof of the theorem is to apply Equation (8.118) to the exterior power of the linear operator $L^k$, which is $(L^k)^{\wedge q}$, in place of $(\mathbf{A})^k$. Assume now that there exists a matrix representation $\mathbf{B}_q$ for the linear operator $L^{\wedge q}$ with respect to an orthonormal basis. We know from Equation (8.76) that

$$(\mathbf{B}_q)^k \text{ is a matrix representation for } L^{\wedge q} L^{\wedge q} \ldots L^{\wedge q} \quad (k-\text{times}). \tag{8.121}$$

We also know from Equations (8.79), (8.119), and (8.121) that

$$(L^k)^{\wedge q} = (LL\ldots L)^{\wedge q} = L^{\wedge q} L^{\wedge q} \ldots L^{\wedge q}. \tag{8.122}$$

From Equation (8.60), we know that the norm of the matrix is equal to its largest singular value:

$$\left\| (\mathbf{B}_q)^k \right\| = \sigma_{max}((\mathbf{B}_q)^k). \tag{8.123}$$

We also know from Equations (8.61) and (8.78) that

$$\sigma_i((\mathbf{B}_q)^k) = (\lambda_i((\mathbf{B}_q^T)^k (\mathbf{B}_q)^k))^{1/2} = (\lambda_i((L^{*k})^{\wedge q} (L^k)^{\wedge q}))^{1/2}. \tag{8.124}$$

By using Equation (8.79), we can write Equation (8.124) as follows

$$\sigma_i((\mathbf{B}_q)^k) = (\lambda_i((L^{*k} L^k)^{\wedge q}))^{1/2}. \tag{8.125}$$

Notice that the norm of a linear operator is defined in the same way as the norm of a matrix (see [Kre98] page 33). So from Equations (8.123) and (8.125), we have

$$\left\|(\mathbf{B}_q)^k\right\| = \left\|(L^k)^{\wedge q}\right\| = (\lambda_{max}((L^{*k}L^k)^{\wedge q}))^{1/2}. \tag{8.126}$$

From Equation (8.81), we have

$$\lambda_{max}((L^* L)^{\wedge q}) = (\sigma_d(\mathbf{A})\sigma_{d-1}(\mathbf{A})\ldots\sigma_{d-q+1}(\mathbf{A}))^2. \tag{8.127}$$

Applying Equation (8.127) to $L^k$ gives

$$\lambda_{max}((L^{*k}L^k)^{\wedge q}) = (\sigma_d((\mathbf{A})^k)\sigma_{d-1}((\mathbf{A})^k)\ldots\sigma_{d-q+1}((\mathbf{A})^k))^2. \tag{8.128}$$

This means that by using Equations (8.126) and (8.128) we can write

$$\frac{\left\|(L^k)^{\wedge(q+1)}\right\|}{\left\|(L^k)^{\wedge q}\right\|} = \frac{\sigma_d((\mathbf{A})^k)\sigma_{d-1}((\mathbf{A})^k)\ldots\sigma_{d-q+1}((\mathbf{A})^k)\sigma_{d-q}((\mathbf{A})^k)}{\sigma_d((\mathbf{A})^k)\sigma_{d-1}((\mathbf{A})^k)\ldots\sigma_{d-q+1}((\mathbf{A})^k)}. \tag{8.129}$$

(Notice that since **A** is full rank, non of its singular values or eigen values is equal to zero.)

Equation (8.129) can be simplified to

$$\frac{\left\|(L^k)^{\wedge(q+1)}\right\|}{\left\|(L^k)^{\wedge q}\right\|} = \sigma_{d-q}((\mathbf{A})^k). \tag{8.130}$$

If we apply Equation (8.118) to $(\mathbf{B}_q)^k$, we have

$$\lim_{k\to\infty}\left\|(\mathbf{B}_q)^k\right\|^{1/k} = \lim_{k\to\infty}(\sigma_{max}((\mathbf{B}_q)^k))^{1/k} = \left|\lambda_{max}(\mathbf{B}_q)\right|. \tag{8.131}$$

According to Equation (8.75),

$$\lambda_i(\mathbf{B}_q) = \lambda_i(L^{\wedge q}). \tag{8.132}$$

From Equation (8.80), the largest eigen value of $L^{\wedge q}$ is

$$\lambda_{max}(L^{\wedge q}) = \lambda_d(\mathbf{A})\lambda_{d-1}(\mathbf{A})\ldots\lambda_{d-q+1}(\mathbf{A}). \tag{8.133}$$

From the left hand side of Equation (8.126), we have

$$\left\| (\mathbf{B}_q)^k \right\| = \left\| (L^k)^{\wedge q} \right\|. \tag{8.134}$$

Now we can use Equations (8.131) and (8.134) to write

$$\lim_{k \to \infty} \left\| (\mathbf{B}_q)^k \right\|^{1/k} = \lim_{k \to \infty} \left\| (L^k)^{\wedge q} \right\|^{1/k} = \left| \lambda_{max}(\mathbf{B}_q) \right|. \tag{8.135}$$

From Equations (8.132) and (8.135), we have

$$\lim_{k \to \infty} \left\| (L^k)^{\wedge q} \right\|^{1/k} = \left| \lambda_{max}(\mathbf{B}_q) \right| = \left| \lambda_{max}(L^{\wedge q}) \right|. \tag{8.136}$$

Applying Equation (8.133) to Equation (8.136) gives

$$\lim_{k \to \infty} \left\| (L^k)^{\wedge q} \right\|^{1/k} = \left| \lambda_d(\mathbf{A}) \lambda_{d-1}(\mathbf{A}) \ldots \lambda_{d-q+1}(\mathbf{A}) \right|. \tag{8.137}$$

Next we need to find the limit of the $k^{th}$ root of the ratio $\left\| (L^k)^{\wedge (q+1)} \right\|$ and $\left\| (L^k)^{\wedge q} \right\|$ by using Equation (8.137):

$$\lim_{k \to \infty} \left( \frac{\left\| (L^k)^{\wedge (q+1)} \right\|}{\left\| (L^k)^{\wedge q} \right\|} \right)^{1/k} = \left| \frac{\lambda_d(\mathbf{A}) \lambda_{d-1}(\mathbf{A}) \ldots \lambda_{d-q+1}(\mathbf{A}) \lambda_{d-q}(\mathbf{A})}{\lambda_d(\mathbf{A}) \lambda_{d-1}(\mathbf{A}) \ldots \lambda_{d-q+1}(\mathbf{A})} \right|. \tag{8.138}$$

Which can be simplified to

$$\lim_{k \to \infty} \left( \frac{\left\| (L^k)^{\wedge (q+1)} \right\|}{\left\| (L^k)^{\wedge q} \right\|} \right)^{1/k} = \left| \lambda_{d-q}(\mathbf{A}) \right|. \tag{8.139}$$

Combining Equation (8.139) with the limit of the $k^{th}$ root of Equation (8.130) gives

$$\lim_{k \to \infty} \left( \sigma_{d-q}((\mathbf{A})^k) \right)^{1/k} = \left| \lambda_{d-q}(\mathbf{A}) \right|. \tag{8.140}$$

By repeating Equation (8.140) for $q = 0, 1, \ldots, d-1$, we have

$$\lim_{k \to \infty} \left( \sigma_i((\mathbf{A})^k) \right)^{1/k} = \left| \lambda_i(\mathbf{A}) \right|, \tag{8.141}$$

where $i = 1, 2, \ldots, d$. Applying Equation (8.141) to the result of Lemma 8.9.3.1 gives

$$\lim_{k \to \infty} \lambda_i(\mathbf{O}_k) = \lim_{k \to \infty} (\sigma_i((\mathbf{A})^k))^{1/k} = \left|\lambda_i(\mathbf{A})\right|. \tag{8.142}$$

Later we will show the result of applying this theorem to estimate the set of LEs for a chaotic system.

## 8.10 Chapter summary

In this chapter, we introduced the theory of Lyapunov exponents for a first order chaotic system and a multidimensional chaotic system. We also presented the notion of invariant sets that make two chaotic systems equivalent. A new theorem that relates the poles of a linear system to the set of Lyapunov exponents was also proven here. In the next chapter, we will show methods to estimate the set of Lyapunov exponents for a chaotic system.

# CHAPTER 9

# ESTIMATION OF LYAPUNOV EXPONENTS

## 9.1 Introduction

In Chapters 5 through 7, we discussed estimating the minimum embedding dimension ($d_L$) of a chaotic system. We said that $d_L$ represents the model order. To complete the modeling process, we need also to estimate the set of $d_L$ LEs of the system. In Chapter 8, we gave the theoretical background of the LEs and proved a new theorem (linear Oseledec theorem) that relates the LEs to the poles of a linear system. The purpose of this chapter is to discuss different algorithms for estimating the LEs. We will also test these algorithms on different chaotic systems.

In Section 9.2, we will talk about the QR decomposition of a matrix. We will also talk about spurious exponents in this section. In Section 9.3, we will explore two geometric algorithms for estimating the LEs. The first algorithm is the Eckmann algorithm. The second algorithm is a new procedure based on the linear Oseledec theorem. We will also discuss a third algorithm in Section 9.4. This is a predictive algorithm that is an improvement of an existing algorithm which uses a neural network to estimate the LEs. Pseudo code that summarizes the three algorithms is shown in Section 9.5. The results of applying the three algorithms to estimate the LEs of six different chaotic systems are tabulated, and the overall results are discussed in Section 9.6. Finally, we present the chapter summary in Section 9.7.

## 9.2 Estimating the LEs from Jacobian matrices

In Section 8.6, we showed that the set of LEs can be computed from the product of Jacobian matrices along the attractor of the system. We said that in order for a system to be chaotic, at least one of its LEs has to be positive. As the number of Jacobian matrices increases, there are inaccuracies that develop as a result of the matrix multiplications. In other words, as the number of multiplied matrices increases, eigen values corresponding to the positive LEs will increase exponentially fast. On the other hand, eigen values corresponding to the negative LEs will decrease exponentially fast. This leads to an over flow in eigen value computations.

In the next two sections, we will present different algorithms for estimating the LEs without directly performing Jacobian matrix multiplications. A standard method that helps in avoiding the direct multiplication of the Jacobian matrices is the QR decomposition, which is explained below.

### 9.2.1 Estimating LE by QR decomposition

To estimate the set of LEs for a system of dimension *d*, we need first to compute the *M* Jacobian matrices $\mathbf{J}_{\mathbf{x}(m)}$ (defined in Equation (8.30)) of the map $\boldsymbol{f}$: $\Re^d \to \Re^d$, where $m = 0, 1, \ldots, M-1$. For the case that $\boldsymbol{f}$ is not known, the Jacobian matrices need to be estimated. Estimation of the Jacobian matrices will be covered in detail later in Sections 9.3 and 9.4.

To estimate the *d* LEs of the system, we start at time $m = 0$ by computing the QR decomposition [EcRu85] of $\mathbf{J}_{\mathbf{x}(0)}$ to produce

$$\mathbf{Q}_1\mathbf{R}_1 = \mathbf{J}_{\mathbf{x}(0)}, \tag{9.1}$$

where $\mathbf{Q}_1$ is an orthonormal matrix, and $\mathbf{R}_1$ is an upper right triangular matrix with diagonal elements that are equal to the eigenvalues of $\mathbf{J}_{\mathbf{x}(0)}$. In the next step, we need to repeat the above process on the product $\mathbf{J}_{\mathbf{x}(1)}\mathbf{Q}_1$ (for time $m = 1$):

$$\mathbf{Q}_2\mathbf{R}_2 = \mathbf{J}_{\mathbf{x}(1)}\mathbf{Q}_1 . \tag{9.2}$$

In general, for time $m-1$, we need to compute the QR decomposition of the product $\mathbf{J}_{\mathbf{x}(m-1)}\mathbf{Q}_{m-1}$:

$$\mathbf{Q}_m\mathbf{R}_m = \mathbf{J}_{\mathbf{x}(m-1)}\mathbf{Q}_{m-1} . \tag{9.3}$$

Notice that by multiplying both sides of Equation (9.1) by $\mathbf{J}_{\mathbf{x}(1)}$, we have

$$\mathbf{J}_{\mathbf{x}(1)}\mathbf{Q}_1\mathbf{R}_1 = \mathbf{J}_{\mathbf{x}(1)}\mathbf{J}_{\mathbf{x}(0)} . \tag{9.4}$$

Substituting $\mathbf{J}_{\mathbf{x}(1)}\mathbf{Q}_1$ from Equation (9.2), into Equation (9.4) gives

$$\mathbf{Q}_2\mathbf{R}_2\mathbf{R}_1 = \mathbf{J}_{\mathbf{x}(1)}\mathbf{J}_{\mathbf{x}(0)} . \tag{9.5}$$

Now we can see that after *m* time steps:

$$\mathbf{Q}_m\mathbf{R}_m\mathbf{R}_{m-1}\cdots\mathbf{R}_1 = \mathbf{J}_{\mathbf{x}(m-1)}\mathbf{J}_{\mathbf{x}(m-2)}\cdots\mathbf{J}_{\mathbf{x}(0)} . \tag{9.6}$$

From Equations (8.34) and (9.6), we can write the product of the *m* Jacobian matrices as:

$$\mathbf{Q}_m\mathbf{R}_m\mathbf{R}_{m-1}\cdots\mathbf{R}_1 = \mathbf{J}_{\mathbf{x}}^m . \tag{9.7}$$

Notice that from Equations (8.36) and (9.7), the LEs of the system can be written as

$$\lambda = \lim_{m\to\infty}\frac{1}{m}\log\left\|\mathbf{Q}_m\mathbf{R}_m\mathbf{R}_{m-1}\cdots\mathbf{R}_1\underline{\delta}(0)\right\| . \tag{9.8}$$

It was proven (see [EcRu85] page 651 and the reference therein) that the *d* LEs can be found from the equation

$$\lambda_i = \lim_{m \to \infty} \frac{1}{m} \sum_{k=1}^{m} \log((\mathbf{R}_k)_{ii}), \tag{9.9}$$

where $(\mathbf{R}_k)_{ii}$ is the $i^{th}$ diagonal element of the matrix $\mathbf{R}_k$, and $i = 1, 2, \ldots, d$. In the next subsection, we show an example of estimating the LEs for a multidimensional chaotic system with a known Jacobian matrix.

9.2.1.1 Example: Estimating LEs of the Henon map

As we have mentioned before, the Henon map is a second order chaotic system (see Equations (2.4) and (2.5)). The Jacobian matrix of the Henon map at $\mathbf{x}(m)$ is

$$\mathbf{J}_{\mathbf{x}(m)} = \left[ \begin{matrix} -2.8x_1 & 1 \\ 0.3 & 0 \end{matrix} \right] \Bigg|_{\mathbf{x} = \mathbf{x}(m)}, \tag{9.10}$$

where $\mathbf{x} = \left[ x_1 \; x_2 \right]^t$. To find the two LEs of the Henon map, we start at time $m = 0$ by evaluating the Jacobian matrix at $\mathbf{x}(0) = [0.5 \; 0.5]^t$ which gives

$$\mathbf{J}_{\mathbf{x}(0)} = \left[ \begin{matrix} -2.8\text{x}0.5 & 1 \\ 0.3 & 0 \end{matrix} \right] = \left[ \begin{matrix} -1.4 & 1 \\ 0.3 & 0 \end{matrix} \right]. \tag{9.11}$$

Using Equation (9.1), we can find the QR decomposition of the matrix $\mathbf{J}_{\mathbf{x}(0)}$ as follows

$$\mathbf{Q}_1\mathbf{R}_1 = \left[ \begin{matrix} -0.98 & 0.21 \\ 0.21 & 0.98 \end{matrix} \right] \left[ \begin{matrix} 1.43 & -0.98 \\ 0 & 0.21 \end{matrix} \right] = \left[ \begin{matrix} -1.4 & 1 \\ 0.3 & 0 \end{matrix} \right] = \mathbf{J}_{\mathbf{x}(0)}. \tag{9.12}$$

By repeating the above process until the last data point, we can find the two LEs of the Henon map from the equation:

$$\lambda_i = \frac{1}{M} \sum_{k=1}^{M} \log((\mathbf{R}_k)_{ii}). \tag{9.13}$$

The estimated LEs of the Henon map when using 10,000 points are $\lambda_2 = 0.4161$ and $\lambda_1 = -1.6201$.

Unfortunately, the Jacobian matrix of the system is generally not known to us, and all we see is a set of scalar measurements $y(m)$ taken from the system. In the next two sections, we present three algorithms for estimating the Jacobian matrices of a chaotic system using $y(m)$. Two of these algorithms are geometric and one is predictive. The estimated Jacobian matrices are then used to estimate the set of LEs for the system. Before we show the three algorithms, let's introduce the concept of spurious exponent.

### 9.2.2 Spurious exponent

We said before in Chapter 3 that by using the embedding theorem, a system whose attractor is in $\Re^k$ can be embedded into a space of dimension $d_E \geq 2d_c + 1$ where $d_c$ is the box-counting dimension of the attractor (see Section 2.3.3.1). We also presented different algorithms used to estimate the minimum embedding dimension ($d_L \leq d_E$), where an embedding map can be found. But if the embedding was into a space of dimension $d$ which is greater than $d_L$, there will be $d - d_L$ spurious exponents [EkRu85, DaBr96]. These exponents are fake, meaning that they were not generated from the dynamics of the system. In other words, they represent numerical artifacts resulting from the lack of knowledge of the exact dimension of the system. This reveals the importance of using a good estimate of $d_L$ before estimating the LEs of the system. In Chapter 6, we presented four new algorithms that can give a good estimate of $d_L$. In the next two sections, we present three different algorithms used to estimate the LEs for a chaotic system.

## 9.3 Estimating LEs by Geometric algorithms

In 1985, Sano and Sawada [SnSd85] and Eckmann and Ruelle [EcRu85] introduced similar algorithms to estimate the LEs of a chaotic system. They do so by using scalar measurements taken from the system. Both algorithms are geometric and use similar orthogonalization techniques to estimate the LEs. In this chapter, we show the Eckmann algorithm as an example. We begin by presenting the Eckmann algorithm, then we present a new algorithm that applies the result of the linear Oseledec theorem.

### 9.3.1 Eckmann's algorithm

Let $\mathbf{y}_d(m) \in \Re^d$ be a delay-vector in the reconstructed space created from the measurements $y(m)$ (see Equation (3.20)). We denote the $N_b$ neighbors of $\mathbf{y}_d(m)$ by $\overset{\frown}{\mathbf{y}}_d^k(m)$ and its time indices by $i_d^k(m)$ where $k = 1, 2, \ldots, N_b$ (see Section 6.2.1). Further, let the perturbation vector from $\mathbf{y}_d(m)$ into its $k^{th}$ neighbor be

$$\underline{\delta}^k(m) = \mathbf{y}_d(m) - \overset{\frown}{\mathbf{y}}_d^k(m) = \mathbf{y}_d(m) - \mathbf{y}_d(i_d^k(m)), \tag{9.14}$$

where $\underline{\delta}^k(m)$ represents the perturbation vector in the direction of the $k^{th}$ neighbor of $\mathbf{y}_d(m)$. After $n$ iterations, $\mathbf{y}_d(m) \to \mathbf{y}_d(m+n)$, and $\mathbf{y}_d(i_d^k(m)) \to \mathbf{y}_d(i_d^k(m)+n)$. The new perturbation vector is $\underline{\delta}^k(m+n) = \mathbf{y}_d(m+n) - \mathbf{y}_d(i_d^k(m)+n)$.

If we assume that the distance between the reference vector and its $N_b$ neighbors at time $m$ is small enough, we can approximate the evolution of the perturbation vectors from

$\underline{\delta}^k(m)$ to $\underline{\delta}^k(m+n)$ by a linear map. That means, we can write the evolution of the perturbation vector $\underline{\delta}^k(m)$ after $n$ time steps as

$$\underline{\delta}^k(m+n) = \mathbf{J}^n_{\mathbf{y}_d(m+n)}\underline{\delta}^k(m), \tag{9.15}$$

where $\mathbf{J}^n_{\mathbf{y}_d(m+n)} = \mathbf{J}_{\mathbf{y}_d(m+n-1)}\mathbf{J}_{\mathbf{y}_d(m+n-2)}\cdots\mathbf{J}_{\mathbf{y}_d(m)}$.

In the next step, the Eckmann algorithm estimates the matrix $\mathbf{J}^n_{\mathbf{y}_d(m+n)}$ by the least squares method. It does so by solving the following equation:

$$(\mathbf{J}^n_{\mathbf{y}_d(m+n)})^t \approx (\mathbf{E}_m\mathbf{E}_m^t)^{-1}\mathbf{E}_m\mathbf{E}_{m+n}^t, \tag{9.16}$$

where $\mathbf{E}_m = \left[\underline{\delta}^1(m)\ \underline{\delta}^2(m)\ \ldots\ \underline{\delta}^{N_b}(m)\right]$. It estimates the Jacobian at each of the following time steps $m = 0, n, 2n, \ldots, U$, where $U = \left\lfloor \frac{M}{n} \right\rfloor$. Next the Eckmann algorithm applies the QR decomposition to the resulting Jacobian matrices, as described in Section 9.2.1. After the QR decomposition is performed, the resulting **R** matrices can be used to compute the LEs by using Equation (9.9).

Actually, Equation (9.9) has to be modified slightly to provide the LEs in continuous time (see [EKRC86]). We need to divide it by the time interval which is $n\tau_s$, where $\tau_s$ is the sampling interval for the original sequence, and $n$ is the number at time steps forward that is used in the Jacobian calculation (see Equation (9.15)). The resulting equation for estimating the LEs is

$$\lambda_i = \frac{1}{n\tau_s U}\sum_{k=1}^{U}\log((\mathbf{R}_k)_{ii}). \tag{9.17}$$

The Eckmann algorithm does not show how to find the dimension of the system *d*. It assumes that *d* is known before estimating the LEs of the system. As we said in Section 9.2.2, if *d* is greater than the minimum embedding dimension of the system ($d_L$), there will be $d - d_L$ spurious exponents among the estimated LEs. Eckmann's algorithm can be improved by starting to estimate $d_L$ of the system before estimating the LEs. In Chapter 6, we presented four new algorithms that can estimate $d_L$. In Section 9.5.1, we will show the pseudo code of the Eckmann algorithm, while in Section 9.6, we will present the results of estimating the LEs by applying this algorithm on different chaotic systems.

After presenting the Eckmann algorithm for estimating the set of LEs, we present the second geometric algorithm, which applies the result of the linear Oseledec theorem to estimate the set of LEs.

### 9.3.2 Linear Oseledec algorithm

We proved in Section 8.9.1 a new theorem (the linear Oseledec theorem) that relates the LEs to the poles of a linear system. We have shown that the LEs are the magnitude of these poles. Now we present a new algorithm that applies the result of the linear Oseledec theorem to estimate the set of LEs for a chaotic system.

This algorithm is called the linear Oseledec algorithm. It is a geometric algorithm that is similar to Eckmann's algorithm. To estimate the LEs of the system, it starts by estimating the Jacobian matrices in the same way as in the Eckmann algorithm. Then it assumes that for a fixed number of time steps (*n*), the eigen values of the Jacobian matrices are the same. Next the algorithm records the logarithm of the magnitude of the eigen values of the Jacobian matrix at the end of these steps (local LEs). By repeating this for the whole

data set, the algorithm computes the LEs of the system by averaging the local LEs resulting from the previous computations. In other words, the linear Oseledec algorithm estimates the Jacobian matrices $\mathbf{J}^{n}_{\mathbf{y}_{d_L}(m+n)}$ (see Equation (9.16)). Then it computes the local LEs from the magnitude of the sorted eigen values of $\mathbf{J}^{n}_{\mathbf{y}_{d_L}(m+n)}$:

$$\lambda_i^m = \left|\lambda_i\left(\mathbf{J}^{n}_{\mathbf{y}_{d_L}(m+n)}\right)\right|. \tag{9.18}$$

By repeating this until the last data point, the algorithm computes the LEs as follows:

$$\lambda_i = \frac{1}{n\tau_s U}\sum_{l=1}^{U}\lambda_i^l. \tag{9.19}$$

In Section 9.5.2, we will present a pseudo code that summarizes the linear Oseledec algorithm. While in Section 9.6 we will show the result of applying this algorithm on six different chaotic systems.

After presenting the two geometric algorithm for estimating the set of LEs of a chaotic system, we present a predictive algorithm.

## 9.4 Estimating LEs by the predictive algorithm

Instead of using the least squares method to approximate the Jacobian matrices (which are then used to estimate the LEs), as in the geometric algorithms, the predictive algorithm approximates the map $\mu: \Re^d \to \Re^1$ in the reconstructed space (see Equation (4.6)). Then it uses the approximated map $\mu$ to approximate the Jacobian matrices. These matrices are subsequently used to estimate the LEs of the system. The idea behind the predictive algorithm comes from the fact that if the embedded system of the delay-vectors

$\mathbf{y}_d(m)$ is equivalent to the original hidden system, we can use $\mathbf{y}_d(m)$ to estimate the LEs of the system (see also Section 4.4).

State evolution in the reconstructed space can be written as

$$\mathbf{y}_d(m+1) = \underline{\phi}(\mathbf{y}_d(m)), \tag{9.20}$$

where $\mathbf{y}_d(m) \in \Re^d$. The Jacobian matrix of the map $\underline{\phi}$ can be computed from Equation (9.20) as follows:

$$(\mathbf{J}_{\mathbf{y}_d(m)})_{(i,k)} = \frac{\partial \phi_i(\mathbf{y}_d(m))}{\partial y_k(m)}, \tag{9.21}$$

where $y_k(m)$ is the $k^{th}$ element of $\mathbf{y}_d(m)$. Recall from Equations (4.10) and (4.11) that the state evolution in the reconstructed space can be written as

$$\mathbf{y}_d(m+1) = \begin{bmatrix} \mu(\mathbf{y}_d(m)) \\ y(m) \\ \vdots \\ y(m-d+2) \end{bmatrix}. \tag{9.22}$$

By comparing Equation (9.22) with Equation (9.20), we can see that

$$\underline{\phi}(\mathbf{y}_d(m)) = \begin{bmatrix} \mu(\mathbf{y}_d(m)) \\ y(m) \\ \vdots \\ y(m-d+2) \end{bmatrix}. \tag{9.23}$$

We can now compute the Jacobian for Equation (9.21) as

$$\mathbf{J}_{\mathbf{y}_d(m)} = \begin{bmatrix} a_1 & a_2 & \dots & a_{d-1} & a_d \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \tag{9.24}$$

where

$$a_i = \frac{\partial y(m+1)}{\partial y_i(m)} = \frac{\partial \mu(\mathbf{y}_d(m))}{\partial y_i(m)}. \tag{9.25}$$

By using a neural network, we can approximate the function $\mu$ (as shown in Section 4.3.2). Figure 9.1 shows a neural network model used for approximating $\mu$. The network has a $d - n_h - 1$ structure. That means the network takes a *d*-dimensional delay-vector as an input, $n_h$ neurons in the hidden layer, and one neuron in the output layer. The network takes *d* previous measurements (delay-vector $\mathbf{y}_d(m)$) to approximate the next measurement in time $y(m+1)$. The hidden layer transfer functions are hyperbolic tangent sigmoid (*tansig*) and the output layer transfer functions are linear (*purelin*).

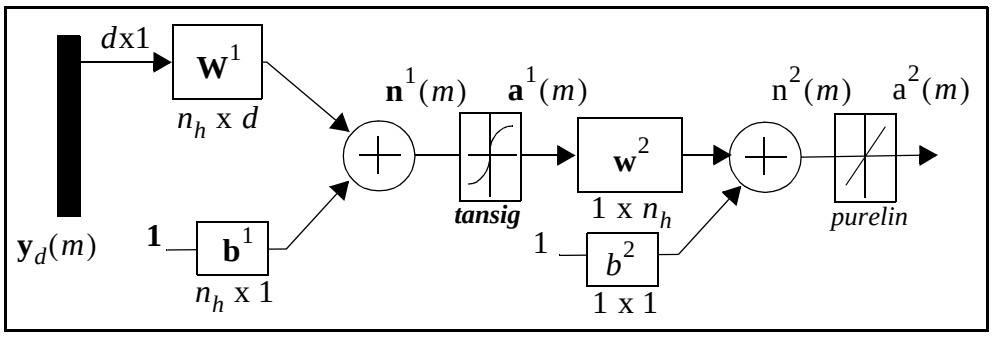



Figure 9.1 The feed forward network used to approximate $\mu$

Assuming that the network has accurately approximated $\mu$, we can use its parameters to approximate the coefficients $a_i$. From Equation (9.25), we can replace the map $\mu$ by its approximation (neural network model) to approximate $\hat{a}_i$:

$$\hat{a}_i = \frac{\partial \hat{y}(m+1)}{\partial y_i(m)}. \tag{9.26}$$

From the neural network model, we can see that the approximated output is

$$\hat{y}(m+1) = b^2 + \mathbf{w}^2 \mathbf{a}^1(m), \tag{9.27}$$

where $\mathbf{w}^2 = \begin{bmatrix} w_1^2 & w_2^2 & w_{n_h}^2 \end{bmatrix}$. Notice that we can write Equation (9.27) as follows

$$\hat{y}(m+1) = b^2 + \mathbf{w}^2 \mathbf{f}^1(\mathbf{W}^1 \mathbf{y}_d(m) + \mathbf{b}^1). \tag{9.28}$$

Taking the derivative of $\hat{y}(m+1)$ with respect to $y_i(m)$ produces the coefficients:

$$\hat{a}_i = \frac{\partial}{\partial y_i(m)}(b^2 + \mathbf{w}^2 \mathbf{f}^1(\mathbf{W}^1 \mathbf{y}_d(m) + \mathbf{b}^1)). \tag{9.29}$$

Equation (9.29) can be written as

$$\hat{a}_i = \mathbf{w}^2 \frac{\partial}{\partial y_i(m)} \mathbf{f}^1(\mathbf{W}^1 \mathbf{y}_d(m) + \mathbf{b}^1), \tag{9.30}$$

or

$$\hat{a}_i = \mathbf{w}^2(\mathbf{w}_k^1 \cdot \dot{\mathbf{f}}^1(\mathbf{W}^1 \mathbf{y}_d(m) + \mathbf{b}^1)), \tag{9.31}$$

where $\mathbf{w}_k^1 = \begin{bmatrix} w_{1,k}^1 & w_{2,k}^1 & w_{n_h,k}^1 \end{bmatrix}^t$ which is the $k^{th}$ column of the weights matrix $\mathbf{W}^1$. Notice that the multiplication $\mathbf{w}_k^1 \cdot \dot{\mathbf{f}}^1(\ )$ is point wise. Since we used the *tansig* transfer function in the hidden layer, Equation (9.31) can be written as

$$\hat{a}_i = \mathbf{w}^2(\mathbf{w}_k^1 \cdot (\mathbf{1} - (\mathbf{f}^1(\mathbf{n}^1))^2)), \tag{9.32}$$

see [Hag95].

After approximating the coefficients $a_i$, we can use them to approximate the Jacobian matrix $\mathbf{J}_{\mathbf{y}_d(m)}$ in Equation (9.24). Next the predictive algorithm applies the QR decomposition on the approximated Jacobian matrices to estimate the set of LEs (see Section 9.2.1).

To apply the predictive algorithm for estimating the LEs, it starts by sampling the measurements $y(m)$ at an interval $T$ (see Section 6.2) to produce $y_s(m)$ (see Equation (6.16)). Next the predictive algorithm creates the new delay-vector $\mathbf{y}_d^s(m)$ (see Equation (6.18)). The mean $y_s(m)$ has to be deducted from it to insure that the input to the network is a zero mean. This step creates the signal $s(m)$ (see Equation (6.17)).The network is presented with $d$ previous values of $s(m)$ and trained to predict the next value in time.

The original predictive algorithm was introduced by L. Djamai and P. Coirault [DjCo02]. We improved their algorithm in four ways: i) we used the delay-vectors $\mathbf{y}_d(m)$ instead of the original states of the system, ii) we estimated the minimum embedding dimension $d_L$ of the delay-vector by using our algorithms that we presented in Chapter 6, iii) we repeat the training process a few times then choose the network with the minimum SSE, and iv) we simplified the Jacobian matrix approximation into the form shown in Equation (9.24). The third Improvement is required to insure that the network has converged to the global, rather that the local minima.

In Section 9.5.3, we will present a pseudo code that summarizes the predictive algorithm. The results of applying the predictive algorithm on six different chaotic systems will be presented in Section 9.6.

## 9.5 Pseudo codes of the LE estimation algorithms

In Section 9.3, we presented two geometric algorithms used for estimating the set of LEs. In Section 9.4, we presented the predictive algorithm for the same purpose. In this section, we show three pseudo codes that summarize these algorithms.

### 9.5.1 Pseudo code of the Eckmann algorithm

To improve the Eckmann algorithm (see Section 9.3.1) for noisy signals, we can use the method suggested by Zeng *et al* [ZxEyPi91]. In this paper, the authors use a shell around the reference point (rather than a sphere as suggested by Eckmann). The shell has a minimum and a maximum radius. This method reduces the effect of noise by eliminating neighbors that are very close to the reference point from the Jacobian computations. These neighbors could actually be noise signals. If the signal is noise free, the minimum radius can be set to zero, which is similar to Eckmann's original algorithm. Figure 9.2 below shows a pseudo code that summarizes the Eckmann algorithm.

{Pseudo code of the Eckmann algorithm

•*Choose n* (forward steps)

•*Choose* $r_{min}$ and $r_{max}$ (minimum and maximum neighbor distances)

•*Estimate* $d_L$ (use the algorithms presented in Chapter 6)

•*Choose* $N_b$ (number of neighbors of the reference point)

•*Compute T* (see Section 6.2)

•*Initialize* **Q** and **R** to be $d_L$ x $d_L$ identity matrices

•*Create* the delay vectors $\mathbf{y}_{d_L}(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d_L-1)T)\right]^t$, $m = 0, 1, \ldots, M-1$

•*For m=0:step n: M*

•*Compute* the distances between $\mathbf{y}_{d_L}(m)$ and the other points.

•*Save* the $N_b$ neighbors of $\mathbf{y}_{d_L}(m)$ with distances between $r_{min}$ and $r_{max}$ in the matrix

$$\mathbf{Y}_{d_L}(m) = \left[\ \mathbf{y}_{d_L}(i^1_{d_L}(m))\ \mathbf{y}_{d_L}(i^2_{d_L}(m))\ \ldots\ \mathbf{y}_{d_L}\left(i^{N_b}_{d_L}(m)\right)\ \right].$$

•*Compute* the perturbation matrix:

$$\mathbf{E}_m = \left[\mathbf{y}_{d_L}(m)-\mathbf{y}_{d_L}(i^1_{d_L}(m))\quad \mathbf{y}_{d_L}(m)-\mathbf{y}_{d_L}(i^2_{d_L}(m))\ \ldots\ \mathbf{y}_{d_L}(m)-\mathbf{y}_{d_L}\left(i^{N_b}_{d_L}(m)\right)\right].$$

(Propagate $\mathbf{y}_{d_L}(m)$ and $\mathbf{y}_{d_L}(i^k_{d_L}(m))$ *n* time steps ahead and compute the new perturbation matrix $\mathbf{E}_{m+n}$)

•The $k^{th}$ element of $\mathbf{E}_{m+n}$ is $(\mathbf{E}_{m+n})_k = \mathbf{y}_{d_L}(m+n)-\mathbf{y}_{d_L}(i^k_{d_L}(m)+n)$, where $k = 1, 2, \ldots, N_b$

•Use $\mathbf{E}_m$ and $\mathbf{E}_{m+n}$ to *estimate* the Jacobian matrix:

$$\left(\mathbf{J}^n_{\mathbf{y}_{d_L}(m+n)}\right)^t = (\mathbf{E}_m\mathbf{E}_m^t)^{-1}\mathbf{E}_m\mathbf{E}_{m+n}^t$$

•*Normalize* and *reorthogonalize* $\mathbf{J}^n_{\mathbf{y}_{d_L}(m+n)}$ by the QR decomposition

$$\mathbf{QR}_m = \mathbf{J}^n_{\mathbf{y}_{d_L}(m+n)}\mathbf{Q}$$

•*Multiply* the triangular matrices:

$$\mathbf{R} = \mathbf{R}_m\mathbf{R}$$

•*end m*

(The estimated LEs are the mean of the logarithm of the magnitude of the diagonal elements of **R**)

•$\lambda_i = \frac{1}{n\tau_s U}\log(\mathbf{R}_{ii})$, where $i = 1, 2, \ldots, d_L$, and $U = \left\lfloor \frac{M}{n} \right\rfloor$

}

Figure 9.2 Pseudo code for the Eckmann algorithm

### 9.5.2 Pseudo code of the linear Oseledec algorithm

The linear Oseledec algorithm (see Section 9.3.2) is similar to the Eckmann algorithm, but here we do not multiply matrices. To compute the LEs using the linear Oseledec algorithm, we compute the local LEs first. The next step is to estimate the LEs of the system by averaging these local exponents. Figure 9.3 shows the linear Oseledec algorithm pseudo code.

{Pseudo code of the linear Oseledec algorithm

•*Choose n* (forward steps)

•*Choose* $r_{min}$ and $r_{max}$ (minimum and maximum neighbor distances)

•*Estimate* $d_L$ (use the algorithms presented in Chapter 6)

•*Choose* $N_b$ (number of neighbors of the reference point)

•*Compute T* (see Section 6.2)

•*Create* the delay vectors $\mathbf{y}_{d_L}(m) = \left[y(m)\ y(m-T)\ \ldots\ y(m-(d_L-1)T)\right]^t$, $m = 0, 1, \ldots, M-1$

•*For m=0: step n: M-1*

•*Compute* the distances between $\mathbf{y}_{d_L}(m)$ and the other points.

•*Save* the $N_b$ neighbors of $\mathbf{y}_{d_L}(m)$ with distances between $r_{min}$ and $r_{max}$ in the matrix

$$\mathbf{Y}_{d_L}(m) = \left[\ \mathbf{y}_{d_L}(i^1_{d_L}(m))\ \mathbf{y}_{d_L}(i^2_{d_L}(m))\ \ldots\ \mathbf{y}_{d_L}\left(i^{N_b}_{d_L}(m)\right)\ \right].$$

•*Compute* the perturbation matrix:

$$\mathbf{E}_m = \left[\mathbf{y}_{d_L}(m) - \mathbf{y}_{d_L}(i^1_{d_L}(m))\quad \mathbf{y}_{d_L}(m) - \mathbf{y}_{d_L}(i^2_{d_L}(m))\ \ldots\ \mathbf{y}_{d_L}(m) - \mathbf{y}_{d_L}\left(i^{N_b}_{d_L}(m)\right)\right].$$

(Propagate $\mathbf{y}_{d_L}(m)$ and $\mathbf{y}_{d_L}(i^k_{d_L}(m))$ $n$ time steps ahead and compute the new perturbation matrix $\mathbf{E}_{m+n}$)

•The $k^{\text{th}}$ element of $\mathbf{E}_{m+n}$ is $(\mathbf{E}_{m+n})_k = \mathbf{y}_{d_L}(m+n) - \mathbf{y}_{d_L}(i^k_{d_L}(m)+n)$, where $k = 1, 2, \ldots, N_b$

•Use $\mathbf{E}_m$ and $\mathbf{E}_{m+n}$ to *estimate* the Jacobian matrix:

$$\left(\mathbf{J}^n_{\mathbf{y}_{d_L}(m+n)}\right)^t = (\mathbf{E}_m\mathbf{E}_m^t)^{-1}\mathbf{E}_m\mathbf{E}_{m+n}^t$$

(Compute the local LEs)

$$\bullet\lambda_i^m = \log\left|\lambda_i\left(\mathbf{J}^n_{\mathbf{y}_{d_L}(m+n)}\right)\right|$$

•*end m*

(The estimated LEs are the average of $\lambda_i^m$

$$\bullet\lambda_i = \frac{1}{n\tau_s U}\sum_{l=0}^{U-1}\lambda_i^l, \text{ where } i = 1, 2, \ldots, d_L, \text{ and } U = \left\lfloor\frac{M}{n}\right\rfloor$$

}

Figure 9.3 Pseudo code for the linear Oseledec algorithm

### 9.5.3 Pseudo code of the predictive algorithm

Now that we have shown the pseudo codes of the two geometric algorithms, we present the pseudo code of the predictive algorithm (see Section 9.4). Figure 9.4 shows a pseudo code that summarizes the predictive algorithm.

{Pseudo code of the predictive algorithm
•*Estimate* $d_L$ (use the algorithms presented in Chapter 6)
•*Choose* $n_h$ (number of hidden layer neurons)
•*Compute T* (see Section 6.2)
•*Create* $y_s(m) = y(1 + (m-1)T)$ sample the measurements. *Then* compute it's mean $\bar{y}_s$
•*Create* $s(m) = y_s(m) - \bar{y}_s$ to insure a zero mean input
•*Set* $Itr_{max}$ (maximum number of trials to train the network)
•*Initialize* **Q** and **R** to be $d_L$ x $d_L$ identity matrices
•*For Iteration = 1:* $Itr_{max}$
    •*Initialize* the network parameters: *Net*
    •*Train* the network with the input $\left[s(m-1)\ s(m-2)\ \ldots\ s(m-d_L)\right]$ to predict the output $s(m)$. Do that until the network reaches its minimum SSE
    (*Record* the minimum SSE as a function of Iteration number and save the resulting network parameters)
    • SE(Iteration) = SSE
    •Network (Iteration) = *Net*
•*end Iteration*
(Find the index of the minimum SSE)
• Ind = $Index(minimum(\text{SE}))$
(Choose the network parameters of the index " Ind ")
•$Net_{\text{Opt}}$ = Network( Ind )
•*Read* the network $Net_{\text{Opt}}$ parameters $\mathbf{W}^2$, $\mathbf{W}^1$, and $\mathbf{b}^1$
•*For m= 1: U,* $U = \left\lfloor \frac{M}{T} \right\rfloor$
    •*For k* = 1:$d_L$
        • $a(k) = \mathbf{w}^2(\mathbf{w}_k^1 \cdot (\mathbf{1} - (Tansig(\mathbf{W}^1 s(m) + \mathbf{b}^1))^2))$, fill up the Jacobian matrix $\mathbf{J}_{\mathbf{y}_{d_L}(m)}$ elements as shown in Equation (9.24)
    •*end k*
    (Perform the QR decomposition)
    • $\mathbf{QR}_m = \mathbf{J}_{\mathbf{y}_{d_L}(m)}\mathbf{Q}$
    (Multiply the triangular matrices)
    • $\mathbf{R} = \mathbf{R}_m\mathbf{R}$
•*end m*
(The estimated LEs are the mean of the logarithm of **R**
• $\lambda_i = \frac{1}{T\tau_s U}\log(\mathbf{R}_{ii})$, $i = 1, 2, \ldots, d_L$ }

Figure 9.4 The predictive algorithm pseudo code

## 9.6 Results of estimating the LEs by using the three algorithms

In the previous three sections, we presented three algorithms for estimating the set of LEs for a chaotic system. These algorithms use a set of scalar measurements taken from the system do the estimation. Two of these algorithms are geometric and one is predictive. We also showed pseudo codes that summarize these algorithms.

In this section, we list the results of the estimated LEs found by applying these algorithms on six different chaotic systems (see Section 7.2). The first column of Table 9.1 shows six different chaotic systems that we used to test the three algorithms: the Eckmann, the predictive, and the linear Oseledec (the last three columns). The first column also shows the sampling time $\tau_s$ of the measurements $y(m)$ and the computed delay-time *T*. The first row of each cell in the second through the fifth columns of the table shows the LE values, while the second row shows the Lyapunov dimension (see Equation (8.23)). For example, we can see that the estimated LEs for the Lorenz model by using the Eckmann algorithm (in second row, third column of the table) are 1.31, -0.03, and -7.8, while the computed Lyapunov dimension is 2.16. The Jacobian matrices of the first three systems are known, while the Jacobian matrices of the next three systems are not known. The set of LEs for the first three systems were computed from the Jacobian matrices by using the Wolf *el al* algorithm [WSSV85]. These values are shown in the first three cell in the second column of the table.

| Chaotic system | Original LEs and Lyapunov dimension | Eckmann alg. | Predictive alg. | Linear Oseledec alg. |
|---|---|---|---|---|
| **Lorenz model**<br>Sampling time = 0.01<br>Delay-Time $T$ = 8, $d_L$ = 3 | {1.34,0.00,-22.29}<br>2.06 | {1.31,-0.03,-7.8}<br>2.16 | {1.33,-0.15,-19.27}<br>2.06 | {4.38,-0.17,-14.02}<br>2.3 |
| **Chaotic circuit**<br>Sampling time = 0.1<br>Delay-Time $T$ = 8, $d_L$ = 3 | {0.35, -0.01, -1.09}<br>2.3 | {0.36, -0.05, -1.03}<br>2.3 | 0.4, -0.00, -1.1<br>2.37 | {0.36, 0.33, -1.39}<br>2.5 |
| **Rossler model**<br>Sampling time = 0.12<br>Delay-Time $T$ = 5, $d_L$ = 3 | {0.07, 0.02, -5.4}<br>2.01 | {0.09, -0.00, -1.12}<br>2.08 | {0.07, 0.00, -5.99}<br>2.01 | {0.07, 0.71, -1.37}<br>2.57 |
| **Santa Fe comp. data set *A***<br>Sampling time = 0.1<br>Delay-Time $T$ = 2, $d_L$ = 3 | Not known | {0.93, -0.21, -9.66}<br>2.07 | {0.66, -0.31, -19.3}<br>2.01 | {1.12, -0.01, -5.2}<br>2.21 |
| **Santa Fe comp. data set $B_1$**<br>Sampling time = 0.08<br>Delay-Time $T$ = 6, $d_L$ = 4 | Not known | {0.56, -0.07,-0.56, -1.1 }<br>2.9 | {-0.55, -1.3, -1.8, -2.4}<br>0 | {0.56, 0.16, -0.43, -1.47}<br>3.19 |
| **Santa Fe comp. data set $D_1$**<br>Sampling time = 0.05<br>Delay-Time $T$ = 3, $d_L$ = 9 | Not known | {1.33, 0.6, 0.13, -0.27, -0.64, -0.94 , -1.4 , -2.3 , -4.9}<br>6.14 | {0.48, -0.18, -0.6, -1.04, -1.2, 1.66, -2.0, -2.78, -5.0}<br>2.48 | {2.27, 1.26, 0.37, -0.07, -0.55, -1.0, -1.6, -2.8, -6.4}<br>7.23 |

Table 9.1 LEs estimation results using the three algorithms. The numbers inside the curly parenthesis are the estimated LEs and the number in the bottom of the cell is the Lyapunov dimension ($d_{Lyp}$).

### 9.6.1 Discussion of the results

From Table 9.1, we can see that the Eckmann algorithm gave good estimates of the LEs for the chaotic circuit and the Santa Fe data sets. The estimate of the smallest LEs for the Rossler model was not perfect. For the predictive algorithm, we can see that it gave good estimates of the LEs for the Lorenz model, the Rossler model, the chaotic circuit, and the *A* Santa Fe data set, while it fails to find good estimates of the LEs for the $B_1$ and $D_1$ Santa Fe data sets. Notice also that the predictive algorithm estimates of the LEs for the Lorenz model were better than those found by the Eckmann algorithm. On the other hand, we can see that the linear Oseledec algorithm gave good estimates of the LEs for the Rossler model, the chaotic circuit, the $B_1$ and $D_1$ Santa Fe data sets, while it gave a poor estimate of the largest LE of the Lorenz model.

In general, we can see that the Eckmann algorithm usually gives good estimates of the LEs. While the predictive algorithm gave good estimates of the LEs for signals with high SNR. The linear Oseledec algorithms on the other hand, gave good estimates in four

cases. This algorithm might be improved if one uses a better estimator than least squares. Finally, by running more than one algorithm to estimate the LEs, one can have confidence in the estimate values if the results are in agreement with each other.

## 9.7 Chapter summary

In this chapter, we have explored the estimation of LEs (model parameters) by using three different algorithms. Full details of the algorithms were presented, and pseudo codes that summarize these algorithms were illustrated. The three algorithms were tested by using six different chaotic systems. A table summarizing the results was presented and conclusions were derived from it. In the next (final) chapter of the dissertation, we will give a summary of the previous chapters, draw final conclusions on modeling chaotic systems, and discuss future recommendations.

# CHAPTER 10

# SUMMARY, CONCLUSIONS, AND FUTURE RECOMMENDATIONS

## 10.1 Summary

In this dissertation, we have explored modeling of chaotic systems. The modeling process uses measurements taken from a chaotic system to find its model order and model parameters. The model order is the minimum embedding dimension of the system ($d_L$), while the model parameters are its Lyapunov exponents (LEs).

In Chapters 3 through 7, we discussed estimating $d_L$. We gave full details of four new algorithms used to estimate the value of $d_L$. Implementation of the algorithms on nine chaotic systems was also discussed. Among the four algorithms, three are geometric algorithms: the CND, the $\mathrm{CDD}_G$, and the $\mathrm{CDT}_G$. They estimate $d_L$ by detecting the existence of FNNs. The CND algorithm detects the existence of FNNs by checking to see if the nearest neighbors in the space of dimension $d$ remain neighbors in dimension $d + 1$. On the other hand, the $\mathrm{CDD}_G$ algorithm detects the existence of FNNs by checking to see if the distance between the nearest neighbors in dimension $d$ will increase significantly as the dimension increases to $d + 1$. Finally, the $\mathrm{CDT}_G$ algorithm detects the existence of FNNs by checking to see if the distance between the nearest neighbors in dimension $d$ will change significantly as time increases. The three geometric algorithms use a global neighbor search

method to search for the nearest neighbors. In the fourth algorithm (the predictive), the estimation of $d_L$ is done by approximating the function $\mu: \Re^d \rightarrow \Re^1$ which operates on the reconstructed attractor. $\mu$ is approximated by using a neural network with a Tapped Delay Line connected to its input. As the number of taps in the TDL ($d$) increases, the prediction error decreases. At one point, further increase of $d$ does not improve the prediction error. At this point, $d_L$ is found.

We have demonstrated estimating $d_L$ by applying the four algorithms on different chaotic systems. The results derived from these algorithm gave confidence in the estimated $d_L$. Conclusions reached from the estimation results are summarized in the next section.

In Chapter 8, we presented some theoretical background on Lyapunov Exponents. We also proved a new theorem that relates the LEs to the poles of a linear system. Estimating the LEs of a chaotic system was explored in Chapter 9. We presented three different algorithms used to estimate the LEs (two of these algorithms are new). The three algorithms approximate the Jacobian matrices of the chaotic model. These matrices are subsequently used to estimate the LEs. The first algorithm is the Eckmann algorithm. This algorithm is a standard algorithm for estimating the LEs. It uses the least squares method to approximate the Jacobian matrices. These matrices are then used to estimate the LEs. The second algorithm which is also used for estimating the LEs is the linear Oseledec algorithm. This algorithm approximates the Jacobian matrices by using the least squares method. Then it applies the results of the linear Oseledec theorem to the approximated Jacobian matrices to estimate the LEs. The third algorithm is the predictive algorithm. This algorithm uses a feed forward neural network to approximate the Jacobian matrices.

The three algorithm used for estimating the LEs were tested on six chaotic systems, and conclusions were derived from the results. These conclusions are summarized in the next section. In total, we introduced four new algorithms to estimate $d_L$, and two new algorithms to estimate the LEs. These algorithms were tested on different chaotic systems. These systems are different in dimension and noise content.

## 10.2 Conclusions

We list below the main conclusions found from this research. We begin with conclusions related to the algorithms for estimating the model order.

1) The predictive algorithm gave the best estimate of $d_L$ as long as the SNR is not too low.
2) The global neighbor search algorithms introduced in this research gave a better estimate of $d_L$ than the local neighbor search algorithms.
3) The use of more than one algorithm to estimate $d_L$ increases our confidence in the estimated value.

Now we summarize the main conclusions related to the algorithms used to estimate the LEs.

1) The Eckmann algorithm usually provides good estimates of the LEs.
2) The predictive algorithm gives good estimates of the LEs for signals with high SNR.
3) The linear Oseledec algorithm gives good estimates of the LEs in some cases.
4) The best approach to estimating the LEs is to use several algorithms and check for consistency in the results.

## 10.3 Future recommendations

For further improvement of the results of this research, we recommend the following:

1) Research on filtering out the noise before applying the modeling algorithms to the signals. The problem that may arise is that the filter may smear out the attractor and cause some dynamical features to be lost. Work needs to be done to determine the optimal filter.
2) Find a better estimator of the Jacobian matrix than least squares. By doing this, both the Eckmann algorithm and the linear Oseledec algorithm may give a more accurate results.
3) Experiment with different types of neural networks to improve the predictive algorithm for estimating the model order and for estimating the LEs. An example of one possible network is the radial basis network.

# REFERENCES


[Aba98] H. Abarbanel, "Obtaining Order in a World of Chaos," *IEEE Signal Processing Magazine*, pp. 49-65, 1998.

[AbKe93] H. Abarbanel and M. Kennel, "Local False Nearest Neighbors and Dynamical Dimensions from Observed Chaotic Data," *Physical Review E*, Vol. 47, pp. 3057-3068, 1993.

[Arn98] L. Arnold, *Random Dynamical Systems*, Princeton University Press, 1998

[ASY96] K. Alligood, T. Sauer and J. Yorke, *Chaos an Introduction to Dynamical Systems*, Springer, 1996.

[Bay96] W. Baylis, *Clifford (Geometric) Algebra*, Birkhauser, 1996.

[Cao97] L. Cao, "Practical Method for Determining the Minimum Embedding Dimension of a Scalar Time Series," *Physica D*, 110 pp. 43-50, 1997.

[DaBr96] A. Darbyshire and D. Broomhead, "Robust Estimation of Tangent Maps and Liapunov Spectra," *Physica D*, pp. 287-305, 1996.

[Dav99] B. Davis, *Exploring Chaos*, Perseus Books, 1999.

[DjCo02] L. Djamai and P. Coirault, "Estimation of Lyapunov Exponents by Using the Perceptron," *Proceedings of the American Control conference Anchorage, Ak. May 8-10*, pp.5150-5155, 2002.

[EcRu85] J. Eckmann and D. Ruelle, "Ergodic Theory of Chaos and Strange Attractors," *Review of Modern Physics*, Vol.57, No. 3, Part 1, pp. 617-656, 1985.

[EKRC86] J. Eckmann, O. Kamphorst, D. Ruelle, and S. Ciliberto," Liapunov exponents from time serires," *Physical Review A*, Vol 34, Num 6, pp.4971-4979, 1986.

[Far81] J. Farmer, "Chaotic Attractors of Infinite Dimensional systems," *Physica 4D*, pp. 366-393, 1981.

[Fra89] A. Fraser, *Information Theory and Strange Attractors*, Ph.D. thesis, University of Texas at Austin, May 1989.

[FS86] A. Fraser and H. Swinney, "Independent Coordinates for Strange Attractors from Mutual Information," *Physical Review A*, Vol. 33 pp. 1134-1140, 1986.

[GGS83] L. Glass, X. Guevan and A. Shrier, "Bifurcation and Chaos in Periodically Stimulated Cardiac Oscillator," *Physica 7D*, pp. 89-101, 1983.

[GoVa96] G. Golub and C. Van Loan, *Matrix Computations,* The Johns Hopkins University Press, 3rd Ed. 1996.

[GPL90] K. Geist, U. Parlitz and W. Lauterborn, "Comparison of Different Methods for Computing Lyapunov Exponents," *Progress of theoretical physics* Vol. 83, No. 5, pp. 875-893, 1990.

[Hag95] M. Hagan, H. Demuth and M. Beale, *Neural Network Design,* PWS, 1995.

[Hak98] S. Haykin, "Making Sense of a Complex World," *IEEE signal Processing Magazine*, pp.66-81, 1998.

[HAW89] N. Hubner, N. Abraham C. and Weiss, "Dimension and Entropies of Chaotic Intensity Pulsations in a Single-Mode Far-Infrared $NH_3$ Laser," *Physical Review A*, Vol. 40, No. 11, pp. 6354-6365, 1989.

[Hen76] M. Henon, "A Two Dimensional Mapping with a Strange Attractor," *Comm. Math. Phy*., Vol. 50, pp. 69-77, 1976.

[Hol86] A. Holden, *Chaos,* Princeton University Press, 1986.

[KaSc00] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis,* Cambridge, 2000.

[KaYo79] J. Kaplan and J. Yorke, "Chaotic Behavior in Multidimensional Difference Equations," *In H.-O. Peigen and H.-O. Walter, editors, functional Di erential Equations and Approximation of fixed Points*, pp. 204-227, Springer, Berlin, 1979.

[KBA92] M. Kennel, R. Brown and H. Abarbanel, "Determining Embedding Dimension for Phase Space Reconstruction Using a Geometric Construction," *Physical Review A*, pp. 3403-3411, 1992.

[Kre98] R. Kress, *Numerical Analysis,* Graduate text in mathematics, Springer, 1998.

[Lo63] E. Lorenz, "Deterministic Non-periodic Flow," *Journal of Atmospheric Science*, Vol. 20, pp. 130-141, 1963.

[MG77] M. Mackey and L. Glass, "Oscillation and Chaos in Physiological Control Systems," *Science*, Vol. 197, p. 287, 1977.

[Moo92] F. Moon, *Chaotic and Fractal Dynamics,* Wiley Interscience, 1992.

[MPL00] C. Merkwirth, U. Parlitz and W. Lauterborn, "Fast Nearest-Neighbor Searching for Nonlinear Signal processing," *Physical Review E,* Vol. 62, Num. 2, pp. 2089-2097, 2000.

[Nak90] M. Nakahara, *Geometry, Topology and Physics*, IOP, 1990.

[Ose68] V. Oseledec, "A Multiplicative Ergodic Theorem. Ljapunov Characteristic Numbers for Dynamical Systems," *Trans. Moscow Math Society,* Vol. 19, pp. 197-231, 1968.

[OSY94] E. Ott, T. Sauer and J. Yorke, *Coping with Chaos,* Wiley Interscience, 1994.

[Par92] U. Parlitz, “Identification of True and Spurious Lyapunov Exponents from Time Series,” *Intern. Jour. of chaos and Bifur.*, Vol. 2, No1, pp. 155-165, 1992.

[PJS92] H. Peitgen, H. Jurgens and D. Saupe, *Chaos and Fractals New Frontier of Science*, Springer, 1992.

[Ros76] O. Rossler, “An Equation for Continuous Chaos,” *Physical Letters A*, Vol. 57, p. 397, 1976.

[RVRDV92] V. Rulkov, R. Volkovskii, A. Rodriguez-Lozano, E. DelRio and M. Velarde, “Mutual synchronization of Chaotic Self-Oscillators with Dissipative Coupling,” *International Journal of Bifurcation and Chaos*, pp. 669-676, 1992.

[SnSd85] M. Sano and Y. Sawada, “Measurements of the Lyapunov Spectrum from a Chaotic Time Series,” *Physical Review Letters*, Vol. 55, No. 10, pp. 1082-1085, 1985.

[SYC91] T. Sauer, J. Yorke and M. Casdagli, “Embedology,” *Journal of Statistical Physics*, Vol. 65, pp. 579-616, 1991.

[Tem88] D. Temam, *Infinite Dimensional Dynamical Systems in Mechanics and Physics*, Springer-Verlag, Berlin Heidelberg New York, 1988.

[TrBa97] L. Trefethen and D. Bau, III, *Numerical Linear Algebra*, SIAM, 1997.

[WeGe95] A. Weigend and N. Gershenfeld, *Time Series Prediction*, Addison Wesley, 1995.

[WSSV85] A. Wolf, J. Swift, H. Swinney, and J. Vastano, “Determining Lyapunov Exponents from a Time Series,” *Physica 16D*, pp. 285-317, 1985.

[ZxEyPi91] X. Zeng, R. Eykholt, and R. Pielke, “Estimating the Lyapunov-Exponents Spectrum from Short Time series of Low Precision,” *Physical Review Letters*, Vol. 66, No. 25, pp. 3229-3232, 1991.

[Zyl01] J. Zyl, *modeling Chaotic Systems with Neural Networks: Application to Seismic Event Predicting in Gold Mines*, Ms. Thesis, University of Stellenbosch, 2001.

# VITA

Khaled Marzoug Al-Mughadhawi

Candidate for the Degree of

Doctor of Philosophy

Thesis: MODELING CHAOTIC SYSTEMS

Major Field: Electrical and Computer Engineering

Biographical:

Personal Data: Born in Madina, Saudi Arabia, on October 16, 1966, the son of Marzoug Saleh Al-Mughadhawi, and Mohelah Sa'ad Al-Mozaini.

Education: Graduated from Taybah high school, Madina 1982; received Bachelor of Science degree in Electrical Engineering from King Fahad University of Petroleum and Minerals, Dhahran, Saudi Arabia in 1989; received Master of Science degree in Electrical Engineering from Southern Methodist University, Dallas, Texas in 1999. Completed the requirements for the Doctor of Philosophy degree in Electrical Engineering at Oklahoma State University in July 2005.

Experience: Employed by Ministry of health in Saudi Arabia as a maintenance engineer 1989. Tough in the technical high school in Madina, Saudi Arabia, for the General Organization of Technical Education and Vocational Training since 1992.

Name: Khaled M. Al-Mughadhawi, Date of Degree: July, 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: MODELING CHAOTIC SYSTEMS

Pages in Study: 168 Candidate for the Degree of Doctor of Philosophy

Major Field: Electrical and Computer Engineering

Scope and Method of Study: The purpose of this research is to model chaotic systems. From a set of scalar measurements taken from a chaotic system, we want to describe the hidden states of the system. The modeling process has two parts: determining the model order, and determining the model parameters. The major characteristic of chaotic systems is their sensitivity to changes in initial conditions. This characteristic limits the ability to make a long term prediction in these systems. To be able to make an accurate model, we need to accurately estimates of the model parameters, which requires an accurate estimate of the model order.

Findings and Conclusions: In this research we found four new algorithms to estimate the model order. These algorithms were tested on different chaotic systems. Three of these algorithms are geometric and one is predictive. We checked on the ability of these algorithms to make accurate estimates of the model order. We also proved a new theorem that relates the Lyapunov exponents (model parameters) to the linear system poles. We introduced a new algorithm that implement the result of this theorem to estimate the model parameters of a chaotic model. Beside this algorithm, we improved an existing algorithm that uses a neural network to estimate the model parameters. Both algorithms were tested on different chaotic systems. Our finding with respect to the algorithms used to estimate the model order can be summarized in the following: i) the predictive algorithm gave the best estimate of model order as long as the signal to noise ratio is not too low, ii) the global neighbor search algorithms proposed in this research gave better estimate of model order than the local neighbor search algorithms, iii) the use of more than one algorithm to estimate model order increases our confidence in the estimated value. With respect to the algorithms used to estimate the model parameters, the main conclusions are: i) the predictive algorithm gives good estimate of the model parameters for signals with high signal to noise ratio, ii) the algorithm that implement the new theorem result gives good estimate of the model parameters in some cases, and iii) when using more than one algorithm to estimate the model parameters, one can have confidence in the estimate values if they are in agreement with each other.

ADVISER'S APPROVAL: Martin T. Hagan